

WebSphere MQ for HP OpenVMS



System Administration Guide

Version 6.0

WebSphere MQ for HP OpenVMS



System Administration Guide

Version 6.0

Note

Before using this information and the product it supports, read the information in "Notices" on page 341 at the back of this book.

This edition applies to WebSphere MQ for HP OpenVMS, Version 6.0 and to all subsequent releases and modifications until otherwise indicated in new editions.

© **Copyright International Business Machines Corporation 1994, 2009.**

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Figures ix

Tables xi

Chapter 1. Introduction to WebSphere

MQ 1

WebSphere MQ and message queuing 1

Time-independent applications 1

Message-driven processing 1

Messages and queues 2

What is a message? 2

What is a queue? 2

Objects 3

Object names 4

Managing objects 4

Object attributes 4

WebSphere MQ queues 5

WebSphere MQ queue managers 8

Process definitions 8

Clusters 8

Namelists 9

Authentication information objects 9

Channels 9

Client connection channels. 9

Listeners 10

Services 10

System default objects 10

Clients and servers 11

WebSphere MQ applications in a client-server environment 11

Extending queue manager facilities 11

User exits 12

API exits 12

Installable services 12

Security 13

Object Authority Manager (OAM) facility 13

User-written or third party channel exits 13

Channel security using SSL 13

Transactional support 13

Chapter 2. An introduction to

WebSphere MQ administration 15

Local and remote administration 15

Performing administration tasks using commands 15

Control commands 16

WebSphere MQ Script (MQSC) commands 16

PCF commands 16

Attributes in WebSphere MQ Script (MQSC) and PCF commands 17

Understanding WebSphere MQ file names 17

Queue manager name transformation. 17

Object name transformation 18

Understanding case sensitivity 18

Case sensitivity in control commands. 18

Case sensitivity in MQSC commands. 19

Chapter 3. Managing queue managers 21

Using control commands 21

Using WebSphere MQ control commands on HP

OpenVMS systems 21

Creating a queue manager 22

Guidelines for creating queue managers 22

Creating a default queue manager 25

Making an existing queue manager the default 25

Backing up configuration files after creating a

queue manager 26

Starting a queue manager 26

Starting a queue manager automatically 26

Quiesced shutdown 26

Immediate shutdown 27

Preemptive shutdown 27

If you have problems shutting down a queue

manager 27

Deleting a queue manager 28

Chapter 4. Administering local

WebSphere MQ objects 29

Supporting application programs that use the MQI 29

Performing local administration tasks using MQSC

commands. 30

WebSphere MQ object names 30

Redirecting input and output 31

Running MQSC commands interactively. 31

Running MQSC commands from text files 32

Resolving problems with MQSC 35

Working with queue managers 36

Displaying queue manager attributes. 37

Altering queue manager attributes. 38

Working with local queues 38

Defining a local queue. 38

Displaying default object attributes 39

Copying a local queue definition 40

Changing local queue attributes 40

Clearing a local queue. 41

Deleting a local queue. 41

Browsing queues 41

Working with alias queues 43

Defining an alias queue 43

Using other commands with alias queues 44

Working with model queues. 44

Defining a model queue 45

Using other commands with model queues. 45

Working with services 45

Defining a service object 46

Managing services 47

Additional environment variables 47

Replaceable inserts on service definitions 48

Examples on using service objects 49

Managing objects for triggering. 51

Defining an application queue for triggering . . .	51
Defining an initiation queue	52
Defining a process	53
Displaying attributes of a process definition . . .	53

Chapter 5. Automating administration tasks 55

PCF commands	55
PCF object attributes	56
Escape PCFs	56
Using the MQAI to simplify the use of PCFs . . .	56
Command servers	57
Administering remote WebSphere MQ objects . . .	57
Channels, clusters, and remote queuing	57
Remote administration from a local queue manager	59
Creating a local definition of a remote queue . .	65
Using remote queue definitions as aliases . . .	68
Data conversion	69

Chapter 6. Configuring WebSphere MQ 71

WebSphere MQ configuration files	71
Editing configuration files	71
The WebSphere MQ configuration file, mqsc.ini . .	72
Queue manager configuration files, qm.ini . . .	73
Attributes for changing WebSphere MQ configuration information	73
AllQueueManagers stanza	73
ClientExitPath stanza	75
DefaultQueueManager stanza	75
ExitProperties stanza	75
The LogDefaults stanza	76
API exits	78
QueueManagers stanza	78
Changing queue manager configuration information .	79
The Service stanza	79
The Log stanza	80
The XAResourceManager stanza	82
The Channels stanza	83
The LU62 and TCP stanzas	85
The ExitPath stanza	86
Example mqsc.ini and qm.ini files	87

Chapter 7. WebSphere MQ security . . . 91

Why you need to protect WebSphere MQ resources	91
Before you begin	91
User IDs in WebSphere MQ for HP OpenVMS with resource identifier MQM	91
For more information	92
Understanding the Object Authority Manager . . .	92
How the OAM works	93
Managing access through rights identifiers . . .	93
Default rights identifier	94
Resources you can protect with the OAM	94
Using rights identifiers for authorizations . . .	94
Disabling the object authority manager	95
Using the Object Authority Manager commands . .	95
What you specify when you use the OAM commands	95
Using the setmqaut command	96

Access authorizations	97
Display authority command	97
Object Authority Manager guidelines	97
User IDs	98
Queue manager directories	98
Queues	98
Alternate user authority	98
Context authority	99
Remote security considerations	99
Channel command security	100
Understanding the authorization specification tables	101
MQI authorizations	102
Administration authorizations	105
Authorizations for MQSC commands in escape PCFs	105
Working with the Secure Sockets Layer (SSL) on OpenVMS systems	107
OpenSSL setup for WebSphere MQ	107

Chapter 8. Transactional support . . . 111

Database coordination	112
Restrictions	112
Database connections	113
Configuring database managers	113
Oracle configuration	115
Checking the environment variable settings . .	115
Enabling Oracle XA support	115
Creating the Oracle switch load file	115
Creating the Oracle switch load file on OpenVMS systems	116
Adding XAResourceManager configuration information for Oracle	116
Changing Oracle configuration parameters . . .	118
Administration tasks	118
In-doubt units of work	119
Displaying outstanding units of work with the dspmqtrn command	119
Resolving outstanding units of work with the rsvmqtrn command	120
Mixed outcomes and errors	121
Changing configuration information	122

Chapter 9. The WebSphere MQ dead-letter queue handler 125

Invoking the DLQ handler	125
The sample DLQ handler, amqsdld	126
The DLQ handler rules table	126
Control data	126
Rules (patterns and actions)	127
Rules table conventions	130
How the rules table is processed	132
Ensuring that all DLQ messages are processed	133
An example DLQ handler rules table	133

Chapter 10. WebSphere MQ for OpenVMS and clustering 137

Installing WebSphere MQ in an OpenVMS cluster	137
OpenVMS cluster failover sets	138
Overview of OpenVMS cluster failover sets . .	138

OpenVMS cluster failover set concepts	139	Are there any return codes explaining the problem?	180
Preparing to configure an OpenVMS cluster failover set	140	Can you reproduce the problem?	180
Configuring an OpenVMS cluster failover set	140	Have any changes been made since the last successful run?	180
OpenVMS cluster failover set post-configuration tasks	141	Has the application run successfully before?	180
Editing the FAILOVER.INI configuration file	141	Problems with commands	182
Command procedures used by failover sets	142	Does the problem affect specific parts of the network?	182
Administration of failover sets	143	Does the problem occur at specific times of the day?	182
Startup of failover monitors	144	Is the problem intermittent?	182
Starting a queue manager within a failover set	144	Have you applied any service updates?	182
Ending a queue manager within a failover set	144	Looking at problems in more detail	183
Moving a queue manager within a failover set	145	Have you obtained incorrect output?	183
Displaying the state of a failover set.	145	Have you failed to receive a response from a PCF command?	186
Setting DCL symbols to the state of a failover set	146	Are some of your queues failing?	187
Halting a failover monitor process	147	Does the problem affect only remote queues?	187
Executing commands while an update is in progress	147	Is your application or system running slowly?	187
Changing the state of a failover set	148	Application design considerations	188
Setting up security for ICC associations	148	Effect of message length.	188
Troubleshooting problems with failover sets	149	Effect of message persistence	188
Using MultiNet for OpenVMS with failover sets	150	Searching for a particular message	188
An example of using failover sets	150	Queues that contain messages of different lengths	189
		Frequency of syncpoints.	189
		Use of the MQPUT1 call.	189
Chapter 11. Recovery and restart	157	Error logs	189
Making sure that messages are not lost (logging)	157	Error log files	189
What logs look like	158	Operator messages	193
Types of logging	158	Dead-letter queues	194
Circular logging	158	Configuration files and problem determination	194
Linear logging	159	Using WebSphere MQ trace	194
Checkpointing – ensuring complete recovery	160	Trace file names	194
Calculating the size of the log	162	Sample trace data	195
Managing logs	164	First-failure support technology (FFST)	195
What happens when a disk gets full.	164	How to examine the FFSTs	195
Managing log files.	165	Problem determination with WebSphere MQ clients	198
Using the log for recovery	166	Terminating clients	198
Recovering from power loss or communications failures	166	Error messages with clients.	198
Recovering damaged objects	167	OpenVMS clients	198
Protecting WebSphere MQ log files	169		
Backing up and restoring WebSphere MQ	169	Chapter 13. How to use WebSphere MQ control commands	199
Backing up queue manager data	170	Names of WebSphere MQ objects.	199
Restoring queue manager data	170	How to read syntax diagrams	200
Using a backup queue manager	171	Example syntax diagram	201
Creating a backup queue manager	171	Syntax help	201
Updating a backup queue manager	172	Examples.	202
Starting a backup queue manager	172		
Recovery scenarios	173	Chapter 14. The control commands	203
Disk drive failures.	173	crtmqcvx (data conversion).	204
Damaged queue manager object	174	crtmqm (create queue manager)	205
Damaged single object	175	dltmqm (delete queue manager)	209
Automatic media recovery failure	175	dmpmqaut (dump authority)	210
Dumping the contents of the log using the dmpmqlog command.	175	dmpmqlog (dump log)	214
		dspmqr (display queue managers).	215
Chapter 12. Problem determination	179	dspmqa (display authority)	216
Preliminary checks	179	dspmqs (display command server)	220
Has WebSphere MQ run successfully before?	179		
Are there any error messages?	180		

dspmqls (display files)	221
dspmqrte (WebSphere MQ display route application)	222
dspmqrtrc (Display WebSphere MQ formatted trace output)	230
dspmqrtrn (display transactions)	231
dspmqrver (display version information)	232
endmqcsv (end command server).	233
endmqslr (end listener)	234
endmqm (end queue manager)	235
endmqtrc (End WebSphere MQ trace)	237
mqftapp (run File Transfer Application GUI)	238
mqftrcv (receive file on server)	239
mqftrcvc (receive file on client)	241
mqftsnd (send file from server)	244
mqftsncd (send file from client)	246
rcdmqimg (record media image)	248
rcrmqobj (recreate object)	250
rsvmqtrn (resolve transactions)	252
runmqchi (run channel initiator)	253
runmqchl (run channel)	254
runmqdlq (run dead-letter queue handler).	255
runmqslr (run listener)	256
runmqsc (run MQSC commands).	258
runmqtmc (start client trigger monitor).	260
runmqtrm (start trigger monitor)	261
setmqaut (grant or revoke authority)	261
Authorizations for MQI calls	267
Authorizations for context	267
Authorizations for commands	267
Authorizations for generic operations	268
setmqprd (enroll production license).	268
strmqcsv (start command server).	268
strmqm (start queue manager).	269

Appendix A. System and default objects 273

Appendix B. Directory structure . . . 275
 Directories and files in MQS_ROOT:[MQM] . . . 277
 Directories and files in the MQS_ROOT:[MQM.QMGRS.QMNAME] subdirectory. 277

Appendix C. Comparing command sets 281

Commands for queue manager administration	281
Commands for command server administration	281
Commands for queue administration	282
Commands for process administration	282
Commands for channel administration	283
Other control commands	284

Appendix D. Stopping and removing queue managers manually. 285

Stopping a queue manager manually	285
Stopping queue managers in WebSphere MQ for OpenVMS systems	285
Removing queue managers manually	285

Removing queue managers in WebSphere MQ for Windows	286
Removing queue managers in WebSphere MQ for UNIX systems	287

Appendix E. Sample MQI programs and MQSC files 289

MQSC command file samples	289
C and COBOL program samples	289
Miscellaneous tools	290
Command file for application triggering	290
Examples.	290

Appendix F. OpenVMS cluster failover set templates 293

Template Configuration File	
FAILOVER.TEMPLATE	293
Template StartCommand procedure	
START_QM.TEMPLATE	294
Template EndCommand procedure	
END_QM.TEMPLATE	295
Template TidyCommand procedure	
TIDY_QM.TEMPLATE	297

Appendix G. MONMQ diagnostic utility 299

Overview.	299
Variables within MONMQ	300
Assigning default values	302
Opening or creating a trace section and associated mailbox	302
Displaying the logical unit definition	303
Closing and deleting an LU	303
Display channel details	303
Display the current trace mask for a channel	304
Display the contents of the target threads stack	304
Display active WebSphere MQ related processes and memory usage	305
Displays all messages held in a channel	305
Display all WebSphere MQ related global sections on the current node	306
Signals target thread to send mutex table to client trace process	307
Signals target thread to send internal events table to client trace process.	308
Signals target thread to send internal mapped shared memory table to the client trace process	309
Displays active WebSphere MQ components by name and hexadecimal ids	310
Display functions within specified component	311
Activate tracing from the point a process starts	312
Prevent WebSphere MQ process from tracing immediately from startup	312
Connect target thread to specified channel.	312
Disconnect target thread to specified channel.	313
Display real-time trace message written to the LUs trace mailbox	313
Detach and end current client process	313
Specify trace data	313
Remove single entry from the trace filter table	314
Client process writes trace messages to a binary file	314

Close binary trace messages file	315
Client process writes trace messages to a text file	315
Close text trace messages file	315
Timestamp messages	315
Stop timestamping messages	315
Enable tracing	315
Disable tracing	316
Save message history	316
Disable message history	316
Delete message history	316
Set history depth	316
Reset stack and history data for a channel.	317
Enable or disable mask bit	317
Set a color for a channel.	318
Redirect output to file	319
Analyze trace binary file	319
Display current state of WebSphere MQ threads	321
Close trace and exit MONMQ.	321
Quit MONMQ without closing trace.	321
Managing shared memory with MONMQ.	322
Scripts and macros in MONMQ	323
Sample trace session	324

Appendix H. User exits	335
Channel and Workload Exits	335
WebSphere MQ Cluster Workload Exits	335

Appendix I. Trusted applications	337
User applications	337
Setting up trusted applications	337
Running channels and listeners as trusted applications	338
Fast, nonpersistent messages	338

Appendix J. SSL CipherSpecs	339
--	------------

Notices	341
--------------------------	------------

Trademarks	343
-----------------------------	------------

Index	345
------------------------	------------

Sending your comments to IBM	357
---	------------

Figures

1. Queues, messages, and applications	29	14. Commented out XAResourceManager stanza	123
2. Extract from the MQSC command file, myprog.in	33	15. Sample entry required for ICC\$SYSTARTUP.COM	149
3. Extract from the MQSC report file, myprog.out.	34	16. Failover.template for creating a FAILOVER.INI configuration file	151
4. Typical output from a DISPLAY QMGR command	37	17. start_failover_set command procedure	153
5. Typical results from a queue browser	42	18. end_failover_set command procedure	155
6. Remote administration using MQSC commands	60	19. Checkpointing	161
7. Setting up channels and queues for remote administration	61	20. Checkpointing with a long-running transaction	162
8. Example of a WebSphere MQ configuration file for WebSphere MQ for HP OpenVMS systems .	88	21. Sample WebSphere MQ for HP OpenVMS trace	195
9. Example queue manager configuration file for WebSphere MQ for HP OpenVMS	89	22. Default directory structure after a queue manager has been started	276
10. Source code for Oracle switch load file, oraswit.c	115	23. Template configuration file: failover.template	294
11. Sample XAResourceManager entry for Oracle	118	24. Template StartCommand procedure: Start_QM.template	295
12. Sample dspmqtrn output	120	25. Template EndCommand procedure: END_QM.template.	297
13. Sample dspmqtrn output for a transaction in error	122	26. Template TidyCommand procedure: TIDY_QM.template	298

Tables

1. Categories of control commands	21	16. Specifying authorities for different object types	265
2. List of possible ISO CCSIDs	74	17. System and default objects for queues	273
3. Default outstanding connection requests (TCP)	85	18. System and default objects for channels	274
4. Security authorization needed for MQI calls	102	19. System and default objects for namelists	274
5. MQSC commands and security authorization needed.	105	20. System and default objects for processes	274
6. PCF commands and security authorization needed.	106	21. Commands for queue manager administration	281
7. Description of the fields within the FAILOVER.INI file.	141	22. Commands for command server administration	281
8. Parameters passed to command procedures	142	23. Commands for queue administration	282
9. Failover set queue manager states.	145	24. Commands for process administration	282
10. Failover set node queue manager states	146	25. Commands for channel administration	283
11. Failover set node monitor states	146	26. Other control commands.	284
12. DCL symbols and description	147	27. MQSC command files.	289
13. Log overhead sizes (all values are approximate).	163	28. Sample programs - source files.	289
14. How to read syntax diagrams	200	29. Miscellaneous files.	290
15. Specifying authorities for different object types	218		

Chapter 1. Introduction to WebSphere MQ

This chapter introduces WebSphere® MQ Version 6 from an administrator's perspective, and describes the basic concepts of WebSphere MQ and messaging. It contains these sections:

- "WebSphere MQ and message queuing"
- "Messages and queues" on page 2
- "Objects" on page 3
- "Clients and servers" on page 11
- "Extending queue manager facilities" on page 11
- "Security" on page 13
- "Transactional support" on page 13

WebSphere MQ and message queuing

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

WebSphere MQ implements a common application programming interface known as the *message queue interface* (or MQI) wherever the applications run. This makes it easier for you to port application programs from one platform to another.

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

Note: Message Queue Interface calls cannot be made from within an AST routine on WebSphere MQ on OpenVMS because WebSphere MQ uses AST routines itself, and these routines cannot run while another AST routine is active.

Time-independent applications

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

Message-driven processing

When messages arrive on a queue, they can automatically start an application using *triggering*. If necessary, the applications can be stopped when the message (or messages) have been processed.

Messages and queues

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

What is a message?

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

WebSphere MQ messages have two parts:

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

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

Message lengths

The default maximum message length is 4 MB, although you can increase this to a maximum length of 100 MB (where 1 MB equals 1 048 576 bytes). In practice, the message length might 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 the queue
- The maximum message length defined by either the sending or receiving application
- The amount of storage available for the message

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

How do applications send and receive messages?

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

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

1. Opens the required queue by issuing an MQI **MQOPEN** call
2. Issues an MQI **MQPUT** call to put the message onto the queue

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

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

What is a queue?

A *queue* is a data structure used to store messages.

Each queue is owned by a *queue manager*. The queue manager is responsible for maintaining the queues it owns, and for storing all the messages it receives onto the appropriate queues. The messages might be put on the queue by application programs, or by a queue manager as part of its normal operation.

WebSphere MQ Version 6.0 supports queues over 2 GB in size. For information about planning the amount of storage you need for queues, see *WebSphere MQ for HP OpenVMS Quick Beginnings*, or visit the WebSphere MQ Web site:

<http://www-306.ibm.com/software/integration/wmqproductline/>

Predefined queues and dynamic queues

Queues can be characterized by the way they are created:

- **Predefined queues** are created by an administrator using the appropriate MQSC or PCF commands. Predefined queues are permanent; they exist independently of the applications that use them and survive WebSphere MQ restarts.
- **Dynamic queues** are created when an application issues an **MQOPEN** request specifying the name of a *model queue*. The queue created is based on a *template queue definition*, which is called a 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 are lost on restart.

Retrieving messages from queues

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

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

The **MQGET** request from the application determines the method used.

Objects

Many of the tasks described in this book involve manipulating WebSphere MQ **objects**. The object types are queue managers, queues, process definitions, namelists, channels, client connection channels, listeners, services, and authentication information objects.

The manipulation or *administration* of objects includes:

- Starting and stopping queue managers.
- Creating objects, particularly queues, for applications.
- Working with channels to create communication paths to queue managers on other (remote) systems. This is described in detail in the *WebSphere MQ Intercommunications* manual.
- Creating *clusters* of queue managers to simplify the overall administration process, and to balance workload.

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

- Chapter 2, “An introduction to WebSphere MQ administration,” on page 15
- Chapter 3, “Managing queue managers,” on page 21
- Chapter 4, “Administering local WebSphere MQ objects,” on page 29
- Chapter 5, “Automating administration tasks,” on page 55

Object names

The naming convention adopted for WebSphere MQ objects depends on the object.

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

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

In WebSphere MQ, names can have a maximum of 48 characters, with the exception of *channels* which have a maximum of 20 characters. For more information about names, see “Names of WebSphere MQ objects” on page 199.

Managing objects

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

- Control commands, which are typed in from a keyboard
- MQSC commands, which can be typed in from a keyboard or read from a file
- Programmable Command Format (PCF) messages, which can be used in an automation program
- WebSphere MQ Administration Interface (MQAI) calls in a program

For more information about these methods, see Chapter 2, “An introduction to WebSphere MQ administration,” on page 15.

Object attributes

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

In WebSphere MQ, there are two ways of referring to an attribute:

- Using its PCF name, for example, *MaxMsgLength*.
- Using its MQSC command name, for example, MAXMSGL.

This book mainly describes how to specify attributes using MQSC commands, and so it refers to most attributes using their MQSC command names, rather than their PCF names.

WebSphere MQ queues

There are four types of queue object available in WebSphere MQ.

Local queue object

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

Remote queue object

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

Before applications can send messages to a queue on another queue manager, you must have defined a transmission queue and channels between the queue managers, **unless** you have grouped one or more queue managers together into a *cluster*. For more information about clusters, see “Remote administration using clusters” on page 58.

Alias queue object

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

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

Model queue object

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

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

Defining queues

Queues are defined to WebSphere MQ using:

- The MQSC command DEFINE
- The PCF Create Queue command

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

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

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

For further details about defining queue objects, see the *WebSphere MQ Script (MQSC) Command Reference* or *WebSphere MQ Programmable Command Formats and Administration Interface* manuals.

Queues used by WebSphere MQ

WebSphere MQ uses some local queues for specific purposes related to its operation. You **must** define these queues before WebSphere MQ 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 connected, 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 get messages only from a local queue.

Initiation queues

Initiation queues are queues that are used in triggering. A queue manager puts a trigger message on an initiation queue when a trigger event occurs. A trigger event is a logical combination of conditions that is detected by a queue manager. For example, a trigger event might be generated when the number of messages on a queue reaches a predefined depth. This event causes the queue manager to put a trigger message on a specified initiation queue. This trigger message is retrieved by a *trigger monitor*, a special application that monitors an initiation queue. The trigger monitor then starts 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 51 and “runmqtrm (start trigger monitor)” on page 261. For more information about triggering, see the *WebSphere MQ Application Programming Guide*.

Transmission queues

Transmission queues are queues that temporarily store messages that are destined for a remote queue manager. You must define at least one transmission queue for each remote queue manager to which the local queue manager is to send messages directly. These queues are also used in remote administration; see “Remote administration from a local queue manager” on page 59. For information about the use of transmission queues in distributed queuing, see the *WebSphere MQ Intercommunications* manual.

Each queue manager can have a default transmission queue. When a queue manager that is not part of a cluster puts a message onto a remote queue, the default action, if there is no transmission queue with the same name as the destination queue manager, is to use the default transmission queue.

Cluster transmission queues

Each queue manager within a cluster has a cluster transmission queue called `SYSTEM.CLUSTER.TRANSMIT.QUEUE`. A definition of this queue is created by default when you define a queue manager.

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

During name resolution, the cluster transmission queue takes precedence over the default transmission queue.

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

Dead-letter queues

A dead-letter (undelivered-message) queue is a queue that stores messages that cannot be routed to their correct destinations. This occurs when, for example, the destination queue is full. The supplied dead-letter queue is called `SYSTEM.DEAD.LETTER.QUEUE`.

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

Command queues

The command queue, `SYSTEM.ADMIN.COMMAND.QUEUE`, is a local queue to which suitably authorized applications can send MQSC commands for processing. These commands are then retrieved by a WebSphere MQ component called the command server. The command server validates the commands, passes the valid ones on for processing by the queue manager, and returns any responses to the appropriate reply-to queue.

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

Reply-to queues

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

Event queues

WebSphere MQ 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 might inform an administrator or initiate some remedial action if the event indicates a problem.

Note: Trigger events are 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 *WebSphere MQ Programmable Command Formats and Administration Interface* manual.

WebSphere MQ queue managers

A *queue manager* provides queuing services to applications, and manages the queues that belong to it. It ensures that:

- Object attributes are changed according to the commands received.
- Special events such as trigger events or instrumentation events are generated when the appropriate conditions are met.
- Messages are put on the correct queue, as requested by the application making the **MQPUT** call. The application is informed if this cannot be done, and an appropriate reason code is given.

Each queue belongs to a single queue manager and is said to be a *local queue* to that queue manager. The queue manager to which an application is connected is said to be the *local queue manager* for that application. For the application, the queues that belong to its local queue manager are local queues.

A *remote queue* is a queue that belongs to another queue manager. A *remote queue manager* is any queue manager other than the local queue manager. A remote queue manager can exist on a remote machine across the network, or might exist on the same machine as the local queue manager. WebSphere MQ supports multiple queue managers on the same machine.

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

Process definitions

A *process definition* object defines an application that starts in response to a trigger event on a WebSphere MQ queue manager. See the Initiation queues entry under “Queues used by WebSphere MQ” on page 6 for more information.

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

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

Clusters

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

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

For information about clusters, see “Administering remote WebSphere MQ objects” on page 57, and the *WebSphere MQ Queue Manager Clusters* manual.

Note: WebSphere MQ clusters are not the same as OpenVMS clusters. When the term cluster is used, it refers to a WebSphere MQ queue manager cluster. An OpenVMS cluster is always referred to as an OpenVMS cluster. For more information on OpenVMS clusters, see Chapter 10, “WebSphere MQ for OpenVMS and clustering,” on page 137

Namelists

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

Namelists are also used with queue manager clusters to maintain a list of clusters referred to by more than one WebSphere MQ object.

Authentication information objects

The queue manager authentication information object forms part of WebSphere MQ support for Secure Sockets Layer (SSL) security. It provides the definitions needed to check certificate revocation lists (CRLs) using LDAP servers. CRLs allow Certification Authorities to revoke certificates that can no longer be trusted.

This book describes using the **setmqaut**, **dspmqaut**, **dmpmqaut**, **rcrmqobj**, **rcdmqimg**, and **dspmqfls** commands with the authentication information object. For an overview of SSL and the use of the authentication information objects, see Chapter 7, “WebSphere MQ security,” on page 91.

Channels

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

Client connection channels

Client connection channels are objects that provide a communication path from a WebSphere MQ client to a queue manager. Client connection channels are used in distributed queuing to move messages between a queue manager and a client.

They shield applications from the underlying communications protocols. The client might exist on the same, or different, platform to the queue manager.

For information on client connection channels and how to use them, see the *WebSphere MQ Intercommunications* manual.

Listeners

Listeners are processes that accept network requests from other queue managers, or client applications, and start associated channels. Listener processes can be started using the **runmqtsr** control command.

Listener objects are WebSphere MQ objects that allow you to manage the starting and stopping of listener processes from within the scope of a queue manager. By defining attributes of a listener object you do the following:

- Configure the listener process.
- Specify whether the listener process automatically starts and stops when the queue manager starts and stops.

Services

Service objects are a way of defining programs to be executed when a queue manager starts or stops. The programs can be split into the following types:

Servers

A *server* is a service object that has the parameter **SERVTYPE** specified as **SERVER**. A server service object is the definition of a program that is executed when a specified queue manager is started. Only one instance of a server process can be executed concurrently. While running, the status of a server process can be monitored using the MQSC command, **DISPLAY SVSTATUS**. Typically server service objects are definitions of programs such as dead letter handlers or trigger monitors, however the programs that can be run are not limited to those supplied with WebSphere MQ. Additionally, a server service object can be defined to include a command that runs when the specified queue manager is shutdown to end the program.

Commands

A *command* is a service object that has the parameter **SERVTYPE** specified as **COMMAND**. A command service object is the definition of a program that is executed when a specified queue manager is started or stopped. Multiple instances of a command process can be executed concurrently. Command service objects differ from server service objects in that once the program is executed the queue manager does not monitor the program. Typically command service objects are definitions of programs that are short lived and perform a specific task such as starting one, or more, other tasks.

System default objects

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

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

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

For more information see Appendix A, “System and default objects,” on page 273

Clients and servers

WebSphere MQ supports client-server configurations for its applications.

A WebSphere MQ *client* is a component that allows an application running on a system to issue MQI calls to a queue manager running on another system. The output from the call is sent back to the client, which passes it back to the application.

A WebSphere MQ *server* is a queue manager that provides queuing services to one or more clients. All the WebSphere MQ objects, for example queues, exist only on the queue manager machine (the WebSphere MQ server machine), and not on the client. A WebSphere MQ server can also support local WebSphere MQ applications.

The difference between a WebSphere MQ 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 *WebSphere MQ Intercommunications* manual.

For information about client support in general, see the *WebSphere MQ Clients* manual.

WebSphere MQ applications in a client-server environment

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

You must link your applications to the appropriate client libraries. See the *WebSphere MQ Clients* manual for further information.

Extending queue manager facilities

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

- User exits
- API exits
- Installable services

See “Installable services” on page 12 for more information about the installable services.

User exits

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

Channel exits

These exits change the way that channels operate. Channel exits are described in the *WebSphere MQ Intercommunications* manual.

Data conversion exits

These exits create source code fragments that can be put into application programs to convert data from one format to another. Data conversion exits are described in the *WebSphere MQ Application Programming Guide*.

The cluster workload exit

The function performed by this exit is defined by the provider of the exit. Call definition information is given in the *WebSphere MQ Queue Manager Clusters* manual.

API exits

API exits let you write code that changes the behavior of WebSphere MQ API calls, such as MQPUT and MQGET, and then insert that code immediately before or immediately after those calls. The insertion is automatic; the queue manager drives the exit code at the registered points. For more information about API exits, see “API exits” on page 78 and the *WebSphere MQ Application Programming Guide*.

Installable services

Installable services 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 WebSphere MQ, or you can write your own component to perform the functions that you require.

Currently, the following installable services are provided:

Authorization service

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

The default service component that implements the service is the Object Authority Manager (OAM). By default, the OAM is active, and you do not have to do anything to configure it. You can use the authorization service interface to create other components to replace or augment the OAM. For more information about the OAM, see Chapter 7, “WebSphere MQ security,” on page 91.

Name service

The name service enables applications to share queues by identifying remote queues as though they were local queues.

You can write your own name service component. You might want to do this if you intend to use the name service with WebSphere MQ Version 6.0, for example. To use the name service you must have either a user-written, or a third party, component. By default, the name service is inactive.

Security

In WebSphere MQ, there are three methods of providing security:

- The Object Authority Manager (OAM) facility
- User-written, or third party, channel exits
- Channel security using Secure Sockets Layer (SSL)

Object Authority Manager (OAM) facility

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 WebSphere MQ entities is controlled through WebSphere MQ user groups and the OAM. We provide a command line interface to enable administrators to grant or revoke authorizations as required.

For more information about creating authorization service components, see Chapter 7, “WebSphere MQ security,” on page 91.

User-written or third party channel exits

Channels can use user-written or third party channel exits. For more information, see the *WebSphere MQ Intercommunications* manual.

Channel security using SSL

The Secure Sockets Layer (SSL) protocol provides industry-standard channel security, with protection against eavesdropping, tampering, and impersonation.

SSL uses public key and symmetric techniques to provide message privacy and integrity and mutual authentication.

For a comprehensive review of security in WebSphere MQ including detailed information on SSL, see the *WebSphere MQ Security* manual. For an overview of SSL, including pointers to the commands described in this book, see “Working with Queue Manager and Client Certificates” on page 108.

Transactional support

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

When a unit of work completes successfully, it is said to *commit*. Once committed, all updates made within that unit of work are made permanent and irreversible. However, if the unit of work fails, all updates are instead *backed out*. This process, where units of work are either committed or backed out with integrity, is known as *syncpoint coordination*.

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

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

For more information, see Chapter 8, “Transactional support,” on page 111.

Chapter 2. An introduction to WebSphere MQ administration

This chapter introduces WebSphere MQ administration.

Administration tasks include creating, starting, altering, viewing, stopping, and deleting clusters, processes and WebSphere MQ objects (queue managers, queues, namelists, process definitions, channels, client connection channels, listeners, services, and authentication information objects).

The chapter contains the following sections:

- “Local and remote administration”
- “Performing administration tasks using commands”
- “Understanding WebSphere MQ file names” on page 17
- “Understanding case sensitivity” on page 18

Local and remote administration

You administer WebSphere MQ objects locally or remotely.

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

WebSphere MQ supports administration from a single point of contact through what is known as *remote administration*. This allows you to issue commands from your local system that are processed on another system. For example, you can issue a remote command to change a queue definition on a remote queue manager. 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.

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.

“Administering remote WebSphere MQ objects” on page 57 describes the subject of remote administration in greater detail.

Performing administration tasks using commands

There are three sets of commands that you can use to administer WebSphere MQ:

- Control commands
- MQSC commands
- PCF commands

Control commands

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

They are described in Chapter 3, “Managing queue managers,” on page 21.

WebSphere MQ Script (MQSC) commands

Use MQSC commands to manage queue manager objects, including the queue manager itself, queues, process definitions, namelists, channels, client connection channels, listeners, services, and authentication information objects.

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

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

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

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

MQSC commands are available on other platforms, including i5/OS[®], and z/OS. MQSC commands are summarized in Appendix C, “Comparing command sets,” on page 281.

See the *WebSphere MQ Script (MQSC) Command Reference* manual for a description of each MQSC command and its syntax.

See “Performing local administration tasks using MQSC commands” on page 30 for more information about using MQSC commands in local administration.

PCF commands

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

PCF commands cover the same range of functions provided by the MQSC commands. See “PCF commands” on page 55 for more information.

You can use the WebSphere MQ Administration Interface (MQAI) to obtain easier programming access to PCF messages. This is described in greater detail in “Using the MQAI to simplify the use of PCFs” on page 56.

Attributes in WebSphere MQ Script (MQSC) and PCF commands

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 PCF commands

Escape PCF commands 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 *WebSphere MQ Programmable Command Formats and Administration Interface* manual.

Understanding WebSphere MQ file names

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

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

- A prefix - 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 WebSphere MQ, 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 on the maximum length a file name can be, and on 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.

Using the example of a queue manager with the name QUEUE.MANAGER individual characters are transformed as follows:

- . becomes \$

- / becomes _
- % becomes _

If the queue manager name is still not valid it is truncated to eight characters and a three-character numeric suffix is appended.

For example, assuming the default prefix, the queue manager name becomes:
 MQS_ROOT: [MQM.QMGRS.QUEUE\$MANAGER] with the name queue.manager

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

Object name transformation

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

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

Queue file names begin with the letter Q.

For more information on naming objects, see "Names of WebSphere MQ objects" on page 199.

Understanding case sensitivity

Case sensitivity in control commands

OpenVMS is normally described as a case-insensitive operating system. This means that, in general, the following three commands all create a queue manager called "QUEUEMANAGER".

```
$ crtmqm QueueManager
$ crtmqm queuemanager
$ crtmqm QUEUEMANAGER
```

With WebSphere MQ for HP OpenVMS, you can use double quotation marks around the name of the queue manager (or similar parameter) to protect its case. When the double quotation marks are used, the three commands now create three different queue managers.

```
$ crtmqm "QueueManager" creates a queue manager called QueueManager
$ crtmqm "queuemanager" creates a queue manager called queuemanager
$ crtmqm "QUEUEMANAGER" creates a queue manager called QUEUEMANAGER
```

The following command on WebSphere MQ for HP OpenVMS handles upper and lower case characters:

```
set process /parse_style = ( traditional | extended )
```


If **set process /parse_style** command is not used, or if it is used with the **traditional** option, then OpenVMS behaves as it always has done with regard to case sensitivity.

If this command is used with the **extended** option, the behavior of the LIB\$GET_FOREIGN runtime library routine changes so that it preserves the case of text that it retrieves. Because WebSphere MQ uses this routine to obtain command line parameters, the case of the parameters is preserved, even when the parameters are not enclosed in double quotation marks.

For example, the following sequence of commands creates three different queue managers. Note that the parameters are not enclosed in double quotation marks.

```
$ set process /parse_style = extended
$ crtmqm QueueManager creates a queue manager called QueueManager
$ crtmqm queuemanager creates a queue manager called queuemanager
$ crtmqm QUEUEMANAGER creates a queue manager called QUEUEMANAGER
```

The OpenVMS **set process /parse_style** command changes a number of things besides case sensitivity. You can learn more about the command from the information provided in the OpenVMS DCL Dictionary before applying it to your system.

Case sensitivity in MQSC commands

WebSphere MQ control commands (for example, **runmqsc** which invokes the MQSC facility) are not case sensitive.

MQSC commands, including their attributes, can be written in upper or lower case. Object names in MQSC commands are automatically converted to upper case unless the names are enclosed in *single* quotation marks. If single quotation marks are not used, the object is processed with a name in upper case. See the *WebSphere MQ Script (MQSC) Command Reference* manual for more information.

Chapter 3. Managing queue managers

This chapter tells you how to perform operations on queue managers using control commands and the WebSphere MQ Explorer.

It contains the following topics:

- “Using control commands”
- “Creating a queue manager” on page 22
- “Starting a queue manager” on page 26
- “Stopping a queue manager manually” on page 285
- “Deleting a queue manager” on page 28

For more information, see Chapter 13, “How to use WebSphere MQ control commands,” on page 199

Using control commands

You use control commands to perform operations on queue managers, command servers, and channels. Control commands can be divided into three categories, as shown in Table 1.

Table 1. Categories of control commands

Category	Description
Queue manager commands	Queue manager control commands include commands for creating, starting, stopping, and deleting queue managers and command servers.
Channel commands	Channel commands include commands for starting and ending channels and channel initiators.
Utility commands	Utility commands include commands associated with: <ul style="list-style-type: none">• 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 WebSphere MQ objects

For information about administration tasks for channels, see the *WebSphere MQ Intercommunications* manual.

Using WebSphere MQ control commands on HP OpenVMS systems

In WebSphere MQ for HP OpenVMS, you enter control commands at a DCL prompt. The command name and flags themselves are not case sensitive, but the

parameters may or may not be converted to upper case, depending on an OpenVMS process option and whether the parameters were enclosed in double quotation marks to protect the case. For more on how the OpenVMS command and double quotation marks affect case, see “Understanding case sensitivity” on page 18.

Typically, in this example:

```
crtmqm -u system.dead.letter.queue "jupiter.queue.manager"
```

- The dead-letter queue could be SYSTEM.DEAD.LETTER.QUEUE, even though it was entered in lower case. Whether its case is automatically converted to upper case depends on the setting of the OpenVMS command **set process/parse_style**. See “Understanding case sensitivity” on page 18.
- The queue manager name is specified as “jupiter.queue.manager” (which is different from “JUPITER.queue.manager”) because it was enclosed in double quotation marks.

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

Creating a queue manager

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

Before you can do anything with messages and queues, you must create and start at least one queue manager and its associated objects. To create a queue manager, use the WebSphere MQ control command **crtmqm** (described in “crtmqm (create queue manager)” on page 205). The **crtmqm** command automatically creates the required default objects and system objects (described in “System default objects” on page 10). Default objects form the basis of any object definitions that you make; system objects are required for queue manager operation. When you have created a queue manager and its objects, use the **strmqm** command to start the queue manager.

Note: WebSphere MQ does not support machine names that contain spaces. If you install WebSphere MQ on a computer with a machine name that contains spaces, you cannot create any queue managers.

Guidelines for creating queue managers

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

Naming conventions

Use uppercase names so that you can communicate with queue managers on all platforms. Remember that names are assigned exactly as you enter them. To avoid the inconvenience of lots of typing, do not use unnecessarily long names.

Specify a unique queue manager name

When you create a queue manager, ensure that no other queue manager has the same name *anywhere* in your network. Queue manager names are

not checked when the queue manager is created, and names that are not unique prevent you from creating channels for distributed queuing.

One way of ensuring uniqueness is to prefix each queue manager name with its own unique node name. For example, if a node is called ACCOUNTS, you can 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 WebSphere MQ for communication with other enterprises, you can also include your own enterprise name as a prefix. This is not done in the examples, because it makes them more difficult to follow.

Note: Queue manager names in control commands may or may not be converted to upper case, depending on an OpenVMS process option and whether the queue manager name was enclosed in double quotation marks to protect the case. This means that you could create two queue managers with the names `jupiter.queue.manager` and `JUPITER.queue.manager`. For more on how the OpenVMS process option and double quotation marks affect case, see “Understanding case sensitivity” on page 18.

Limit the number of queue managers

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

In production systems, many processors can be exploited with a single queue manager, but larger server machines might run more effectively with multiple queue managers.

Specify a default queue manager

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

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

Changing the default queue manager can affect other users or applications. The change has no effect on currently-connected applications, because they can use the handle from their original connect call in any further `MQI` calls. This handle ensures that the calls are directed to the same queue manager. Any applications connecting *after* you have changed the default queue manager connect to the new default queue manager. This might be what you intend, but you should take this into account before you change the default.

Creating a default queue manager is described in “Creating a default queue manager” on page 25.

Specify a dead-letter queue

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

It is important to have a dead-letter queue on each queue manager in your network. If you do not define one, errors in application programs might cause channels to be closed, and replies to administration commands might not be received.

For example, if an application tries to put a message on a queue on another queue manager, but gives the wrong queue name, 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.

When you create a queue manager, 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 that you have already defined to specify the dead-letter queue to be used. See “Altering queue manager attributes” on page 38 for an example of the MQSC command ALTER.

Specify a default transmission queue

A transmission queue is a local queue on which messages in transit to a remote queue manager are queued before 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, 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 38 for more information.

Specify the logging parameters you require

You can specify logging parameters on the `crtmqm` command, including the type of logging, and the path and size of the log files.

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

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

Once you have set the logging parameters, some of them can only be changed by deleting the queue manager and recreating it with the same name but with different logging parameters.

For more information about logging parameters, see Chapter 11, “Recovery and restart,” on page 157.

Installing multiple queue managers

If you install multiple queue managers with different logical volumes for each queue manager, the `ftok` function might cause shared memory problems. The `ftok` function calculates the shared memory key from the node and the minor number of the housing logical volume. On AIX®, shared memory problems can occur if two queue managers are installed in an HACMP™ environment with different logical volumes that have identical minor device numbers.

These shared memory problems do not occur if the different logical volumes are created such that they have different minor device numbers.

Creating a default queue manager

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

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

```
crtmqm -q -d MY.DEFAULT.XMIT.QUEUE -u SYSTEM.DEAD.LETTER.QUEUE SATURN.QUEUE.MANAGER
```

where:

-q Indicates that this queue manager is the default queue manager.

-d MY.DEFAULT.XMIT.QUEUE

Is the name of the default transmission queue to be used by this queue manager.

Note: WebSphere MQ does not create a default transmission queue for you; you have to define it yourself.

-u SYSTEM.DEAD.LETTER.QUEUE

Is the name of the default dead-letter queue created by WebSphere MQ on installation.

SATURN.QUEUE.MANAGER

Is the name of this queue manager. This must be the last parameter specified on the **crtmqm** command.

The complete syntax of the **crtmqm** command is shown in “**crtmqm (create queue manager)**” on page 205.

The system and default objects are listed in Appendix A, “System and default objects,” on page 273.

Making an existing queue manager the default

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

- To make an existing queue manager the default, change the queue manager name on the `Name` attribute to the name of the new default queue manager. You can 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, create the *DefaultQueueManager* stanza with the required name yourself.
- If you accidentally make another queue manager the default and want to revert to the original default queue manager, edit the `DefaultQueueManager` stanza in `mqs.ini`, replacing the unwanted default queue manager with that of the one you want.

For more information about configuration files, see Chapter 6, “Configuring WebSphere MQ,” on page 71.

Backing up configuration files after creating a queue manager

There are two types of configuration file:

- When you install the product, the WebSphere MQ configuration file (mq5.ini) is created. It contains a list of queue managers that is updated each time you create or delete a queue manager. There is one mq5.ini file per node.
- When you create a new queue manager, a new queue manager configuration file (qm.ini) is automatically created. This contains configuration parameters for the queue manager.

After creating a queue manager, we recommend that you back up your configuration files.

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

For more information about configuration files, see Chapter 6, “Configuring WebSphere MQ,” on page 71.

Starting a queue manager

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

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

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

Starting a queue manager automatically

To start a queue manager automatically on WebSphere MQ for OpenVMS do the following:

1. Edit the file SYS\$STARTUP:MQS_STARTUP.COM
2. Locate the following line:
\$ mcr sys\$system:mqvmsinit
3. After the above line, add the following command:
\$ strmqm <queuemanagename>

where *queuemanagename* is the name of the queue manager you want to start automatically.

Quiesced shutdown

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

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


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

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

However, if you issue:

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

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

Immediate shutdown

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

For an immediate shutdown, type:

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

Preemptive shutdown

Attention!

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

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

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

This stops the queue manager immediately.

If this method still does not work, see “Stopping a queue manager manually” on page 285 for an alternative solution.

For a detailed description of the **endmqm** command and its options, see “endmqm (end queue manager)” on page 235.

Note: After a forced or preemptive shutdown, or if the queue manager fails, the queue manager may have ended without cleaning up the shared memory that it owns. This can lead to problems restarting. For information on how to use the MONMQ utility to clean up after an abrupt ending of this type, see “Managing shared memory with MONMQ” on page 322.

If you have problems shutting down a queue manager

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

- Do not check MQI return codes properly
- Do not request notification of a quiesce

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

If a problem occurs when you stop the queue manager, you can break out of the **endmqm** command using Ctrl-Y.

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

Deleting a queue manager

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

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

Attention:

- Deleting a queue manager is a drastic step, because you also delete all resources associated with the queue manager, including all queues and their messages and all object definitions. There is no displayed prompt that allows you to change your mind; when you press Enter all the associated resources are lost.
- In WebSphere MQ for Windows[®], the **dltmqm** command also removes a queue manager from the automatic startup list (described in “Starting a queue manager automatically” on page 26). When the command has completed, a WebSphere MQ queue manager ending message is displayed; you are not told that the queue manager has been deleted.

For a description of the **dltmqm** command and its options, see “dltmqm (delete queue manager)” on page 209. Ensure that only trusted administrators have the authority to use this command. For information about security, see Chapter 7, “WebSphere MQ security,” on page 91.

Chapter 4. Administering local WebSphere MQ objects

This chapter tells you how to administer local WebSphere MQ objects to support application programs that use the Message Queue Interface (MQI). In this context, local administration means creating, displaying, changing, copying, and deleting WebSphere MQ objects.

This chapter contains the following sections:

- “Supporting application programs that use the MQI”
- “Performing local administration tasks using MQSC commands” on page 30
- “Working with queue managers” on page 36
- “Working with local queues” on page 38
- “Working with alias queues” on page 43
- “Working with model queues” on page 44
- “Working with services” on page 45
- “Managing objects for triggering” on page 51

Supporting application programs that use the MQI

WebSphere MQ application programs need certain objects before they can run successfully. For example, Figure 1 shows an application that removes messages from a queue, processes them, and then sends some results to another queue on the same queue manager.

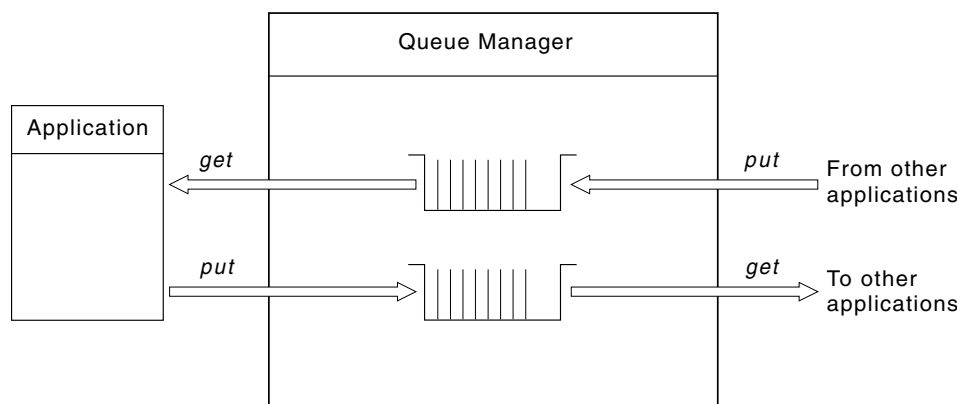


Figure 1. Queues, messages, and applications

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

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

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

- The application must be able to connect to the queue manager. To do this it must be linked to WebSphere MQ. See the *WebSphere MQ 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 1 on page 29.

Performing local administration tasks using MQSC commands

This section introduces you to MQSC commands and tells you how to use them for some common tasks.

You can use MQSC commands to manage queue manager objects, including the queue manager itself, queues, process definitions, channels, client connection channels, listeners, services, namelists, clusters, and authentication information objects. This section deals with queue managers, queues, and process definitions; for information about administering channel, client connection channel, and listener objects, see the *WebSphere MQ Intercommunications* manual. For information about all the MQSC commands for managing queue manager objects, see the *WebSphere MQ Script (MQSC) Command Reference* manual.

You issue MQSC commands to a queue manager using the **runmqsc** command. (For details of this command, see “runmqsc (run MQSC commands)” on page 258.) You can do this interactively, issuing commands from a keyboard, or you can redirect the standard input device (stdin) to run a sequence of commands from an ASCII text file. In both cases, the format of the commands is the same. (For information about running the commands from a text file, see “Running MQSC commands from text files” on page 32.)

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

- Verify a command without running it, where the MQSC commands are verified on a local queue manager, but are not actually run.
- Run a command on a local queue manager, where the MQSC commands are run on a local queue manager.
- Run a command on a remote queue manager, where the MQSC commands are run on a remote queue manager.

You can also run the command followed by a question mark to display the syntax.

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

WebSphere MQ object names

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

When you issue MQSC commands, you need specify only 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.

Case-sensitivity in MQSC commands

MQSC commands, including their attributes, can be written in uppercase or lowercase. Object names in MQSC commands are folded to uppercase (that is, QUEUE and queue are not differentiated), unless the names are enclosed within single quotation marks. If quotation marks are not used, the object is processed with a name in uppercase. See the *WebSphere MQ Script (MQSC) Command Reference* manual for more information.

The **runmqsc** command invocation, in common with all WebSphere MQ control commands, is case sensitive in some WebSphere MQ environments. See “Using control commands” on page 21 for more information.

Redirecting input and output

To improve the ease of migration from other operating systems to OpenVMS, WebSphere MQ 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 source for SYS\$OUTPUT
- 2> specifies the source for SYS\$error

This feature is also included with the executable versions of the sample programs. However, it is not included in the source of the samples and, therefore, is not available if you rebuild the samples from the source code.

Running MQSC commands interactively

To use MQSC commands interactively, at a DCL prompt enter:

```
runmqsc
```

In this command, a queue manager name has not been specified, so the MQSC commands are processed by the default queue manager. If you want to use a different queue manager, specify the queue manager name on the **runmqsc** command. For example, to run MQSC commands on queue manager jupiter.queue.manager, use the command:

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

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

Now you can type in any MQSC commands, as required. For example, try this one:

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

For commands that have too many parameters to fit on one line, use continuation characters to indicate that a command is continued on the following line:

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

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

Feedback from MQSC commands

When you issue MQSC commands, the queue manager returns operator messages that confirm your actions or tell you about the errors you have made. For example:
AMQ8006: WebSphere MQ queue created.

This message confirms that a queue has been created.

AMQ8405: Syntax error detected at or near end of command segment below:-

AMQ8426: Valid MQSC commands are:

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

This message 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 *WebSphere MQ Script (MQSC) Command Reference* manual for the correct syntax.

Ending interactive input of MQSC commands

To stop working with MQSC commands, enter the END command.

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

Running MQSC commands from text files

Running MQSC commands interactively is suitable for quick tests, but if you have very long commands, or are using a particular sequence of commands repeatedly, consider redirecting `sys$input` from a text file. (See “Redirecting input and output” on page 31 for information about `sys$input` and `stdout`). To do this, first create a text file containing the MQSC commands using your usual text editor. When you

use the **runmqsc** command, use the redirection operators. For example, the following command runs a sequence of commands contained in the text file `myprog.in`:

```
runmqsc < myprog.in
```

Similarly, you can also redirect the output to a file. A file containing the MQSC commands for input is called an *MQSC command file*. The output file containing replies from the queue manager is called the *output 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 they do not specify a queue manager name, the MQSC commands run against the default queue manager. The output is sent to the text file `myprog.out`. Figure 2 shows an extract from the MQSC command file `myprog.in` and Figure 3 on page 34 shows the corresponding extract of the output in `myprog.out`.

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 2 is an extract from an MQSC command file showing an MQSC command (`DEFINE QLOCAL`) with its attributes. A description of each MQSC command and its syntax is contained in the *WebSphere MQ Script (MQSC) Command Reference* manual.

```
.  
. .  
. .  
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 2. Extract from the MQSC command file, `myprog.in`

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

For WebSphere MQ for HP OpenVMS, you must limit lines to a maximum of 80 characters, including the continuation character. 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 WebSphere MQ 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 3. 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: WebSphere MQ queue created.
- Other messages resulting from general errors when running the script file.
- A brief statistical summary of the report indicating the number of commands read, the number of commands with syntax errors, and the number of commands that could not be processed.

Note: The queue manager attempts to process only those commands that have no syntax errors.

```
Starting WebSphere MQ 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: WebSphere MQ queue created.
:
.
.
15 MQSC commands read.
0 commands have a syntax error.
0 commands cannot be processed.
```

Figure 3. Extract from the MQSC report file, *myprog.out*.

Running the supplied MQSC command files

When you install WebSphere MQ for HP OpenVMS, the following MQSC command file is supplied:

amqscos0.tst

Definitions of objects used by sample programs.

The file is located in the directory MQS_EXAMPLES:

The command that runs this file is:

```
runmqsc < amqscos0.tst >test.out
```

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. You can then 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 2 on page 33.

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

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

the **-w** flag, which you use to indicate that the queue manager is remote, is ignored, and the command is run locally in verification mode; 30 is the number of seconds that WebSphere MQ waits for replies from the remote queue manager.

Resolving 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 to:

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

To do this, look in the configuration file `mq.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.

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

```
AMQ8146: WebSphere MQ queue manager not available.
```

To correct this type of problem, see "Starting a queue manager" on page 26.

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

```
AMQ8118: WebSphere MQ queue manager does not exist.
```

4. Use the > indirection operator, if you redirect output to a file. 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.
5. Start the queue manager if it is not already started; you get an error message if it is already started.

When you use the **runmqsc** command, the following restrictions also apply:

- You cannot specify an MQSC command as a **runmqsc** parameter. For example, the following is invalid:
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
0790997, 5724-A38 (C) Copyright IBM Corp. 1996, 2004 ALL RIGHTS RESERVED.
Starting WebSphere MQ Commands.
```

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

AMQ8426: Valid MQSC commands are:

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

CANCEL

```
One MQSC command read.
One command has a syntax error.
All valid MQSC commands were processed.
$
```

See also “If you have problems using MQSC commands remotely” on page 65.

Working with queue managers

This section contains examples of some MQSC commands that you can use to display or alter queue manager attributes. See the *WebSphere MQ Script (MQSC) Command Reference* manual for detailed information about these commands.

Displaying queue manager attributes

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

```
DISPLAY QMGR
```

Typical output from this command is shown in Figure 4.

```
DISPLAY QMGR
  1 : DISPLAY QMGR
AMQ8408: Display Queue Manager details.
QMNAME(SATURN)                ACCTCONO(DISABLED)
ACCTINT(1800)                  ACCTMQI(OFF)
ACCTQ(OFF)                     ACTIVREC(MSG)
ALTDATE(2005-02-09)           ALTTIME(17.21.40)
AUTHOREV(DISABLED)           CCSID(850)
CHAD(DISABLED)                CHADEV(DISABLED)
CHADEXIT( )                   CHLEV(DISABLED)
CLWLDATA( )                   CLWLEXIT( )
CLWLEN(100)                   CLWLMRUC(999999999)
CLWLUSEQ(LOCAL)              CMDLEVEL(600)
COMMANDQ(SYSTEM.ADMIN.COMMAND.QUEUE) CRDATE(2005-02-09)
CRTIME(17.21.40)              DEADQ( )
DEFXMITQ( )                   DESCR( )
DISTL(YES)                    INHIBTEV(DISABLED)
IPADDRV(IPV4)                 LOCALEV(DISABLED)
LOGGEREV(DISABLED)           MAXHANDS(256)
MAXMSGL(4194304)             MAXPRTY(9)
MAXUMSGS(10000)              MONACLS(QMGR)
MONCHL(OFF)                   MONQ(OFF)
PERFMEV(DISABLED)           PLATFORM(WINDOWSNT)
QMID(SATURN_2005-02-09_02.00.31) REMOTEEV(DISABLED)
REPOS( )                      REPOSNL( )
ROUTEREC(MSG)                 SCHINIT(QMGR)
SCMDSERV(QMGR)                SSLCRLNL( )
SSLCRYP( )                    SSLEV(DISABLED)
SSLFIPS(NO)
SSLKEYR(C:\Program Files\IBM\WebSphere MQ\Qmgrs\saturn\ssl\key)
SSLRKEYC(0)                   STATACLS(QMGR)
STATCHL(OFF)                  STATINT(1800)
STATMQI(OFF)                  STATQ(OFF)
STRSTPEV(ENABLED)            SYNCPT
TRIGINT(999999999)
```

Figure 4. Typical output from a `DISPLAY QMGR` command

The ALL parameter (the default) on the `DISPLAY QMGR` command displays all the queue manager attributes. In particular, the output tells you the default queue manager name (`saturn.queue.manager`), the dead-letter queue name (`SYSTEM.DEAD.LETTER.QUEUE`), and the command queue name (`SYSTEM.ADMIN.COMMAND.QUEUE`).

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

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

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

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.

Working with local queues

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

Defining a local queue

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

Use the MQSC command `DEFINE QLOCAL` to create a local queue. You can also use the default defined in the default local queue definition, or you can modify the queue characteristics from those of the default local queue.

Note: The default local queue is named `SYSTEM.LOCAL.DEFAULT.QUEUE` and it was created on system installation.

Using the MQSC command shown below, we define a queue called `ORANGE.LOCAL.QUEUE`, with the following characteristics:

- It is enabled for gets, enabled for puts, and operates on a priority order basis.
- It is an *normal* queue; it is not an initiation queue or transmission queue, and it does not generate trigger messages.
- The maximum queue depth is 5000 messages; the maximum message length is 4194304 bytes.

```
DEFINE QLOCAL (ORANGE.LOCAL.QUEUE) +  
  DESCR('Queue for messages from other systems') +  
  PUT (ENABLED) +  
  GET (ENABLED) +  
  NOTRIGGER +  
  MSGDLVSQ (PRIORITY) +  
  MAXDEPTH (5000) +  
  MAXMSGL (4194304) +  
  USAGE (NORMAL);
```

Note:

1. With the exception of the value for the description, all the attribute values shown are the default values. We have shown them 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 39.

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 40.

Defining a dead-letter queue

We recommend that each queue manager has 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 tell the queue manager about the dead-letter queue. You do this by specifying a dead-letter queue name on the **crtmqm** command (`crtmqm -u DEAD.LETTER.QUEUE`, for example), or by using the DEADQ attribute on the ALTER QMGR command to specify one later. You must define the dead-letter queue before using it.

We supply a sample dead-letter queue called SYSTEM.DEAD.LETTER.QUEUE with the product. This queue is automatically created when you create the queue manager. You can modify this definition if required, and rename it.

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

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

WebSphere MQ provides a dead-letter queue handler that allows you to specify how messages found on a dead-letter queue are to be processed or removed. For further information, see Chapter 9, “The WebSphere MQ dead-letter queue handler,” on page 125.

Displaying default object attributes

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

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

The syntax of this command is different from that of the corresponding DEFINE command. On the DISPLAY command you can give just the queue name, whereas on the DEFINE command you have to specify the type of the queue, that is, QLOCAL, QALIAS, QMODEL, or QREMOTE.

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)          TYPE(QLOCAL)
  CURDEPTH(0)                        MAXDEPTH(5000)
  MAXMSGL(4194304)
```

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. Enter the name of the queue to be copied *exactly* as it was entered when you created the queue. If the name contains lower case characters, enclose the name in single quotation marks.

You can also use this form of the DEFINE command to copy a queue definition, but 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 4194304.

Note:

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 38, we defined the queue called ORANGE.LOCAL.QUEUE. Suppose, for example, you want to decrease 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 also 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.

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)
```

Note: There is no prompt that enables you to change your mind; once you press the Enter key the messages are lost.

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 be deleted only if you specify the PURGE option. For example:

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

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

Browsing queues

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

Source

```
MQS_EXAMPLES:AMQSBCG0.C
```

Executable

```
[.BIN]AMQSBCG.EXE, under  
MQS_EXAMPLES:
```

The sample takes two parameters, which are the:

- Queue name, for example, SYSTEM.ADMIN.RESPQ.TEST.
- Queue manager name, for example, JJJH

as shown in the following command:

```
amqsbcg "SYSTEM.ADMIN.RESPQ.TEST" "JJJH"
```

There are no defaults; both parameters are required. Typical results from this commands are shown in Figure 5 on page 42.

Working with alias queues

An alias queue provides a way of referring indirectly to another queue. The other queue can be either:

- A local queue (see “Defining a local queue” on page 38)
- A local definition of a remote queue (see “Creating a local definition of a remote queue” on page 65)

An alias queue is not a real queue, but a definition that resolves to a real (or target) queue at run time. The alias queue definition specifies the target queue. When an application makes an **MQOPEN** call to an alias queue, the queue manager resolves the alias to the target queue name. An alias queue cannot resolve to another alias queue.

Alias queues are useful for:

- Giving different applications different levels of access authorities to the target queue.
- Allowing different applications to work with the same queue in different ways. (Perhaps you want to assign different default priorities or different default persistence values.)
- Simplifying maintenance, migration, and workload balancing. (Perhaps you want to change the target queue name without having to change your application, which continues to use the alias.)

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

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

Defining an alias queue

The following command creates an alias queue:

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

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

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

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

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

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

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

The following command defines an alias that is put enabled and get disabled for application ALPHA:

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

The following command defines an alias that is put disabled and get enabled for application BETA:

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

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

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

Using other commands with alias queues

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

Use the following command to display the alias queue's attributes:

```
DISPLAY QUEUE (ALPHAS.ALIAS.QUEUE)
```

Use the following command to alter the base queue name, to which the alias resolves, where the force option forces the change even if the queue is open:

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

Use the following command to delete this queue alias:

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

You cannot delete an alias queue if an application currently has the queue open. See the *WebSphere MQ Script (MQSC) Command Reference* manual for more information about this and other alias queue 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 required.

Defining a model queue

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

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

This command creates a model queue definition. From the DEFTYPE attribute, you can see that the actual queues created from this template are permanent dynamic queues. Any 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 to delete the model queue object. For example:

Use the following command to display the model queue's attributes:

```
DISPLAY QUEUE (GREEN.MODEL.QUEUE)
```

Use the following command to alter the model to enable puts on any dynamic queue created from this model:

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

Use the following command to delete this model queue:

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

Working with services

Service objects are a means by which additional processes can be managed as part of a queue manager. With services you can define programs that are started and stopped when the queue manager starts and ends.

Service objects can be either of the following types:

Server A server is a service object that has the parameter SERVTYPE specified as SERVER. A server service object is the definition of a program that is executed when a specified queue manager is started. Server service objects

define programs that typically run for a long period of time. For example, a server service object can be used to execute a trigger monitor process, such as **runmqtrm**.

Only one instance of a server service object can run concurrently. The status of running server service objects can be monitored using the MQSC command, **DISPLAY SVSTATUS**.

Command

A command is a service object that has the parameter **SERVTYPE** specified as **COMMAND**. Command service objects are similar to server service objects, however multiple instances of a command service object can run concurrently and their status can not be monitored using the MQSC command **DISPLAY SVSTATUS**.

If the MQSC command, **STOP SERVICE**, is executed no check is made to determine whether the program started by the MQSC command, **START SERVICE**, is still active before executing the stop program.

Defining a service object

The attributes used to define a service object are:

SERVTYPE

Defines the type of the service object. Possible values are:

SERVER

A server service object.

Only one instance of a server service object can be executed at a time. The status of server service objects can be monitored using the MQSC command, **DISPLAY SVSTATUS**.

COMMAND

A command service object.

Multiple instances of a command service object can be executed concurrently. The status of a command service objects cannot be monitored.

STARTCMD

The program that is executed to start the service. A fully qualified path to the program must be specified.

STARTARG

Arguments passed to the start program.

STDERR

Specifies the path to a file to which the standard error (stderr) of the service program should be redirected.

STDOUT

Specifies the path to a file to which the standard output (stdout) of the service program should be redirected.

STOPCMD

The program that is executed to stop the service. A fully qualified path to the program must be specified.

STOPARG

Arguments passed to the stop program.

CONTROL

Specifies how the service is to be started and stopped:

MANUAL

The service is not to be started automatically or stopped automatically. It is controlled by use of the START SERVICE and STOP SERVICE commands. This is the default value.

QMGR

The service being defined is to be started and stopped at the same time as the queue manager is started and stopped.

STARTONLY

The service is to be started at the same time as the queue manager is started, but is not requested to stop when the queue manager is stopped.

Managing services

By using the CONTROL parameter, an instance of a service object can be either started and stopped automatically by the queue manager, or started and stopped using the MQSC commands START SERVICE and STOP SERVICE.

When an instance of a service object is started, a message is written to the queue manager error log containing the name of the service object and the process id of the started process. An example log entry for a server service object starting follows:

```
02/15/2005 11:54:24 AM - Process(10363.1) User(mqm) Program(amqzmgr0)
AMQ5028: The Server 'S1' has started. ProcessId(13031).
```

EXPLANATION:

The Server process has started.

ACTION:

None.

An example log entry for a command service object starting follows:

```
02/15/2005 11:53:55 AM - Process(10363.1) User(mqm) Program(amqzmgr0)
AMQ5030: The Command 'C1' has started. ProcessId(13030).
```

EXPLANATION:

The Command has started.

ACTION:

None.

When an instance server service stops, a message is written to the queue manager error logs containing the name of the service and the process id of the ending process. An example log entry for a server service object stopping follows:

```
02/15/2005 11:54:54 AM - Process(10363.1) User(mqm) Program(amqzmgr0)
AMQ5029: The Server 'S1' has ended. ProcessId(13031).
```

EXPLANATION:

The Server process has ended.

ACTION:

None.

Additional environment variables

When a service is started, the environment in which the service process is started is inherited from the environment of the queue manager. It is possible to define additional environment variables to be set in the environment of the service process by adding the variables you want to define to one of the service.env environment override files.

There are two possible files to which you can add environment variables:

- The machine scope `service.env` file, which is located in `MQS_ROOT` directory
- The queue manager scope `service.env` file, which is located in the queue manager data directory. For example, the location of the environment override file for a queue manager named `QMNAME` is: `MQS_ROOT[QMGRS.QMNAME]SERVICE.ENV`

Both files are processed, if available, with definitions in the queue manager scope file taking precedence over those in the machine scope file.

The format of the variables defined in the file, `service.env`, is a list of name and value variable pairs. Each variable must be defined on a new line, and each variable is taken as it is explicitly defined, including white space. An example of the file, `service.env`, follows:

```
#####  
**                                     **  
** <N_OCO_COPYRIGHT>                 **  
** Licensed Materials - Property of IBM **  
**                                     **  
** 63H9336                             **  
** (C) Copyright IBM Corporation 2005   **  
**                                     **  
** <NOC_COPYRIGHT>                   **  
**                                     **  
#####  
#####  
** Module Name: service.env           **  
** Type      : WebSphere MQ service environment file **  
** Function   : Define additional environment variables to be set **  
**            : for SERVICE programs. **  
** Usage     : <VARIABLE>=<VALUE> **  
**            : **  
#####  
MYLOC=/opt/myloc/bin  
MYTMP=/tmp  
TRACEDIR=/tmp/trace  
MYINITQ=ACCOUNTS.INITIATION.QUEUE
```

Replaceable inserts on service definitions

In the definition of a service object, it is possible to substitute tokens. Tokens that are substituted are automatically replaced with their expanded text when the service program is executed. Substitute tokens can be taken from the following list of common tokens, or from any variables that are defined in the file, `service.env`.

Common tokens

The following are common tokens that can be used to substitute tokens in the definition of a service object:

MQ_INSTALL_PATH

The install location of WebSphere MQ on OpenVMS systems is `MQS_ROOT`.

MQ_DATA_PATH

The location of the WebSphere MQ data directory on OpenVMS systems is `MQS_ROOT`.

QMNAME

The current queue manager name.

MQ_SERVICE_NAME

The name of the service.

MQ_SERVER_PID

This token can only be used by the STOPARG and STOPCMD arguments.

For server service objects this token is replaced with the process id of the process started by the STARTCMD and STARTARG arguments. Otherwise, this token is replaced with 0.

MQ_Q_MGR_DATA_PATH

The location of the queue manager data directory.

MQ_Q_MGR_DATA_NAME

The transformed name of the queue manager. For more information on name transformation, see “Understanding WebSphere MQ file names” on page 17.

To use replaceable inserts, insert the token within + characters into any of the STARTCMD, STARTARG, STOPCMD, STOPARG, STDOUT or STDERR strings. For examples of this, see “Examples on using service objects.”

Examples on using service objects

The following services are written with UNIX® style path separator characters, except where otherwise stated.

Using a server service object

This example shows how to define, use, and alter, a server service object to start a trigger monitor.

1. A server service object is defined, using the following MQSC command:

```
DEFINE SERVICE(S1) +
  CONTROL(QMGR) +
  SERVTYPE(SERVER) +
  STARTCMD('+MQ_INSTALL_PATH+bin/runmqtrm') +
  STARTARG('-m +QMNAME+ -q ACCOUNTS.INITIATION.QUEUE') +
  STOPCMD('+MQ_INSTALL_PATH+bin/amqsstop') +
  STOPARG('-m +QMNAME+ -p +MQ_SERVER_PID+')
```

Where:

+MQ_INSTALL_PATH+ is a token representing the installation directory.

+QMNAME+ is a token representing the name of the queue manager.

ACCOUNTS.INITIATION.QUEUE is the initiation queue.

amqsstop is a sample program provided with WebSphere MQ which requests the queue manager to break all connections for the process id.

amqsstop generates PCF commands, therefore the command server must be running.

+MQ_SERVER_PID+ is a token representing the process id passed to the stop program.

2. An instance of the server service object executes when the queue manager is next started. However, an instance of the server service object starts immediately with the following MQSC command:

```
START SERVICE(S1)
```

3. The status of the server service process is displayed, using the following MQSC command:

```
DISPLAY SVSTATUS(S1)
```

4. This example now shows how to alter the server service object and have the updates picked up by manually restarting the server service process. The server

service object is altered so that the initiation queue is specified as JUPITER.INITIATION.QUEUE. The following MQSC command is used:

```
ALTER SERVICE(S1) +
    STARTARG('-m +QMNAME+ -q JUPITER.INITIATION.QUEUE')
```

Note: A running service does not pick up any updates to its service definition until it is restarted.

5. The server service process is restarted so that the alteration is picked up, using the following MQSC commands:

```
STOP SERVICE(S1)
```

Followed by:

```
START SERVICE(S1)
```

The server service process is restarted and picks up the alterations made in Step 4.

Note: The MQSC command, `STOP SERVICE`, can only be used if a `STOPCMD` argument is specified in the service definition.

Using a command service object

This example shows how to define a command service object to start a program that writes entries to the operating system's system log when a queue manager is started or stopped:

1. The command service object is defined, using the following MQSC command:

```
DEFINE SERVICE(S2) +
    CONTROL(QMGR) +
    SERVTYPE(COMMAND) +
    STARTCMD('/MQS_ROOT/logger') +
    STARTARG('Queue manager +QMNAME+ starting') +
    STOPCMD('/MQS_ROOT/logger') +
    STOPARG('Queue manager +QMNAME+ stopping')
```

Where:

logger is the UNIX supplied command to write to the system log.

+QMNAME+ is a token representing the name of the queue manager.

Using a command service object when a queue manager ends only

This example shows how to define a command service object to start a program that writes entries to the operating system's system log when a queue manager is stopped only:

1. The command service object is defined, using the following MQSC command:

```
DEFINE SERVICE(S3) +
    CONTROL(QMGR) +
    SERVTYPE(COMMAND) +
    STOPCMD('/MQS_ROOT/logger') +
    STOPARG('Queue manager +QMNAME+ stopping')
```

Where:

logger is a sample program provided with WebSphere MQ that can write entries to the operating system's system log.

+QMNAME+ is a token representing the name of the queue manager.

More on passing arguments

This example is written with Windows style path separator characters.

This example shows how to define a server service object to start a program called `runserv` when a queue manager is started. One of the arguments that is to be passed to the starting program is a string containing a space. This argument needs to be passed as a single string. To achieve this, double quotes are used as shown in the following command to define the command service object:

1. The server service object is defined, using the following MQSC command:

```
DEFINE SERVICE(S1) SERVTYPE(SERVER) CONTROL(QMGR) +
  STARTCMD('C:\Program Files\Tools\runserv.exe') +
  STARTARG('-m +QMNAME+ -d "MQS_ROOT"') +
  STDOUT('MQS_ROOT\+MQ_SERVICE_NAME+.out')
```

Where:

+QMNAME+ is a token representing the name of the queue manager.

"MQS_ROOT" is a string containing a space, which will be passed as a single string.

Autostarting a Service

This example shows how to define a server service object that can be used to automatically start the Trigger Monitor when the queue manager starts.

1. The server service object is defined, using the following MQSC command:

```
DEFINE SERVICE(TRIG_MON_START) +
  CONTROL(QMGR) +
  SERVTYPE(SERVER) +
  STARTCMD('runmqtrm') +
  STARTARG('-m +QMNAME+ -q +IQUEUE+')
```

Where:

+QMNAME+ is a token representing the name of the queue manager.

+IQUEUE+ is an environment variable defined by the user in one of the `service.env` files representing the name of the initiation queue.

Managing objects for triggering

WebSphere MQ enables you to start an application automatically when certain conditions on a queue are met. For example, you might want to start an application when the number of messages on a queue reaches a specified number. This facility is called *triggering* and is described in detail in the *WebSphere MQ Application Programming Guide*.

This section tells you how to set up the required objects to support triggering on WebSphere MQ.

Defining an application queue for triggering

An application queue is a local queue that is used by applications for messaging, through the MQI. Triggering requires a number of queue attributes to be defined on the application queue. Triggering itself is enabled by the *Trigger* attribute (TRIGGER in MQSC commands).

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

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

where:

QLOCAL (MOTOR.INSURANCE.QUEUE)

Is the name of the application queue being defined.

PROCESS (MOTOR.INSURANCE.QUOTE.PROCESS)

Is the name of the process definition that defines the application to be started by a trigger monitor program.

MAXMSGL (2000)

Is the maximum length of messages on the queue.

DEFPSIST (YES)

Specifies that messages on this queue are persistent by default.

INITQ (MOTOR.INS.INIT.QUEUE)

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

TRIGGER

Is the trigger attribute value.

TRIGTYPE (DEPTH)

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

TRIGDPTH (100)

Is the number of messages required to generate a trigger event.

TRIGMPRI (5)

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

Defining an initiation queue

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

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

Defining a process

Use the DEFINE PROCESS command to create a process definition. A process definition defines the application to be used to process messages from the application queue. The application queue definition names the process to be used and thereby associates the application queue with the application to be used to process its messages. This is done through the PROCESS attribute on the application queue MOTOR.INSURANCE.QUEUE. The following MQSC command defines the required process, MOTOR.INSURANCE.QUOTE.PROCESS, identified in this example:

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

Where:

MOTOR.INSURANCE.QUOTE.PROCESS

Is the name of the process definition.

DESCR ('Insurance request message processing')

Describes the application program to which this definition relates. This text is displayed when you use the DISPLAY PROCESS command. This can help you to identify what the process does. If you use spaces in the string, you must enclose the string in single quotation marks.

APPLTYPE (OPENVMS)

Is the type of application that runs on OpenVMS.

APPLICID ('DKA0:[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 attributes of a 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)          ENVRDATA()+
PROCESS(MOTOR.INS.PROC)            ALTDATE(2004-08-25)
ALTTIME(17.30.44)                  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 5. Automating administration tasks

This chapter assumes that you have experience of administering WebSphere MQ objects.

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

The section “PCF commands” describes how to use programmable command formats to automate administration tasks.

PCF commands

The purpose of WebSphere MQ programmable command format (PCF) commands is to allow administration tasks to be programmed into an administration program. In this way, from a program you can manipulate queue manager objects (queues, process definitions, namelists, channels, client connection channels, listeners, services, and authentication information objects), and even manipulate the queue managers themselves.

PCF commands cover the same range of functions provided by MQSC commands. You can write a program to issue PCF commands to any queue manager in the network from a single node. In this way, you can both centralize and automate administration tasks.

Each PCF command is a data structure that is embedded in the application data part of a WebSphere MQ message. Each command is sent to the target queue manager using the MQI function **MQPUT** in the same way as any other message. Providing the command server is running on the queue manager receiving the message, the command server 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 needed to create a PCF command message:

Message descriptor

This is a standard WebSphere MQ message descriptor, in which:

- Message type (*MsgType*) is MQMT_REQUEST.
- Message format (*Format*) is MQFMT_ADMIN.

Application data

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

- The PCF message type (*Type*) specifies MQCFT_COMMAND.
- The command identifier specifies the command, for example, *Change Queue* (MQCMD_CHANGE_Q).

For a complete description of the PCF data structures and how to implement them, see the *WebSphere MQ Programmable Command Formats and Administration Interface* manual.

PCF object attributes

Object attributes in PCF are not limited to eight characters as they are for MQSC commands. They are shown in this book in italics. For example, the PCF equivalent of RQMNAME is *RemoteQMgrName*.

Escape PCFs

Escape PCFs are PCF commands that contain MQSC commands within the message text. You can use PCFs to send commands to a remote queue manager. For more information about using escape PCFs, see the *WebSphere MQ Programmable Command Formats and Administration Interface* manual.

Using the MQAI to simplify the use of PCFs

The MQAI is an administration interface to WebSphere MQ that is available on the HP OpenVMS platform.

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

Use the MQAI:

To simplify the use of PCF messages

The MQAI is an easy way to administer WebSphere MQ; you do not have to write your own PCF messages, avoiding the problems associated with complex data structures.

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

Programs written using the MQAI pass parameters into the appropriate data bag and you need only one statement for each structure. The use of MQAI data bags removes the need for you to handle arrays and allocate storage, and provides some degree of isolation from the details of the PCF.

To implement self-administering applications and administration tools

For example, the Active Directory Services provided by WebSphere MQ for Windows Version 5.3 uses the MQAI. (There is currently no example of this usage on the OpenVMS platform.)

To handle error conditions more easily

It is difficult to get return codes back from PCF commands, but the MQAI makes it easier for the program to handle error conditions.

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

For more information about using the MQAI, and PCFs in general, see the *WebSphere MQ Programmable Command Formats and Administration Interface* manual.

Command servers

Each queue manager can have a command server associated with it. A command server processes any incoming commands from remote queue managers, or PCF commands from applications.

For more information about command servers see “Managing the command server for remote administration” on page 63 in “Administering remote WebSphere MQ objects”

Administering remote WebSphere MQ objects

This section tells you how to administer WebSphere MQ objects on a remote queue manager using MQSC commands, and how to use remote queue objects to control the destination of messages and reply messages.

This chapter describes:

- “Channels, clusters, and remote queuing”
- “Remote administration from a local queue manager” on page 59
- “Creating a local definition of a remote queue” on page 65
- “Using remote queue definitions as aliases” on page 68
- “Data conversion” on page 69

Channels, clusters, and remote queuing

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

- On the same machine
- On another machine in the same location (or even on the other side of the world)
- Running on the same platform as the local queue manager
- Running on another platform supported by WebSphere MQ

These messages might originate from:

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

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

- At least one channel
- A transmission queue

- A message channel agent (MCA)
- A channel listener
- A channel initiator

For a remote queue manager to receive a message, a listener is required.

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

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

If you want the remote queue manager to respond to messages sent by the local queue manager, set up a second channel to send responses back to the local queue manager.

Use the MQSC command `DEFINE CHANNEL` to define channels. In this chapter, the examples relating to channels use the default channel attributes unless otherwise specified.

There is a message channel agent (MCA) at each end of a channel, controlling the sending and receiving of messages. The MCA takes messages from the transmission queue and puts them on the communication link between the queue managers.

A transmission queue is a specialized local queue that temporarily holds messages before the MCA picks them up and sends them to the remote queue manager. You specify the name of the transmission queue on a *remote queue definition*.

You can allow an MCA to transfer messages using multiple threads. This process is known as *pipelining*. Pipelining enables the MCA to transfer messages more efficiently, improving channel performance. See “The Channels stanza” on page 83 for details of how to configure a channel to use pipelining.

“Preparing channels and transmission queues for remote administration” on page 60 tells you how to use these definitions to set up remote administration.

For more information about setting up distributed queuing in general, see the *WebSphere MQ Intercommunications* manual.

Remote administration using clusters

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

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

be set up easily, and typically contain queue managers that are logically related in some way and need to share data or applications. Even the smallest cluster reduces system administration overheads.

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

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

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

Remote administration from a local queue manager

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

Preparing the queues and channels is essentially the same for both MQSC and PCF commands. In this book, the examples show MQSC commands, because they are easier to understand. For more information about writing administration programs using PCF commands, see the *WebSphere MQ Programmable Command Formats and Administration Interface* manual.

You send MQSC commands to a remote queue manager either interactively or from a text file containing the commands. The remote queue manager might be on the same machine or, more typically, on a different machine. You can remotely administer queue managers in other WebSphere MQ environments, including AIX , AS/400®, and MVS/ESA™ .

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

Preparing queue managers for remote administration

Figure 6 on page 60 shows the configuration of queue managers and channels that you need for remote administration using the `runmqsc` command. The object `source.queue.manager` is the source queue manager from which you can issue MQSC commands and to which the results of these commands (operator messages) are returned. The object `target.queue.manager` is the name of the target queue manager, which processes the commands and generates any operator messages.

Note: If you are using `runmqsc` with the `-w` option, `source.queue.manager` *must* be the default queue manager. For further information on creating a queue manager, see “`crtmqm` (create queue manager)” on page 205.

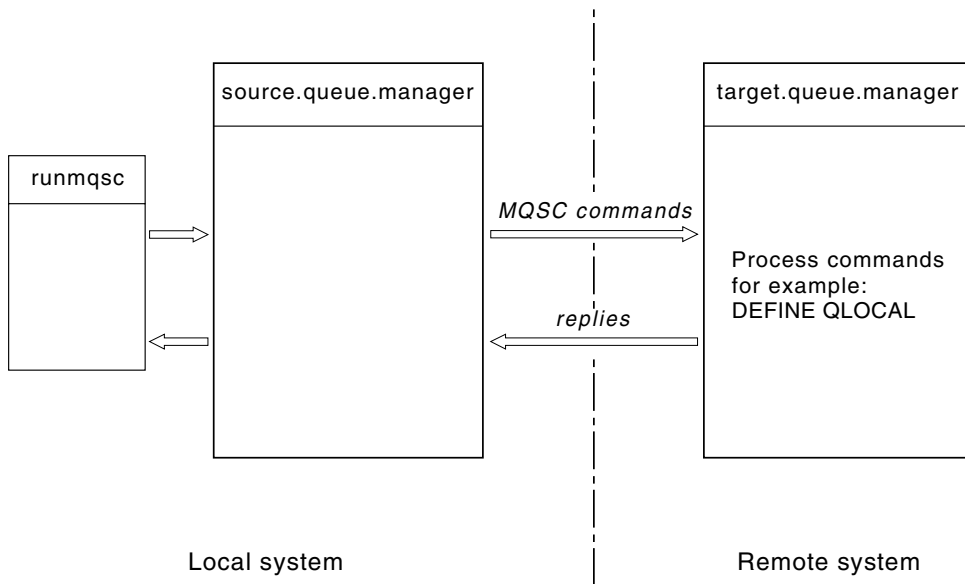


Figure 6. Remote administration using MQSC commands

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

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

On the target queue manager:

- The command queue, `SYSTEM.ADMIN.COMMAND.QUEUE`, must be present. This queue is created by default when a queue manager is created.

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

Preparing channels and transmission queues for remote administration

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

The channel `source.to.target` is for sending MQSC commands from the source queue manager to the target queue manager. Its sender is at `source.queue.manager` and its receiver is at `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 channel. This queue is a local queue that is given the name of the receiving queue manager. The XMITQ name must match the remote queue manager name in order for remote administration to work, unless you are using a queue manager alias. Figure 7 on page 61 summarizes this configuration.

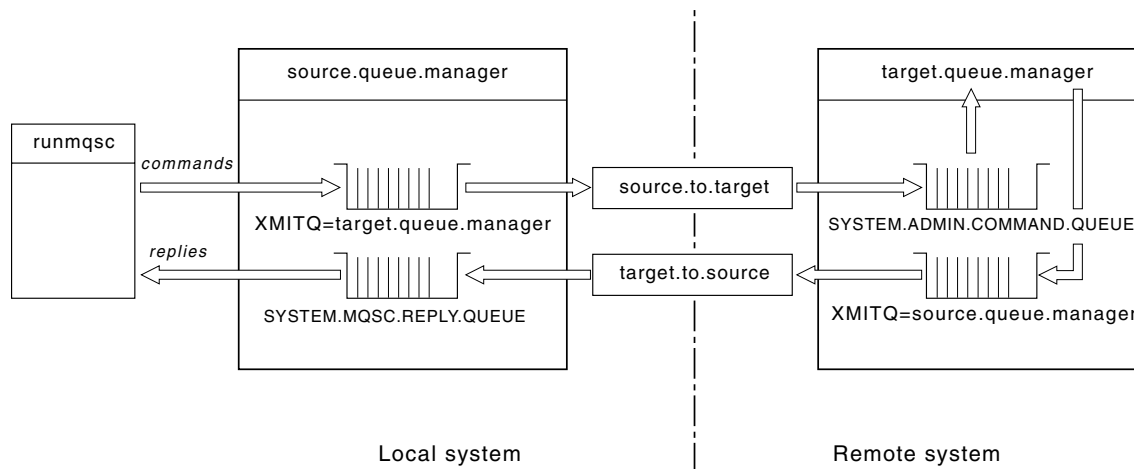


Figure 7. Setting up channels and queues for remote administration

See the *WebSphere MQ Intercommunications* manual for more information about setting up channels.

Defining channels and transmission queues:

On the source queue manager (`source.queue.manager`), issue the following MQSC commands to define the channels, listener, and the transmission queue:

1. Define the sender channel at the source queue manager:

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

2. Define the receiver channel at the source queue manager:

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

3. Define the transmission queue on the source queue manager:

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

Issue the following commands on the target queue manager (`target.queue.manager`), to create the channels, listener, and the transmission queue:

1. Define the sender channel on the target queue manager:

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

2. Define the receiver channel on the target queue manager:

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

3. Define the transmission queue on the target 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.

Starting the listeners and channels:

Start both listeners by using the following MQSC commands:

1. Start the listener on the source queue manager, `source.queue.manager`, by issuing the following MQSC command:

```
START LISTENER ('source.queue.manager')
```

Or at a DCL prompt enter:

```
runmq1sr -m "source.queue.manager" -t tcp
```

2. Start the listener on the target queue manager, `target.queue.manager`, by issuing the following MQSC command:

```
START LISTENER ('target.queue.manager')
```

Or at a DCL prompt enter:

```
runmq1sr -m "target.queue.manager" -t tcp
```

Start both sender channels by using the following MQSC commands:

1. Start the sender channel on the source queue manager, `source.queue.manager`, by issuing the following MQSC command:

```
START CHANNEL ('source.to.target')
```

Or at a DCL prompt enter:

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

2. Start the sender channel on the target queue manager, `target.queue.manager`, by issuing the following MQSC command:

```
START CHANNEL ('target.to.source')
```

Or at a DCL prompt enter:

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

The **runmq1sr** and **runmqchl** commands are WebSphere MQ control commands. They cannot be issued using **runmqsc**.

Automatic definition of channels:

If WebSphere MQ receives an inbound attach request and cannot find an appropriate receiver or server-connection channel, it creates a channel automatically. Automatic definitions are based on two default definitions supplied with WebSphere MQ: `SYSTEM.AUTO.RECEIVER` and `SYSTEM.AUTO.SVRCONN`.

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

For more information about creating channel definitions automatically, see the *WebSphere MQ Intercommunications* manual. For information about automatically defining channels for clusters, see the *WebSphere MQ Queue Manager Clusters* manual.

Managing the command server for remote administration

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

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

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

Starting the command server:

Depending on the value of the queue manager attribute, *SCMDSERV*, the command server is either started automatically when the queue manager starts, or must be started manually. The value of the queue manager attribute can be altered using the MQSC command ALTER QMGR specifying the parameter *SCMDSERV*. By default, the command server is started automatically.

If *SCMDSERV* is set to *MANUAL*, start the command server using the command:

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

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

Displaying the status of the command server:

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

To display the status of the command server for a queue manager, issue the following MQSC command:

```
DISPLAY QMSTATUS CMDSERV
```

Stopping a command server:

To end the command server started by the previous example use the following command:

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

You can stop the command server in two ways:

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

Note: Stopping a queue manager also ends the command server associated with it.

Issuing MQSC commands on a remote queue manager

The command server *must* be running on the target queue manager, if it is going to process MQSC commands remotely. (This is not necessary on the source queue manager). For information on how to start the command server on a queue manager, see “Starting the command server” on page 63.

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

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

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

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

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

When you stop issuing MQSC commands, the local queue manager displays any timed-out responses that have arrived and discards any further responses.

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

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

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

Working with queue managers on MVS/ESA:

You can issue MQSC commands to an MVS/ESA queue manager from a WebSphere MQ for HP 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 issuing commands remotely

When you are issuing commands on a remote queue manager:

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

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

3. Check that the command file runs locally without error.
4. Run the command file against the remote system.

If you have problems using MQSC commands remotely

If you have difficulty in running MQSC commands remotely, make sure that you have:

- Started the command server on the target queue manager.
- Defined a valid transmission queue.
- Defined the two ends of the message channels for both:
 - The channel along which the commands are being sent.
 - The channel along which the replies are to be returned.
- Specified the correct connection name (CONNNAME) 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.
- Sent requests from a source queue manager that do not make sense to the target queue manager (for example, requests that include parameters that are not supported on the remote queue manager).

See also “Resolving problems with MQSC” on page 35.

Creating a local definition of a remote queue

A local definition of a remote queue is a definition on a local queue manager that refers to a queue on a remote queue manager.

You do not have to define a remote queue from a local position, but the advantage of doing so is that applications can refer to the remote queue by its locally-defined name instead of having to specify a name that is qualified by the ID of the queue manager on which the remote queue is located.

Understanding how local definitions of remote queues work

An application connects to a local queue manager and then issues an `MQOPEN` call. In the open call, the queue name specified is that of a remote queue definition on the local queue manager. The remote queue definition supplies the names of the target queue, the target queue manager, and optionally, a transmission queue. To put a message on the remote queue, the application issues an `MQPUT` call, specifying the handle returned from the `MQOPEN` call. The queue manager uses the remote queue name and the remote queue manager name in a transmission header at the start of 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 needs to put a message on a queue owned by a remote queue manager.

How it works:

The application connects to a queue manager, for example, saturn.queue.manager. The target queue is owned by another queue manager.

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

Field value	Description
<i>ObjectName</i> CYAN.REMOTE.QUEUE	Specifies the local name of the remote queue object. This defines the target queue and the target 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 exists.)

After this, the application issues an **MQPUT** call to put a message onto 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')

Provides additional text that describes the use of the queue.

RNAME (AUTOMOBILE.INSURANCE.QUOTE.QUEUE)

Specifies the name of the target queue on the remote queue manager. This is the real target queue for messages sent by applications that specify the

queue name CYAN.REMOTE.QUEUE. The queue AUTOMOBILE.INSURANCE.QUOTE.QUEUE must be defined as a local queue on the remote queue manager.

RQMNAME (jupiter.queue.manager)

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

XMITQ (INQUOTE.XMIT.QUEUE)

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

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

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, including the remote queue manager name, as part of the **MQOPEN** call. In this case, you do not need a local definition of a remote queue. However, this means that applications must either know, or have access to, the name of the remote queue manager at run time.

Using other commands with remote queues

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

- To display the remote queue's attributes:
DISPLAY QUEUE (CYAN.REMOTE.QUEUE)
- To change the remote queue to enable puts. This does not affect the target queue, only applications that specify this remote queue:
ALTER QREMOTE (CYAN.REMOTE.QUEUE) PUT(ENABLED)
- To delete this remote queue. This does not affect the target queue, only its local definition:
DELETE QREMOTE (CYAN.REMOTE.QUEUE)

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

Defining 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 MQSC command attribute USAGE defines whether a queue is a transmission queue or a normal queue.

Default transmission queues:

When a queue manager sends messages to a remote queue manager, it identifies the transmission queue using the following sequence:

1. The transmission queue named on the XMITQ attribute of the local definition of a remote queue.
2. A transmission queue with the same name as the target queue manager. (This value is the default value on XMITQ of the local definition of a remote queue.)
3. The transmission queue named on the DEFXMITQ attribute of the local queue manager.

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

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

Applications can put messages directly on a transmission queue, or indirectly through a remote queue definition. See also “Creating a local definition of a remote queue” on page 65.

Using remote queue definitions as aliases

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

- Queue manager aliases
- Reply-to queue aliases

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

You must set up the appropriate channels for the message to arrive at its destination.

Queue manager aliases

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

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

For more information about queue manager aliases, see the *WebSphere MQ Intercommunications* manual.

Reply-to queue aliases

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

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

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

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

For more information about request messages, reply messages, and reply-to queues, see the *WebSphere MQ Application Programming Guide*.

For more information about reply-to queue aliases, see the *WebSphere MQ Intercommunications* manual.

Data conversion

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

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

For EBCDIC new line (NL) character conversions to ASCII, see “AllQueueManagers stanza” on page 73.

Supported conversions are defined in the *WebSphere MQ Application Programming Reference* manual.

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

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

File `ccsid.tbl`

The file `ccsid.tbl` specifies:

- Any additional code sets. To specify additional code sets, you need to edit `ccsid.tbl` (guidance on how to do this is provided in the file).
- Any default data conversion.

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

In WebSphere MQ for HP OpenVMS, a sample `ccsid.tbl` is provided as `MQS_EXAMPLES:CCSID.TBL`, and the active `ccsid.tbl` file is located in directory `MQS_ROOT:[MQM.CONV.TABLE]`

Default data conversion:

To implement default data conversion, edit the `ccsid.tbl` file to specify a default EBCDIC CCSID and a default ASCII CCSID, and also to specify the defaulting CCSIDs. Instructions on how to do this are included in the file.

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

The default data-conversion process is as follows:

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

Note: Try to restrict the characters being converted to those that have the same code values in the coded character set specified for the message and in the default coded character set. Using only the set of characters that is valid for WebSphere MQ object names satisfies this requirement, in general. Exceptions occur with EBCDIC CCSIDs 290, 930, 1279, and 5026 used in Japan, where the lowercase characters have different codes from those used in other EBCDIC CCSIDs.

Converting messages in user-defined formats

The queue manager cannot convert messages in user-defined formats from one coded character set to another. If you need to convert data in a user-defined format, you must supply a data-conversion exit for each such format. Do **not** use default CCSIDs to convert character data in user-defined formats. For more information about converting data in user-defined formats and about writing data conversion exits, see the *WebSphere MQ Application Programming Guide*.

Changing the queue manager CCSID

When you have used the CCSID attribute of the ALTER QMGR command to change the CCSID of the queue manager, stop and restart the queue manager to ensure that all running applications, including the command server and channel programs, are stopped and restarted.

This is necessary, because any applications that are running when the queue manager CCSID is changed continue to use the existing CCSID.

Chapter 6. Configuring WebSphere MQ

This chapter tells you how to change the behavior of WebSphere MQ or an individual queue manager to suit your installation's needs.

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

This chapter describes:

- The attributes you can modify in “WebSphere MQ configuration files”
- The attributes you can use to modify WebSphere MQ configuration (for all queue managers) in “Attributes for changing WebSphere MQ configuration information” on page 73
- The attributes you can use to modify the configuration of an individual queue manager in “Changing queue manager configuration information” on page 79
- Examples of `mqs.ini` and `qm.ini` files for WebSphere MQ for HP OpenVMS in “API exits” on page 86

WebSphere MQ configuration files

Users of WebSphere MQ for HP OpenVMS modify WebSphere MQ configuration attributes within:

- A WebSphere MQ configuration file (`mqs.ini`) to make changes to WebSphere MQ on the node as a whole. There is one `mqs.ini` file per node.
- A queue manager configuration file (`qm.ini`) to make changes to specific queue managers. There is one `qm.ini` file for each queue manager on the node.

A configuration file (which may also be referred to as a *stanza* file or *.ini* file) contains one or more stanzas, which are simply groups of lines in the file that together have a common function or define part of a system, for example, log functions, channel functions, and installable services.

Any changes you make to a configuration file do not take effect until the next time the queue manager is started.

Editing configuration files

Before attempting to edit a configuration file, back it up so that you have a copy you can revert to if the need arises!

You can edit configuration files either:

- Automatically, using commands that change the configuration of queue managers on the node
- Manually, using a standard text editor

You can edit the default values in the WebSphere MQ configuration files after installation.

If you set an incorrect value on a configuration file attribute, the value is ignored and an operator message is issued to indicate the problem. (The effect is the same as missing out the attribute entirely.)

When you create a new queue manager, you should:

- Back up the WebSphere MQ configuration file
- Back up the new queue manager configuration file

Comments can be included in configuration files by adding a semi colon (;) or a number sign (#) character before the comment text. If you want to use one of these characters without it representing a comment, you can prefix the character with a backslash (\) for it to be used as part of the configuration data.

When do you need to edit a configuration file?

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

- You lose a configuration file; recover from backup if possible.
- You need to move one or more queue managers to a new directory.
- You need to change your default queue manager; this could happen if you accidentally delete the existing queue manager.
- You are advised to do so by your IBM Support Center.

Configuration file priorities

The attribute values of a configuration file are set according to the following priorities:

- Parameters entered on the command line take precedence over values defined in the configuration files
- Values defined in the `qm.ini` files take precedence over values defined in the `mqs.ini` file.

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 WebSphere MQ configuration file are only implemented when WebSphere MQ 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.

The WebSphere MQ configuration file, `mqs.ini`

The WebSphere MQ configuration file, `mqs.ini`, contains information relevant to all the queue managers on the node. It is created automatically during installation. In particular, the `mqs.ini` file is used to locate the data associated with each queue manager.

The `mqs.ini` file is stored in the data directory by default, `MQS_ROOT:[MQM]`.

The `mqs.ini` file contains:

- The names of the queue managers
- The name of the default queue manager
- The location of the files associated with each of them.

For more information on `mqs.ini` contents, see “Attributes for changing WebSphere MQ configuration information.”

Queue manager configuration files, `qm.ini`

A queue manager configuration file, `qm.ini`, contains information relevant to a specific queue manager. There is one queue manager configuration file for each queue manager. The `qm.ini` file is automatically created when the queue manager with which it is associated is created.

A `qm.ini` file is held in the root of the directory tree occupied by the queue manager.

For example, in WebSphere MQ for HP OpenVMS, the path and the 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 WebSphere MQ file names” on page 17.

For more information about `qm.ini`, see “Changing queue manager configuration information” on page 79.

Attributes for changing WebSphere MQ configuration information

The attributes described here modify the configuration of WebSphere MQ by editing the `mqs.ini` configuration file.

The following are detailed:

- “AllQueueManagers stanza”
- “ClientExitPath stanza” on page 75
- “DefaultQueueManager stanza” on page 75
- “ExitProperties stanza” on page 75
- “The LogDefaults stanza” on page 76
- “QueueManagers stanza” on page 78

A sample `mqs.ini` is shown in “API exits” on page 86.

AllQueueManagers stanza

The AllQueueManagers stanza can specify:

- The path to the `qmgrs` directory where the files associated with a queue manager are stored
- The method for converting EBCDIC-format data to ASCII format

DefaultPrefix=*directory_name*

This attribute specifies the path to the `qmgrs` directory, within which the queue manager data is kept.

If you change the default prefix for the queue manager, replicate the directory structure that was created at installation time (see Figure 22 on page 276).

In particular, you must create the `qmgrs` structure. Stop WebSphere MQ before changing the default prefix, and restart WebSphere MQ only after you have moved the structures to the new location and changed the default prefix.

As an alternative to changing the default prefix, you can use the environment variable `MQSPREFIX` to override the `DefaultPrefix` for the `crtmqm` command.

Because of operating system restrictions, keep the supplied path sufficiently short so that the sum of the path length and any queue manager name is a maximum of 70 characters long.

ConvEBCDICNewline=NL_TO_LF|TABLE|ISO

EBCDIC code pages contain a new line (NL) character that is not supported by ASCII code pages (although some ISO variants of ASCII contain an equivalent).

Use the `ConvEBCDICNewline` attribute to specify how WebSphere MQ is to convert the EBCDIC NL character into ASCII format.

NL_TO_LF

Convert the EBCDIC NL character (X'15') to the ASCII line feed character, LF (X'0A'), for all EBCDIC to ASCII conversions.

`NL_TO_LF` is the default.

TABLE

Convert the EBCDIC NL character according to the conversion tables used on your platform for all EBCDIC to ASCII conversions.

The effect of this type of conversion might vary from platform to platform and from language to language; even on the same platform, the behavior might vary if you use different CCSIDs.

ISO

Convert:

- ISO CCSIDs using the `TABLE` method
- All other CCSIDs using the `NL_TO_CF` method

Possible ISO CCSIDs are shown in Table 2.

Table 2. List of possible ISO CCSIDs

CCSID	Code Set
819	ISO8859-1
912	ISO8859-2
915	ISO8859-5
1089	ISO8859-6
813	ISO8859-7
916	ISO8859-8
920	ISO8859-9
1051	roman8

If the ASCII CCSID is not an ISO subset, `ConvEBCDICNewline` defaults to `NL_TO_LF`.

For more information about data conversion, see the *WebSphere MQ Application Programming Guide* or “Data conversion” on page 69.

ClientExitPath stanza

The `ClientExitPath` stanza specifies the default path for location of channel exit on the client.

ExitsDefaultPath=*defaultprefix*

The default prefix for the platform, for the location of 32-bit channel exits.

ExitsDefaultPath64=*defaultprefix*

The default prefix for the platform, for the location of 64-bit channel exits.

DefaultQueueManager stanza

The `DefaultQueueManager` stanza specifies the default queue manager for the node.

Name=*default_queue_manager*

The default queue manager processes any commands for which a queue manager name is not explicitly specified. The `DefaultQueueManager` attribute is automatically updated if you create a new default queue manager. If you inadvertently create a new default queue manager and then want to revert to the original, alter the `DefaultQueueManager` attribute manually.

ExitProperties stanza

The `ExitProperties` stanza specifies configuration options used by queue manager exit programs.

CLWLMode=SAFE | FAST

The cluster workload exit, CLWL, allows you to specify which cluster queue in the cluster to open in response to an MQI call (**MQOPEN**, **MQPUT**, and so on). The CLWL exit runs either in FAST mode or SAFE mode depending on the value you specify on the `CLWLMode` attribute. If you omit the `CLWLMode` attribute, the cluster workload exit runs in SAFE mode.

SAFE

Run the CLWL exit in a separate process from the queue manager. This is the default.

If a problem arises with the user-written CLWL exit when running in SAFE mode, the following happens:

- The CLWL server process (`amqzlw0`) fails.
- The queue manager restarts the CLWL server process.
- The error is reported to you in the error log. If an MQI call is in progress, you receive notification in the form of a return code.

The integrity of the queue manager is preserved.

Note: Running the CLWL exit in a separate process can affect performance.

FAST

Run the cluster exit inline in the queue manager process.

Specifying this option improves performance by avoiding the overheads associated with running in SAFE mode, but does so at the expense of queue manager integrity. You should only run the CLWL exit in FAST

mode if you are convinced that there are **no** problems with your CLWL exit, and you are particularly concerned about performance.

If a problem arises when the CLWL exit is running in FAST mode, the queue manager fails and you run the risk of the integrity of the queue manager being compromised.

The LogDefaults stanza

The LogDefaults stanza specifies the default log attributes for the node. The log attributes are used as default values when you create a queue manager, but can be overridden if you specify the log attributes on the **crtmqm** command. See “crtmqm (create queue manager)” on page 205 for details of this command.

Once a queue manager has been created, the log attributes for that queue manager are read from its log stanza in the `qm.ini` file.

The *DefaultPrefix* attribute (in the AllQueueManagers stanza) and the *LogPath* attribute in the LogDefaults stanza allow for the queue manager and its log to be on different physical drives. This is the recommended method, although, by default, they are on the same drive.

For information about calculating log sizes, see “Calculating the size of the log” on page 162.

Note: The limits given in the following parameter list are limits set by WebSphere MQ. Operating system limits may reduce the maximum possible log size.

LogPrimaryFiles=3 | 2-62

Primary log files are the log files allocated during creation for future use.

The minimum number of primary log files you can have is 2 and the maximum is 62. The default is 3.

The total number of primary and secondary log files must not exceed 63, and must not be less than 3.

This value is overwritten by the `-lp` parameter of the **crtmqm** command when the queue manager is created.

LogSecondaryFiles=2 | 1-61

Secondary log files are the log files allocated when the primary files are exhausted.

The minimum number of secondary log files is 1 and the maximum is 61. The default number is 2.

The total number of primary and secondary log files must not exceed 63, and must not be less than 3.

This value is overwritten by the `-ls` parameter of the **crtmqm** command when the queue manager is created.

LogFilePages=number

The log data is held in a series of files called log files. The log file size is specified in units of 4 KB pages.

For WebSphere MQ for HP OpenVMS, the default number of log file pages is 1024, giving a log file size of 4 MB. The minimum number of log file pages is 64 and the maximum is 16 384.

This value is overwritten by the `-lf` parameter of the `crtmqm` command when the queue manager is created.

LogType=CIRCULAR | LINEAR

The `LogType` attribute is used to define the type to be used. The default is `CIRCULAR`.

CIRCULAR

Set this value if you want to start restart recovery using the log to roll back transactions that were in progress when the system stopped.

See “Circular logging” on page 158 for a fuller explanation of circular logging.

LINEAR

Set this value if you want both restart recovery and media or forward recovery (creating lost or damaged data by replaying the contents of the log).

See “Linear logging” on page 159 for a fuller explanation of linear logging.

If you want to change the default logtype, you can edit the `LogType` attribute in the `mqs.ini` file. Alternatively, you can override the default by specifying linear logging using the `-ll` parameter on the `crtmqm` command. You cannot change the logging method after a queue manager has been created.

LogBufferPages=17 | 4-32

The amount of memory allocated to buffer records for writing is configurable. The size of the buffers is specified in units of 4 KB pages.

The minimum number of buffer pages is 4 and the maximum is 32. Larger buffers lead to higher throughput, especially for larger messages.

The default number of buffer pages is 17, equating to 68 KB.

The value is examined when the queue manager is created or started, and may be increased or decreased at either of these times. However, a change in the value is not effective until the queue manager is restarted.

LogDefaultPath=directory_name

You can specify the directory in which the log files for a queue manager reside. The directory should exist on a local device to which the queue manager can write and, preferably, should be on a different drive from the message queues. Specifying a different drive gives added protection in case of system failure.

The default for WebSphere MQ for HP OpenVMS is `MQS_ROOT:[MQM.LOG]`.

Alternatively, you can specify the name of a directory on the `crtmqm` command using the `-ld` flag. When a queue manager is created, a directory is also created under the queue manager directory, and this is used to hold the log files. The name of this directory is based on the queue manager name. This ensures that the Log File Path is unique, and also that it conforms to any limitations on directory name lengths.

If you do not specify `-ld` on the `crtmqm` command, the value of the `LogDefaultPath` attribute in the `mqs.ini` file is used by default and this is `MQS_ROOT:[MQM.LOG]`.

The queue manager name is appended to the log file 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 qm.ini file giving the complete directory name for the queue manager's log files. This value is used to locate the log files when the queue manager is started or deleted.

LogWriteIntegrity=SingleWrite | DoubleWrite | TripleWrite

The LogWriteIntegrity attribute is used to reliably write log records. The default is TripleWrite .

SingleWrite

Some hardware guarantees that, if a write operation writes a page and fails for any reason, a subsequent read of the same page into a buffer results in each byte in the buffer being either:

- The same as before the write, or
- The byte that should have been written in the write operation

On this type of hardware (for example, ssa write cache enabled), it is safe for the logger to write log records in a single write as the hardware assures full write integrity. This method provides the highest level of performance.

DoubleWrite

The DoubleWrite method was the default method used in WebSphere MQ V5.2 and is available for back-compatibility purposes only.

TripleWrite

TripleWrite is the default method. Where hardware that assures write integrity is not available, write log records using the TripleWrite method because it provides full write integrity.

If you want to change the default logtype, you can edit the LogType attribute in the mqz.ini file. Alternatively, you can override the default by specifying linear logging using the -ll parameter on the **crtmqm** command. You cannot change the logging method after a queue manager has been created.

API exits

Use the ApiExitTemplate and ApiExitCommon stanza in the mqz.ini file to identify API exit routines for all queue managers. (To identify API exit routines for individual queue managers, you use the ApiExitLocal stanza in the qm.ini file.

QueueManagers stanza

There is one QueueManager stanza for every queue manager. These attributes specify the queue manager name, and the name of the directory containing the files associated with that queue manager. The name of the directory is based on the queue manager name, but is transformed if the queue manager name is not a valid file name. See "Understanding WebSphere MQ file names" on page 17 for more information about name transformation.

Name=*queue_manager_name*

The name of the queue manager.

Prefix=*prefix*

Where the queue manager files are stored. By default, this is the same as the value specified on the DefaultPrefix attribute of the All Queue Managers information.

Directory=*name*

The name of the subdirectory under the <prefix>\QMGRS directory where the

queue manager files are stored. This name is based on the queue manager name, but can be transformed if there is a duplicate name or if the queue manager name is not a valid file name.

Changing queue manager configuration information

The following groups of attributes can appear in a `qm.ini` file particular to a given queue manager, or be used to override values set in `mqm.ini`.

- “The Service stanza”
- “The ServiceComponent stanza” on page 80
- “The Log stanza” on page 80
- “The XAResourceManager stanza” on page 82
- “The Channels stanza” on page 83
- “The LU62 and TCP stanzas” on page 85
- “The ExitPath stanza” on page 86

The Service stanza

The Service stanza specifies the name of an installable service, and the number of entry points to that service. There must be one Service stanza for every service used.

For each component within a service, there must be a ServiceComponent stanza, which identifies the name and path of the module containing the code for that component. See “The ServiceComponent stanza” on page 80 for more information.

Name=AuthorizationService | NameService

Specifies the name of the required service.

AuthorizationService

For WebSphere MQ, the Authorization Service component is known as the Object Authority Manager, or OAM.

In WebSphere MQ for HP OpenVMS, the AuthorizationService stanza and its associated ServiceComponent stanza are added automatically when the queue manager is created, but can be overridden through the use of `mqmnoaut`, by setting the `mqmnoaut` logical before creating the queue manager. (See Chapter 7, “WebSphere MQ security,” on page 91 for more information). Any other ServiceComponent stanzas must be added manually.

NameService

The NameService stanza must be added to the `qm.ini` file manually to enable the supplied name service.

EntryPoints=number-of-entries

Specifies the number of entry points defined for the service. This includes the initialization and termination entry points.

For more information about installable services and components, see the *WebSphere MQ Programmable Command Formats and Administration Interface* manual .

For more information about security services in general, see Chapter 7, “WebSphere MQ security,” on page 91.

The ServiceComponent stanza

The ServiceComponent stanza identifies the name and path of the module containing the code for that component.

There can be more than one ServiceComponent stanza for each service, but each ServiceComponent stanza must match the corresponding Service stanza.

In WebSphere MQ for HP OpenVMS, the authorization service stanza is present by default, and the associated component, the OAM, is active.

Service=*service_name*

Specifies the name of the required service. This name must match the value specified on the Name attribute of the Service stanza.

Name=*component_name*

Specifies the descriptive name of the service component. This name must be unique, and must contain only those characters that are valid for the names of WebSphere MQ objects (for example, queue names). This name occurs in operator messages generated by the service. Begin the name with a company trademark or similar distinguishing string.

Module=*module_name*

Specifies the name of the module to contain the code for this component.

Note: Specify a full path name.

ComponentDataSize=*size*

Specifies the size, in bytes, of the component data area passed to the component on each call. Specify zero if no component data is required.

For more information about installable services and components, see the *WebSphere MQ Programmable Command Formats and Administration Interface* manual.

The Log stanza

The Log stanza specifies the log attributes for a particular queue manager. By default, these are inherited from the settings specified in the LogDefaults stanza in the `mq5.ini` file when the queue manager is created, unless overridden by specific parameters in the `crtmqm` command. For more information, see both “The LogDefaults stanza” on page 76 and “`crtmqm` (create queue manager)” on page 205.

Change attributes of this stanza only if this particular queue manager needs to be configured differently from your other ones.

The values specified on the attributes in the `qm.ini` file are read when the queue manager is started. The file is created when the queue manager is created.

For information about calculating log sizes, see “Calculating the size of the log” on page 162.

Note: The limits given in the following parameter list are limits set by WebSphere MQ. Operating system limits may reduce the maximum possible log size.

LogPrimaryFiles=3 | 2-62

Primary log files are the log files allocated during creation for future use.

The minimum number of primary log files you can have is 2 and the maximum is 62. The default is 3.

The total number of primary and secondary log files must not exceed 63, and must not be less than 3.

The value is examined when the queue manager is created or started. You can change it after the queue manager has been created. However, a change in the value is not effective until the queue manager is restarted, and the effect may not be immediate.

LogSecondaryFiles=2 | 1-61

Secondary log files are the log files allocated when the primary files are exhausted.

The minimum number of secondary log files is 1 and the maximum is 61. The default number is 2.

The total number of primary and secondary log files must not exceed 63, and must not be less than 3.

The value is examined when the queue manager is started. You can change this value, but changes do not become effective until the queue manager is restarted, and even then the effect may not be immediate.

LogFilePages=number

The log data is held in a series of files called log files. The log file size is specified in units of 4 KB pages.

In WebSphere MQ for HP OpenVMS, the default number of log file pages is 1024, giving a log file size of 4 MB. The minimum number of log file pages is 64 and the maximum is 16 384.

Note: The size of the log files specified during queue manager creation cannot be changed for an existing queue manager.

LogType=CIRCULAR | LINEAR

The LogType attribute defines the type of logging to be used by the queue manager. However, you cannot change the type of logging to be used once the queue manager has been created. Refer to the description of the LogType attribute in “The LogDefaults stanza” on page 76 for information about creating a queue manager with the type of logging you require.

CIRCULAR

Set this value if you want to start restart recovery using the log to roll back transactions that were in progress when the system stopped.

See “Circular logging” on page 158 for a fuller explanation of circular logging.

LINEAR

Set this value if you want both restart recovery and media or forward recovery (creating lost or damaged data by replaying the contents of the log).

See “Linear logging” on page 159 for a fuller explanation of linear logging.

LogBufferPages=17 | 4-32

The amount of memory allocated to buffer records for writing is configurable. The size of the buffers is specified in units of 4 KB pages.

The minimum number of buffer pages is 4 and the maximum is 32. Larger buffers lead to higher throughput, especially for larger messages.

The default number of buffer pages is 17, equating to 68 KB.

The value is examined when the queue manager is started, and may be increased or decreased at either of these times. However, a change in the value is not effective until the queue manager is restarted.

LogPath=*directory_name*

You can specify the directory in which the log files for a queue manager reside. The directory should exist on a local device to which the queue manager can write and, preferably, should be on a different drive from the message queues. Specifying a different drive gives added protection in case of system failure.

The default is MQS_ROOT: [MQM.LOG].

You can specify the name of a directory on the **crtmqm** command using the **-ld** flag. When a queue manager is created, a directory is also created under the queue manager directory, and this is used to hold the log files. The name of this directory is based on the queue manager name. This ensures that the log file path is unique, and also that it conforms to any limitations on directory name lengths.

If you do not specify **-ld** on the **crtmqm** command, the value of the **LogDefaultPath** attribute in the **mqm.ini** file is used.

Note: In WebSphere MQ for HP OpenVMS, user ID **mqm** and group **mqm** must have full authorities to the log files. If you change the locations of these files, you must give these authorities yourself. This is not required if the log files are in the default locations supplied with the product.

The XAResourceManager stanza

The XAResourceManager stanza specifies the resource managers to be involved in global units of work coordinated by the queue manager.

One XAResourceManager stanza is required in **qm.ini** for each instance of a resource manager participating in global units of work; no default values are supplied through **mqm.ini**.

See “Database coordination” on page 112 for more information about adding XAResourceManager attributes to **qm.ini**.

Name=*name* (mandatory)

This attribute identifies the resource manager instance.

The Name value can be up to 31 characters in length and must be unique within **qm.ini**. You can use the name of the resource manager as defined in its XA-switch structure. However, if you are using more than one instance of the same resource manager, you must construct a unique name for each instance. You could ensure uniqueness by including the name of the database in the Name string, for example.

WebSphere MQ uses the Name value in messages and in output from the **dspmqrn** command.

Do not change the name of a resource manager instance, or delete its entry from **qm.ini** once the associated queue manager has started and the resource-manager name is in effect.

SwitchFile=*name* (mandatory)

This attribute specifies the fully-qualified name of the load file containing the resource manager’s XA switch structure.

XAOpenString=string (optional)

This attribute specifies the string of data to be passed to the resource manager's xa_open entry point. The contents of the string depend on the resource manager itself. For example, the string could identify the database that this instance of the resource manager is to access. For more information about defining this attribute, see "Adding XAResourceManager configuration information for Oracle" on page 116 and consult your resource manager documentation for the appropriate string.

XACloseString=string (optional)

This attribute specifies the string of data to be passed to the resource manager's xa_close entry point. The contents of the string depend on the resource manager itself. For more information about defining this attribute, see "Adding XAResourceManager configuration information for Oracle" on page 116 and consult your database documentation for the appropriate string.

ThreadOfControl=THREAD | PROCESS

The value set on the ThreadOfControl attribute is used by the queue manager for serialization purposes when it needs to call the resource manager from one of its own multithreaded processes.

THREAD

Means that the resource manager is fully "thread aware". In a multithreaded WebSphere MQ process, XA function calls can be made to the external resource manager from multiple threads at the same time.

PROCESS

Means that the resource manager is not "thread safe". In a multithreaded WebSphere MQ process, only one XA function call at a time can be made to the resource manager.

The ThreadOfControl entry does not apply to XA function calls issued by the queue manager in a multithreaded application process. In general, an application that has concurrent units of work on different threads requires this mode of operation to be supported by each of the resource managers.

The Channels stanza

The Channels stanza contains information about the channels.

MaxChannels=100 | number

This attribute specifies the maximum number of channels allowed. The default is 100.

MaxActiveChannels=MaxChannels_value

This attribute specifies the maximum number of channels allowed to be active at any time. The default is the value specified on the MaxChannels attribute.

MaxInitiators=3 | number

This attribute specifies the maximum number of initiators.

MQIBINDTYPE=FASTPATH | STANDARD

This attribute specifies the binding for applications.

FASTPATH

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

STANDARD

Channels connect using STANDARD.

AdoptNewMCA=NO|SVR|SDR|RCVR|CLUSRCVR|ALL|FASTPATH

If WebSphere MQ receives a request to start a channel but finds that an amqcrsta process already exists for the same channel, the existing process must be stopped before the new one can start. The AdoptNewMCA attribute allows you to control the termination of an existing process and the startup of a new one for a specified channel type.

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

1. The new channel tries to stop the previous one by inviting it to end.
2. If the previous channel server does not respond to this invitation by the time the AdoptNewMCATimeout wait interval expires, the process (or the thread) for the previous channel server is terminated.
3. If the previous channel server has not ended after step 2, and after the AdoptNewMCATimeout wait interval expires for a second time, WebSphere MQ ends the channel with a "CHANNEL IN USE" error.

You specify one or more values, separated by commas or blanks, from the following list:

NO

The AdoptNewMCA feature is not required. This is the default.

SVR

Adopt server channels

SDR

Adopt sender channels

RCVR

Adopt receiver channels

CLUSRCVR

Adopt cluster receiver channels

ALL

Adopt all channel types, except for FASTPATH channels

FASTPATH

Adopt the channel if it is a FASTPATH channel. This happens only if the appropriate channel type is also specified, for example, AdoptNewMCA=RCVR,SVR,FASTPATH

Attention: The AdoptNewMCA attribute may behave in an unpredictable fashion with FASTPATH channels because of the internal design of the queue manager. Therefore, be very careful when enabling the AdoptNewMCA attribute for FASTPATH channels.

AdoptNewMCATimeout=60|1—3600

This attribute specifies the amount of time, in seconds, that the new process should wait for the old process to end. Specify a value, in seconds, in the range 1—3600. The default value is 60.

AdoptNewMCACheck=QM|ADDRESS|NAME|ALL

The AdoptNewMCACheck attribute allows you to specify the type checking required when enabling the AdoptNewMCA attribute. It is important for you to perform all three of the following checks, if possible, to protect your channels from being, inadvertently or maliciously, shut down. At the very least check that the channel names match.

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

QM

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

ADDRESS

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

NAME

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

ALL

This means that you want the listener process to check for matching queue manager names, the communications address, and for matching channel names.

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

The LU62 and TCP stanzas

These stanzas specify network protocol configuration parameters. They override the default attributes for channels.

Note: Only attributes representing changes to the default values need to be specified.

LU62

The following attributes can be specified:

TPName

This attribute specifies the TP name to start on the remote site.

LocalLU

This is the name of the logical unit to use on local systems.

TCP

The following attributes can be specified:

Port=1414 | port_number

This attribute specifies the default port number, in decimal notation, for TCP/IP sessions. The port number for WebSphere MQ is 1414.

KeepAlive=YES | NO

Use this attribute to switch the KeepAlive function on or off. KeepAlive=YES causes TCP/IP to check periodically that the other end of the connection is still available. If it is not, the channel is closed.

ListenerBacklog=number

When receiving on TCP/IP, a maximum number of outstanding connection requests is set. This can be considered to be a *backlog* of requests waiting on the TCP/IP port for the listener to accept the request. The default listener backlog values are shown in Table 3.

Table 3. Default outstanding connection requests (TCP)

Platform	Default ListenerBacklog value
OS/390®	255
OS/2® Warp	10
Windows NT® Server	100
Windows NT Workstation	5
AS/400	255
Sun Solaris	100

Table 3. Default outstanding connection requests (TCP) (continued)

Platform	Default ListenerBacklog value
HP-UX	20
AIX V4.2 or later	100
AIX V4.1 or earlier	10
All other platforms	5

If the backlog reaches the values shown in Table 3 on page 85, the TCP/IP connection is rejected and the channel cannot start.

For MCA channels, this results in the channel going into a RETRY state and retrying the connection at a later time.

For client connections, the client receives an MQRC_Q_MGR_NOT_AVAILABLE reason code from MQCONN and should retry the connection at a later time.

The ListenerBacklog attribute allows you to override the default number of outstanding requests for the TCP/IP listener.

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

The ExitPath stanza

ExitDefaultPath=string

The ExitDefaultPath attribute specifies the location of:

- Channel exits for clients
- Channel exits and data conversion exits for servers

ExitDefaultPath64=string

The ExitDefaultPath64 attribute specifies the location of:

- 64-bit channel exits for clients
- 64-bit channel exits and data conversion exits for servers
- Unqualified XA switch load files

The exit path is read from the ClientExitPath stanza in the mqsc.ini file for clients and from this (ExitPath) stanza for servers.

API exits

Use the ApiExitLocal stanza in the qm.ini file to identify API exit routines for a queue manager. (To identify API exit routines for all queue managers, you use the ApiExitCommon and ApiExitTemplate stanzas, in the mqsc.ini file.)

Queue manager error logs

Use the QMErrorLog stanza in the qm.ini file to tailor the operation and contents of queue manager error logs.

ErrorLogSize=maxsize

Specifies the size of the queue manager error log at which it is copied to the backup. maxsize must be between 32768 and 2147483648 bytes. If **ErrorLogSize** is not specified, the default value of 262144 bytes (256KB) is used.

ExcludeMessage=msgIds

Specifies messages that are not to be written to the queue manager error log. *msgIds* contain a comma separated list of message id's from the following:

- 7163 - Job started message (iSeries® only)
- 7234 - Number of messages loaded
- 9001 - Channel program ended normally
- 9002 - Channel program started
- 9202 - Remote host not available
- 9208 - Error on receive from host
- 9209 - Connection closed
- 9228 - Cannot start channel responder
- 9508 - Cannot connect to queue manager
- 9524 - Remote queue manager unavailable
- 9528 - User requested closure of channel
- 9558 - Remote Channel is not available
- 9999 - Channel program ended abnormally

SuppressMessage=msgIds

Specifies messages that are written to the queue manager error log once only in a specified time interval. The time interval is specified by **SuppressInterval**. *msgIds* contain a comma separated list of message id's from the following:

- 7163 - Job started message (iSeries only)
- 7234 - Number of messages loaded
- 9001 - Channel program ended normally
- 9002 - Channel program started
- 9202 - Remote host not available
- 9208 - Error on receive from host
- 9209 - Connection closed
- 9228 - Cannot start channel responder
- 9508 - Cannot connect to queue manager
- 9524 - Remote queue manager unavailable
- 9528 - User requested closure of channel
- 9558 - Remote Channel is not available
- 9999 - Channel program ended abnormally

If the same message id is specified in both **SuppressMessage** and **ExcludeMessage**, the message is excluded.

SuppressInterval=length

Specifies the time interval, in seconds, in which messages specified in **SuppressMessage** are written to the queue manager error log once only. *length* must be between 1 and 86400 seconds. If **SuppressInterval** is not specified, the default value of 30 seconds is used.

Example mqs.ini and qm.ini files

Figure 8 on page 88 shows an example of an *mqs.ini* file in WebSphere MQ for HP OpenVMS.

```

#####
** Module Name: mqs.ini                                **
** Type       : WebSphere MQ Configuration File       **
** Function    : Define WebSphere MQ resources for the node **
**                                                    **
#####
** Notes      :                                       **
** 1) This is an example WebSphere MQ configuration file **
**                                                    **
#####
AllQueueManagers:
  #####
  ** The path to the qmgrs directory, below which queue manager data **
  ** is stored                                                         **
  #####
  DefaultPrefix=mqs_root:[mqm]

ClientExitPath:
  ExitsDefaultPath=mqs_root:[mqm.exits]

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

DefaultQueueManager:
  Name=saturn.queue.manager

QueueManager:
  Name=pluto.queue.manager
  Prefix=mqs_root:[mqm]
  Directory=pluto$queue$manager

```

Figure 8. Example of a WebSphere MQ configuration file for WebSphere MQ for HP OpenVMS systems

Figure 9 on page 89 shows how groups of attributes might be arranged in a queue manager configuration file in WebSphere MQ for HP OpenVMS.

```

#####
#* Module Name: qm.ini                                     *#
#* Type      : WebSphere MQ queue manager configuration file   *#
# Function   : Define the configuration of a single queue manager *#
#*          *#
#####
#* Notes      :                                             *#
#* 1) This file defines the configuration of the queue manager *#
#*          *#
#####
ExitPath:
  ExitsDefaultPath=mqm_root:[mqm.exits]

Service:
  Name=AuthorizationService
  EntryPoints=9

ServiceComponent:
  Service=AuthorizationService
  Name=WebSphere MQ.UNIX.auth.service
  Module=amqzfu
  ComponentDataSize=0

Service:
  Name=NameService
  EntryPoints=5

ServiceComponent:
  Service=NameService
  Name=WebSphere MQ.DCE.name.service
  Module=amqzfa
  ComponentDataSize=0

Log:
  LogPrimaryFiles=3
  LogSecondaryFiles=2
  LogFilePages=1024
  LogType=CIRCULAR
  LogBufferPages=17
  LogPath=mqm_root:[mqm.log.saturn$queue$manager]

XAResourceManager:
  Name=Oracle Resource Manager Bank
  SwitchFile=sys$share:oraswit0.exe
  XAOpenString=MQBankDB
  XACloseString=
  ThreadOfControl=PROCESS

CHANNELS:
  MaxChannels = 20      ; Maximum number of Channels allowed.
                       ; Default is 100.
  MaxActiveChannels = 10 ; Maximum number of Channels allowed to be
                       ; active at any time. The default is the
                       ; value of MaxChannels.

TCP:
  KeepAlive = Yes      ; TCP/IP entries.
                       ; Switch KeepAlive on

```

Figure 9. Example queue manager configuration file for WebSphere MQ for HP OpenVMS

Notes®:

WebSphere MQ 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 OpenVMS file system.

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

Chapter 7. WebSphere MQ security

This chapter describes the features of security control in WebSphere MQ for HP OpenVMS and how you can implement this control.

It contains these sections:

- “Why you need to protect WebSphere MQ resources”
- “Before you begin”
- “Understanding the Object Authority Manager” on page 92
- “Using the Object Authority Manager commands” on page 95
- “Object Authority Manager guidelines” on page 97
- “Understanding the authorization specification tables” on page 101
- “Working with the Secure Sockets Layer (SSL) on OpenVMS systems” on page 107

Why you need to protect WebSphere MQ resources

Because WebSphere MQ 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 WebSphere MQ objects such as queues, clusters, channels, and processes.
- Commands for queue manager administration, including MQSC commands and PCF commands.
- Access to WebSphere MQ messages.
- Context information associated with messages.

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

Before you begin

All queue manager resources run with the VMS Rights Identifier:

MQM

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

User IDs in WebSphere MQ for HP OpenVMS with resource identifier MQM

If your user ID holds the MQM OpenVMS rights identifier, you have all authorities to all WebSphere MQ resources. Your user ID **must** hold the OpenVMS MQM

rights identifier to be able to use all the WebSphere MQ for HP OpenVMS control commands except **crtmqcvx**. In particular, you need this authority to:

- Use the **runmqsc** command to run MQSC commands.
- Administer authorities on WebSphere MQ for HP 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 100.

In addition, installation of WebSphere MQ creates an identifier MQS_SERVER. This is granted ownership of the resource domain where VMS keeps lock information for WebSphere MQ. By default, access authorities to this identifier are granted to users who:

- Are in the same user group as the user MQM, or
- Are system users, or
- Have SYSPRV, SYSLCK, or BYPASS privilege set

To allow other users access to the MQ resources, you need to ensure that the MQS_SERVER identifier has appropriate WORLD privilege by running the command:

```
SET SECURITY/CLASS=RESOURCE [MQS_SERVER] /PROTECTION=(W:RWL)
```

Note: 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:

- WebSphere MQ for HP OpenVMS command sets, see Chapter 2, “An introduction to WebSphere MQ administration,” on page 15.
- WebSphere MQ for HP OpenVMS control commands, see Chapter 13, “How to use WebSphere MQ control commands,” on page 199.
- PCF commands and Escape PCFs, see the *WebSphere MQ Programmable Command Formats and Administration Interface* manual.
- MQI calls, see the *WebSphere MQ Application Programming Guide* and *WebSphere MQ Application Programming Reference* manual.

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 WebSphere MQ for HP OpenVMS. It is supplied with WebSphere MQ for HP 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 *WebSphere MQ Programmable Command Formats and Administration Interface* manual.

The OAM manages users' authorizations to manipulate WebSphere MQ 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 WebSphere MQ 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 WebSphere MQ commands. By default, no authorizations are given to this identifier. Users without specific authorizations can be granted access to WebSphere MQ resources through this rights identifier.

Resources you can protect with the OAM

Through OAM you can control:

- Access to WebSphere MQ 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 management 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)
- **dspmqaut** (Display authority)

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

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

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

What you specify when you use the OAM commands

The authority commands apply to the specified queue manager; if you do not specify a queue manager, the default queue manager is used. On these commands, you must specify the object uniquely, that is, you must specify the object name and its type. You also have to specify the principal or 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 plus (+) or minus (-) 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:

```
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 plus (+) or minus (-) 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 (grant or revoke authority)” on page 261.

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 (grant or revoke authority)” on page 261.

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 216 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:

- WebSphere MQ objects are not necessarily the same as the corresponding system object name. See “Understanding WebSphere MQ file names” on page 17 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 note 1 on page 104 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 a WebSphere MQ 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 WebSphere MQ 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 *WebSphere MQ Application Programming Guide*. For descriptions of the message descriptor fields relating to context, see the *WebSphere MQ Application Programming Reference* manual.

Remote security considerations

For remote security, you should consider:

Put authority

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

Specify the channel attribute PUTAUT as follows:

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

CTX The user ID in the message context.

Transmission queues

Queue managers automatically put remote messages on a transmission queue; no special authority is required for this. However, putting a message directly on a transmission queue requires special authorization; see Table 4 on page 102.

Channel exits

Channel exits can be used for added security.

For more information, see the *WebSphere MQ Intercommunication* book.

Channel command security

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

PCF commands

You can issue PCF channel commands by sending a PCF message to the SYSTEM.ADMIN.COMMAND.QUEUE on a remote 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*
- *StartChannelListener*
- *StopChannel*
- *ResolveChannel*

See the *WebSphere MQ Programmable Command Formats and Administration Interface* manual book 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
- START LISTENER
- STOP CHANNEL
- RESOLVE CHANNEL

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

Control commands for channels

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

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

Understanding the authorization specification tables

The authorization specification tables starting in topic Table 4 on page 102 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.

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 4 summarizes the authorizations needed for each call.

Table 4. Security authorization needed for MQI calls

Authorization required for:	Queue object (1 on page 103)	Process object	Queue manager object	Namelist
MQCONN option	Not applicable	Not applicable	MQZAO_CONNECT	Not applicable
MQOPEN Option				
MQOO_INQUIRE	MQZAO_INQUIRE (2 on page 104)	MQZAO_INQUIRE (2 on page 104)	MQZAO_INQUIRE (2 on page 104)	MQZAO_INQUIRE (2 on page 104)
MQOO_BROWSE	MQZAO_BROWSE	Not applicable	No check	Not applicable
MQOO_INPUT_*	MQZAO_INPUT	Not applicable	No check	Not applicable
MQOO_SAVE_ALL_CONTEXT (3 on page 104)	MQZAO_INPUT	Not applicable	No check	Not applicable
MQOO_OUTPUT (Normal queue) (4 on page 104)	MQZAO_OUTPUT	Not applicable	No check	Not applicable
MQOO_PASS_IDENTITY_CONTEXT (5 on page 104)	MQZAO_PASS_IDENTITY_CONTEXT	Not applicable	No check	Not applicable
MQOO_PASS_ALL_CONTEXT (5 on page 104, 6 on page 104)	MQZAO_PASS_ALL_CONTEXT	Not applicable	No check	Not applicable
MQOO_SET_IDENTITY_CONTEXT (5 on page 104, 6 on page 104)	MQZAO_SET_IDENTITY_CONTEXT	Not applicable	MQZAO_SET_IDENTITY_CONTEXT (7 on page 104)	Not applicable
MQOO_SET_ALL_CONTEXT (5 on page 104, 8 on page 104)	MQZAO_SET_ALL_CONTEXT	Not applicable	MQZAO_SET_ALL_CONTEXT (7 on page 104)	Not applicable

Table 4. Security authorization needed for MQI calls (continued)

Authorization required for:	Queue object (1)	Process object	Queue manager object	Namelists
MQOO_OUTPUT (Transmission queue) (9 on page 104)	MQZAO_SET_ALL_CONTEXT	Not applicable	MQZAO_SET_ALL_CONTEXT (7 on page 104)	Not applicable
MQOO_SET	MQZAO_SET	Not applicable	No check	Not applicable
MQOO_ALTERNATE_USER_AUTHORITY	(10 on page 104)	(10 on page 104)	MQZAO_ALTERNATE_USER_AUTHORITY (10 on page 104, 11 on page 104)	(10 on page 104)
MQPUT1 Option				
MQPMO_PASS_IDENTITY_CONTEXT	MQZAO_PASS_IDENTITY_CONTEXT (12 on page 104)	Not applicable	No check	Not applicable
MQPMO_PASS_ALL_CONTEXT	MQZAO_PASS_ALL_CONTEXT (12 on page 104)	Not applicable	No check	Not applicable
MQPMO_SET_IDENTITY_CONTEXT	MQZAO_SET_IDENTITY_CONTEXT (12 on page 104)	Not applicable	MQZAO_SET_IDENTITY_CONTEXT (7 on page 104)	Not applicable
MQPMO_SET_ALL_CONTEXT	MQZAO_SET_ALL_CONTEXT (12 on page 104)	Not applicable	MQZAO_SET_ALL_CONTEXT (7 on page 104)	Not applicable
(Transmission queue) (9 on page 104)	MQZAO_SET_ALL_CONTEXT	Not applicable	MQZAO_SET_ALL_CONTEXT (7 on page 104)	Not applicable
MQPMO_ALTERNATE_USER_AUTHORITY	(13 on page 104)	Not applicable	MQZAO_ALTERNATE_USER_AUTHORITY (11 on page 104)	Not applicable
MQCLOSE Option				
MQCO_DELETE	MQZAO_DELETE (14 on page 104)	Not applicable	Not applicable	Not applicable
MQCO_DELETE_PURGE	MQZAO_DELETE (14 on page 104)	Not applicable	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, namelist or queue manager object is checked, depending on the type of object being opened.
 3. MQOO_INPUT_* must also be specified. This is valid for a local, model, or alias queue.
 4. This check is performed for all output cases, except the case specified in note 9.
 5. MQOO_OUTPUT must also be specified.
 6. MQOO_PASS_IDENTITY_CONTEXT is also implied by this option.
 7. This authority is required for both the queue manager object and the particular queue.
 8. MQOO_PASS_IDENTITY_CONTEXT, MQOO_PASS_ALL_CONTEXT, and MQOO_SET_IDENTITY_CONTEXT are also implied by this option.
 9. This check is performed for a local or model queue that has a *Usage* queue attribute of MQUS_TRANSMISSION, and is being opened directly for output. It does not apply if a remote queue is being opened (either by specifying the names of the remote queue manager and remote queue, or by specifying the name of a local definition of the remote queue).
 10. At least one of MQOO_INQUIRE (for any object type), or (for queues) MQOO_BROWSE, MQOO_INPUT_*, MQOO_OUTPUT, or MQOO_SET must also be specified. The check carried out is as for the other options specified, using the supplied alternate user identifier for the specific-named object authority, and the current application authority for the MQZAO_ALTERNATE_USER_IDENTIFIER check.
 11. This authorization allows any *AlternateUserId* to be specified.
 12. An MQZAO_OUTPUT check is also carried out, if the queue does not have a *Usage* queue attribute of MQUS_TRANSMISSION.
 13. The check carried out is as for the other options specified, using the supplied alternate user identifier for the specific-named queue authority, and the current application authority for the MQZAO_ALTERNATE_USER_IDENTIFIER check.
 14. The check is carried out only if both of the following are true:
 - A permanent dynamic queue is being closed and deleted.
 - The queue was not created by the MQOPEN which returned the object handle being used.

Otherwise, there is no check.

General notes:

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

- MQZAO_PASS_ALL_CONTEXT
 - MQZAO_SET_IDENTITY_CONTEXT
 - MQZAO_SET_ALL_CONTEXT
 - MQZAO_ALTERNATE_USER_AUTHORITY
2. MQZAO_DELETE (see note 14 on page 104) and MQZAO_DISPLAY are classed as administration authorizations. They are not therefore included in MQZAO_ALL_MQI.
 3. 'No check' means that no authorization checking is carried out.
 4. 'Not applicable' means that authorization checking is not relevant to this operation. For example, you cannot issue an MQPUT call to a process object.

Administration authorizations

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

Authorizations for MQSC commands in escape PCFs

Table 5 summarizes the authorizations needed for each MQSC command that is contained in Escape PCF.

Table 5. MQSC commands and security authorization needed

Authorization required for: (2)	Queue object	Process object	Queue manager object	Namelists
MQSC command				
ALTER object	MQZAO_CHANGE	MQZAO_CHANGE	MQZAO_CHANGE	MQZAO_CHANGE
CLEAR QLOCAL	MQZAO_CLEAR	Not applicable	Not applicable	Not applicable
DEFINE object NOREPLACE (3)	MQZAO_CREATE (4 on page 106)	MQZAO_CREATE (4 on page 106)	Not applicable	MQZAO_CREATE (4 on page 106)
DEFINE object REPLACE (3, 5 on page 106)	MQZAO_CHANGE	MQZAO_CHANGE	Not applicable	MQZAO_CHANGE
DELETE object	MQZAO_DELETE	MQZAO_DELETE	Not applicable	MQZAO_DELETE
DISPLAY object	MQZAO_DISPLAY	MQZAO_DISPLAY	MQZAO_DISPLAY	MQZAO_DISPLAY

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, namelist or queue manager object is checked, depending on the type of object.
3. For DEFINE commands, MQZAO_DISPLAY authority is also needed for the LIKE object if one is specified, or on the appropriate SYSTEM.DEFAULT.xxx object if LIKE is omitted.

4. The MQZAO_CREATE authority is not specific to a particular object or object type. Create authority is granted for all objects, for a specified queue manager, by specifying an object type of QMGR on the SETMQAUT command.
5. This applies if the object to be replaced does in fact already exist. If it does not, the check is as for DEFINE object NOREPLACE.

General notes:

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

Authorizations for PCF commands

Table 6 summarizes the authorizations needed for each PCF command.

Table 6. PCF commands and security authorization needed

Authorization required for: (2)	Queue object	Process object	Queue manager object	Namelists
PCF command				
Change object	MQZAO_CHANGE	MQZAO_CHANGE	MQZAO_CHANGE	MQZAO_CHANGE
Clear Queue	MQZAO_CLEAR	Not applicable	Not applicable	Not applicable
Copy object (without replace) (3)	MQZAO_CREATE (4 on page 107)	MQZAO_CREATE (4 on page 107)	Not applicable	MQZAO_CREATE (4 on page 107)
Copy object (with replace) (3, 6 on page 107)	MQZAO_CHANGE	MQZAO_CHANGE	Not applicable	MQZAO_CHANGE
Create object (without replace) (5 on page 107)	MQZAO_CREATE (4 on page 107)	MQZAO_CREATE (4 on page 107)	Not applicable	MQZAO_CREATE (4 on page 107)
Create object (with replace) (5 on page 107, 6 on page 107)	MQZAO_CHANGE	MQZAO_CHANGE	Not applicable	MQZAO_CHANGE
Delete object	MQZAO_DELETE	MQZAO_DELETE	Not applicable	MQZAO_DELETE
Inquire object	MQZAO_DISPLAY	MQZAO_DISPLAY	MQZAO_DISPLAY	MQZAO_DISPLAY
Inquire object names	No check	No check	No check	No check
Reset queue statistics	MQZAO_DISPLAY and MQZAO_CHANGE	Not applicable	Not applicable	Not applicable

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, namelist or queue-manager object is checked, depending on the type of object.
3. For Copy commands, MQZAO_DISPLAY authority is also needed for the From object.

4. The MQZAO_CREATE authority is not specific to a particular object or object type. Create authority is granted for all objects, for a specified queue manager, by specifying an object type of QMGR on the SETMQAUT command.
5. For Create commands, MQZAO_DISPLAY authority is also needed for the appropriate SYSTEM.DEFAULT.* object.
6. This applies if the object to be replaced already exists. If it does not, the check is as for Copy or Create without replace.

General notes:

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

Working with the Secure Sockets Layer (SSL) on OpenVMS systems

The Secure Sockets Layer (SSL) protocol provides channel security, with protection against eavesdropping, tampering, and impersonation.

- You cannot run SSL channels from a WebSphere MQ installation that uses DCE security exits or the DCE name service.
- Support for cryptographic hardware is not available for WebSphere MQ for HP OpenVMS, Version 6.0.

On OpenVMS systems, to manage keys and digital certificates, HP SSL for OpenVMS provides a certificate tool that is a simple menu-driven interface for viewing and creating SSL certificates. The HP SSL Certificate Tool enables you to perform the most important certification functions. Using it, you can view certificates and certificate requests, create certificate requests, sign your own certificate, create your own certificate authority, and sign client certificate requests.

HP-SSL for OpenVMS does not provide any repository of digital certificates. Each queue manager has its own certificate. The certificate can be held in the SSL directory for the queue manager or WebSphere MQ client.

OpenSSL setup for WebSphere MQ

Certificate Handling

Personal Certificates:

You can use the SSL certificate tool for creating a self-signed personal certificate. This generates a certificate file with the extension .CRT and a private key file with

the extension .KEY. The locations on OpenVMS of these files default to SSL\$KEY:CERT.KEY and SSL\$KEY:CERT.CRT. WebSphere MQ for HP OpenVMS, V6.0 accepts the certificates in PEM format only. For generating the certificate in PEM format, follow these steps:

1. Enter the command \$ COPY CERT.KEY CERT.PEM
2. Enter the command \$ APPEND CERT.CRT CERT.PEM

The PEM extension is mandatory for the certificate file. The private key is a part of the digital certificate, which is in the PEM format. Normally the private key is encrypted and accessed by a password. The SSL server needs to add the private key found in the file to SSL_CTX structure. However, to access this, a password is required. A utility (cryptpasswd) is provided along with the server, which encrypts the password and puts it in a file.

```
$ CRYPTPASSWD <password> <name of the certificate file>
```

For example:

```
$ CRYPTPASSWD mypassword cert
```

This generates a file called CERT.PWD in the current directory. Place the generated PWD file in the same directory as the certificate file.

Certificate Authorities (CA) Certificate:

For generating the list of Certificate Authorities, create a file called CACert.PEM. This needs to be populated with the list of all the CA certificates. Make sure that the certificates are in PEM format. This file should be placed in the same directory as the certificate file.

Note: It is mandatory to name the file CACert.PEM.

Certificate Revocation List (CRL) certificates:

If you want to use certificate revocation lists (CRLs) on the system, you must put the CRLs in the same file that contains the trusted CA certificates (CACert.PEM) or the directory that contains the trusted CA certificates. As part of the authentication process, the system examines the CRLs, if any exist, to determine if the certificate provided by the partner node has been revoked. The following shows the contents of a sample CRL file:

```
-----BEGIN X509 CRL-----
MIIBODCB4zANBgkqhkiG9w0BAQFADBgMQswCQYDVQQGEwJBVTEEMMAoGA1UECBMD
UUXEMRkwFwYDVQQKEExBNaW5jb20gUHR5LiBMdGQuMQswCQYDVQQLEwJDUzEhMBkG
A1UEAxMSU1NMZWV5IGR1bW8gc2VydMvYFw0wMTAxMTUxNjI2NTdaFw0wMTAyMTQx
NjI2NTdaMFwEgIBARcNOTUxMDA5MjMjA1WjASAgEDFw05NTEyMDEwMTAwMDBa
MBMCAhI0Fw0wMTAxMTUxNjE5NDdaMBMCAhI1Fw0wMTAxMTUxNjIzNDZaMA0GCsqG
SIb3DQEBAUAQEAHPjQ3M93Q0j8Ufi+jZM7Y78TfAzG4jJn/E6MYBPFVQFYo/Gp
UZexfjSVo5CIyyS0tYscz8o07avwBxTiMpDEQg==
-----END X509 CRL--
```

Working with Queue Manager and Client Certificates

Locating the certificate for a queue manager:

Use this procedure to obtain information about the location of your queue manager's certificate file:

1. Display your queue manager's attributes using either of the following MQSC commands:

```
DISPLAY QMGR ALL
```

DISPLAY QMGR SSLKEYR

2. Examine the output of the above command for the path and name of the certificate file. For example: /mqs_root/mqm/qmgrs/QM1/ssl/qmcert where /mqs_root/mqm/qmgrs/QM1/ssl is the default directory path and qmcert represents the name of the certificate file.

Note: The QMCERT.PWD and CACert.PEM files should be placed in the default path.

Changing the certificate location for a queue manager:

You can change the location of your queue manager's certificate file using the following method:

Use the ALTER QMGR MQSC command to set your queue manager's certificate for example:

```
ALTER QMGR SSLKEYR('/mqs_root/mqm/qmgrs/QM1/ssl/MyQMCert')
```

The fully qualified path name for the certificate file has to be specified for the SSLKEYR attribute in UNIX style. The actual location of the certificate on OpenVMS would be mqs_root:[mqm.qmgrs.QM1.ssl]MyQMCert.pem.

Note:

1. The .pem extension is a mandatory part of the filename, but is not included as part of the value of the parameter.
2. The MyQMCert.PWD and CACert.PEM files should be placed in the same directory path, which you have specified in ALTER QMGR SSLKEYR command.

Specifying the certificate location for a WebSphere MQ client:

You can specify the location of your WebSphere MQ client's certificate file in one of the following ways:

- By setting the MQSSLKEYR logical, for example:
\$ DEFINE/LOG MQSSLKEYR "/dka100/users/user1/ClientCert"

The fully qualified path name for the certificate file has to be specified for the above logical in UNIX style. The actual location of the certificate on OpenVMS would be dka100:[users.user1]ClientCert.pem.

Note:

1. The .pem extension is a mandatory part of the filename, but is not included as part of the value of the logical.
 2. The ClientCert.PWD and CACert.PEM files should be placed in the same directory path.
- By providing the path and name of the certificate file in the KeyRepository field of the MQSCO structure when an application makes an MQCONN call. For more information about using the MQSCO structure in MQCONN, refer to the *WebSphere MQ Application Programming Guide*.

The path to the CA certificates can be given by two logical names. These are CACertFile and CACertPath. For example:

```
$ DEFINE/LOG CACertFile "/path/to/CA/file"  
$ DEFINE/LOG CACertPath "/path/to/CA/directory"
```

HP SSL version

You must install HP Open VMS V1.3 for OpenVMS Alpha for SSL support. This is the supported version of SSL for WebSphere MQ for HP OpenVMS, V6.0. It can be downloaded from:

<http://h71000.www7.hp.com/openvms/products/ssl/ssl.html>

Chapter 8. Transactional support

The *WebSphere MQ Application Programming Guide* contains a complete introduction to the subject of this chapter. A brief introduction only is provided here.

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

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

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

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

In summary, queue manager resources can be updated as part of local or global units of work:

Local unit of work

Use local units of work when the only resources to be updated are those of the WebSphere MQ queue manager. Updates are committed using the MQCMIT verb or backed out using MQBACK.

Global unit of work

Use global units of work when you also need to include updates to XA-compliant database managers. Here, the coordination may be internal or external to the queue manager.

Queue manager coordination

Global units of work are started using the MQBEGIN verb and then committed using MQCMIT or backed out using MQBACK. A two-phase commit process is used whereby XA-compliant resource managers such as Oracle are all asked to prepare to commit. Only if all are prepared successfully are they then be asked to commit. If any resource manager signals that it cannot prepare to commit, it is asked to back out instead.

External coordination

Here, the coordination is performed by an XA-compliant transaction manager such as IBM CICS®, Transarc Encina®, or BEA Tuxedo. Units of work are started and committed under control of the transaction manager. The MQBEGIN, MQCMIT and MQBACK verbs are unavailable.

This chapter describes how to enable support for global units of work (support for local units of work does not need to be specifically enabled).

It contains these sections:

- “Database coordination”
- “Oracle configuration” on page 115
- “Administration tasks” on page 118

Database coordination

When the queue manager coordinates global units of work itself it becomes possible to integrate database updates within MQ units of work. That is, a mixed MQI and SQL application can be written, and the MQCMIT and MQBACK verbs can be used to commit or roll back the changes to the queues and databases together.

The queue manager achieves this using a two-phase commit protocol. When a unit of work is to be committed, the queue manager first asks each participating database manager whether it is prepared to commit its updates. Only if all of the participants, including the queue manager itself, are prepared to commit, are all of the queue and database updates committed. If any participant cannot prepare its updates, the unit of work is backed out instead.

Full recovery support is provided if the queue manager loses contact with any of the database managers during the commit protocol. If a database manager becomes unavailable while it is in doubt, that is, it has been called to prepare but has yet to receive a commit or back out decision, the queue manager remembers the outcome of the unit of work until it has been successfully delivered. Similarly, if the queue manager terminates with incomplete commit operations outstanding, these are remembered over queue manager restart.

The MQI verb, MQBEGIN, must be used to denote units of work that are also to involve database updates. The *WebSphere MQ Application Programming Guide* identifies sample programs that make WebSphere MQ and database updates within the same unit of work.

The queue manager communicates with the database managers using the XA interface as described in *X/Open Distributed Transaction Processing: The XA Specification* (ISBN 1 872630 24 3). This means that the queue manager can communicate to database managers that also adhere to this standard. Such database managers are known as *XA-compliant* database managers.

Restrictions

The following restrictions apply to the database coordination support:

- The ability to coordinate database updates within WebSphere MQ units of work is not supported in an MQI client application.
- The MQI updates and database updates must be made on the same queue manager server machine.
- The database server may reside on a different machine from the queue manager server. In this case, the database needs to be accessed via an XA-compliant client feature provided by the database manager itself.
- Although the queue manager itself is XA-compliant, it is not possible to configure another queue manager as a participant in global units of work. This is because only one connection at a time can be supported.

Database connections

An application that establishes a standard connection to the queue manager is associated with a thread in a separate local queue manager agent process. When the application issues MQBEGIN then both it and the agent process need to connect to the databases that are to be involved in the unit of work. The database connections are maintained while the application remains connected to the queue manager. This is an important consideration if the database supports only a limited number of users or connections.

One method of reducing the number of connections is for the application to use the MQCONNX call to request a fastpath binding. In this case the application and the local queue manager agent become the same process and consequently can share a single database connection. Before using MQCONNX, consult the *WebSphere MQ Application Programming Guide* for a list of restrictions that apply to fastpath applications.

Configuring database managers

There are two tasks that you must perform before a database manager can participate in global units of works coordinated by the queue manager:

1. Create an *XA switch load file*¹ for the database manager.
2. Define the database manager in the queue manager's configuration file, `qm.ini`. Various items, including the name of the switch load file, must be defined in `qm.ini`.

Creating switch load files

Instructions for creating switch load files for the supported database managers are provided in "Creating the Oracle switch load file on OpenVMS systems" on page 116.

Refer to your WebSphere MQ installation documentation for more information about the installation procedure.

The sample source modules that are used to produce the switch load files all contain a single function called MQStart. When the switch load file is loaded, the queue manager calls this function and it returns the address of a structure called an XA switch. The switch load file is linked to a library provided by the database manager, which enables WebSphere MQ to call that database manager.

The sample source modules used to build the switch load files for Oracle are `oraswit.c`.

Defining database managers

When you have created a switch load file for your database manager, you must specify its location to your queue manager. This is done in the queue manager's `qm.ini` file in the `XAResourceManager` stanza.

You need to add an `XAResourceManager` stanza for each database manager that your queue manager is going to coordinate.

1. An XA switch load file is a dynamically-loaded object that enables the queue manager and the database manager to communicate with each other.

The attributes of the XAResourceManager stanza are as follows.

Name=name

User-chosen string that identifies the database manager instance.

The name is mandatory and can be up to 31 characters in length. It must be unique. It could simply be the name of the database manager, although to maintain its uniqueness in more complicated configurations it could, for example, also include the name of the database being updated.

The name that you choose should be meaningful because the queue manager uses it to refer to this database manager instance both in messages and in output when the **dspmqrn** command is used.

Once you have chosen a name, do not change this attribute. Information about changing configuration information is given in “Changing configuration information” on page 122.

SwitchFile=name

This is the fully-qualified name of the database manager’s XA switch load file. This is a mandatory attribute.

XAOpenString=string

This is a string of data that is passed in calls to the database manager’s `xa_open` entry point. The format for this string depends on the particular database manager, but it should usually identify the name of the database that is to be updated.

This is an optional attribute; if it is omitted a blank string is assumed.

XACloseString=string

This is a string of data that is passed in calls to the database manager’s `xa_close` entry point. The format for this string depends on the particular database manager.

This is an optional attribute; if it is omitted a blank string is assumed.

ThreadOfControl=THREAD | PROCESS

The *ThreadOfControl* value can be `THREAD` or `PROCESS`. The queue manager uses it for serialization purposes.

If the database manager is “thread-safe”, the value for *ThreadOfControl* can be `THREAD`, and the queue manager can call the database manager from multiple threads at the same time.

If the database manager is not thread-safe, the value for *ThreadOfControl* should be `PROCESS`. The queue manager serializes all calls to the database manager so that only one call at a time is made from within a particular process.

See “The XAResourceManager stanza” on page 82 for fuller descriptions of these attributes.

“Oracle configuration” on page 115 gives more information about the specific tasks you need to perform to configure WebSphere MQ with each of the supported database managers.

Oracle configuration

You need to perform the following tasks:

- Check Oracle level and apply patches if you have not already done so.
- Check environment variable settings.
- Enable Oracle XA support.
- Create the Oracle switch load file.
- Add XAResourceManager configuration information to the `qm.ini` file.
- Change the Oracle configuration parameters, if necessary.

Checking the environment variable settings

Ensure that your Oracle environment variables are set for queue manager processes as well as in your application processes. In particular, the following environment variables should always be set prior to starting the queue manager:

ORACLE_HOME

Is the Oracle home directory

ORACLE_SID

Is the Oracle SID being used

Enabling Oracle XA support

You need to ensure that Oracle XA support is enabled. In particular, an Oracle shared library must have been created; this happens during installation of the Oracle XA library.

During installation of Oracle, the library is built automatically. If you need to rebuild the library, refer to the *Oracle 9i Administrator's Reference* publication appropriate to your platform.

Creating the Oracle switch load file

The simplest method for creating the Oracle switch load file is to use the sample file. The source code used to create the Oracle switch load file is shown in Figure 10.

```
#include <cmqc.h>
#include "xa.h"

extern struct xa_switch_t xaosw;

struct xa_switch_t * MQENTRY MQStart(void)
{
    return(&xaosw);
}
```

Figure 10. Source code for Oracle switch load file, `oraswit.c`

The `xa.h` header file that is included is shipped with WebSphere MQ in the same directory as `oraswit.c`.

Creating the Oracle switch load file on OpenVMS systems

To create the Oracle switch load file on OpenVMS systems, oraswit.c must be compiled and linked against the Oracle client library oraclient_v901.exe.

1. Create the directory into which the Oracle switch load file, oraswit, is to be built.
2. Copy the following files from mqs_examples:[xatm] into this directory:
 - xa.h
 - oraswit.c
3. Compile the copied source file (oraswit.c).

For example:

```
$ cc oraswit0.c
```

4. Generate the switch load file:

```
$ link/share oraswit0.obj, sys$input/options  
ora_root:[util]oraclient_v901.exe/share  
SYMBOL_VECTOR=(MQStart=PROCEDURE)
```

5. The Oracle switch load file must be copied to sys\$share and then installed as a known image using the OpenVMS INSTALL utility.

For example:

```
$ install create/open/header/shared sys$share:oraswit0.exe
```

Before starting a queue manager (for which Oracle XA Transactional support is required) the logical name MQS_ORA_STARTUP must be defined. This definition must be the location of the Oracle command file that sets up the database specific, and the instance specific logicals required to access a given Oracle instance (for example, ORA_DB:ORAUSER_<database_name>.COM).

For example:

```
$ define/sys MQS_ORA_STARTUP disk$oracle9:[oracle9.db_mqseries]orauser_mqseries.com
```

Adding XAResourceManager configuration information for Oracle

The next step is to modify the qm.ini configuration file of the queue manager, to define Oracle as a participant in global units of work. You need to add an XAResourceManager stanza with the following attributes:

Name=name

This attribute is mandatory. Choose a suitable name for this participant. You could include the name of the database being updated.

SwitchFile=name

This attribute is mandatory. The fully-qualified name of the Oracle switch load file.

XAOpenString=string

The XA open string for Oracle has the following format:

```
Oracle_XA+Acc=P//|P/userName/password  
+SesTm=sessionTimeLimit  
[+DB=databaseName]  
[+GPwd=P/groupPassword]  
[+LogDir=logDir]  
[+MaxCur=maximumOpenCursors]  
[+SqlNet=connectString]
```

where:

Acc= Is mandatory and is used to specify user access information. **P//** indicates that no explicit user or password information is provided and that the **ops\$login** form is to be used. **P/userName/password** indicates a valid ORACLE user ID and the corresponding password.

SesTm=

Is mandatory and is used to specify the maximum amount of time that a transaction can be inactive before the system automatically deletes it. The unit of time is in seconds.

DB= Is used to specify the database name, where *DataBaseName* is the name Oracle precompilers use to identify the database. This field is required only when applications explicitly specify the database name (that is, use an **AT**[®] clause in their SQL statements).

GPwd=

GPwd is used to specify the server security password, where *P/groupPassWord* is the server security group password name. Server security groups provide an extra level of protection for different applications running against the same ORACLE instance. The default is an ORACLE-defined server security group.

LogDir=

LogDir is used to specify the directory on a local machine where the Oracle XA library error and tracing information can be logged. If a value is not specified, the current directory is assumed. Make sure that user mqm has write-access to this directory.

MaxCur=

MaxCur is used to specify the number of cursors to be allocated when the database is opened. It serves the same purpose as the precompiler option, `maxopencursors`.

SqlNet=

SqlNet is used to specify the SQL*Net connect string that is used to log on to the system. The connect string can be an SQL*Net V1 string, SQL*Net V2 string, or SQL*Net V2 alias. This field is required when you are setting up Oracle on a machine separate from the queue manager.

See the *Oracle Server Application Developer's Guide* for more information.

XACloseString=string

Oracle does not require an XA close string.

ThreadOfControl=THREAD | PROCESS

You do not need to specify this parameter on WebSphere MQ for HP OpenVMS platforms.

For fuller descriptions of each of these attributes, see "The XAResourceManager stanza" on page 82.

In Figure 11 on page 118, the database to be updated is called MQBankDB. Add a LogDir to the XA open string so that all error and tracing information is logged to the same place. It is assumed that the Oracle switch load file was copied to the sys\$share directory after it had been created.

```
XAResourceManager:  
  Name=Oracle MQBankDB  
  SwitchFile=sys$share:oraswit0  
  XAOpenString=Oracle_XA+Acc=P/jim/tiger+SesTm=35+LogDir=/tmp/ora.log+DB=MQBankDB
```

Figure 11. Sample XAResourceManager entry for Oracle

Changing Oracle configuration parameters

The queue manager and user applications use the user ID specified in the XA open string when they connect to Oracle.

- **Database privileges** The Oracle user ID specified in the open string must have the privileges to access the DBA_PENDING_TRANSACTIONS view.

The necessary privilege can be given using the following command, where userID is the user ID for which access is being given.

```
grant select on DBA_PENDING_TRANSACTIONS to userID;
```

See Chapter 7, “WebSphere MQ security,” on page 91 for more information about security.

Administration tasks

In normal operations, only a minimal amount of administration is necessary after you have completed the configuration steps. The administration job is made easier because the queue manager tolerates database managers not being available. In particular this means that:

- The queue manager can start at any time without first starting each of the database managers.
- The queue manager does not need to stop and restart if one of the database managers becomes unavailable.

This allows you to start and stop the queue manager independently from the database server.

Whenever contact is lost between the queue manager and a database, they need to resynchronize when both become available again. Resynchronization is the process by which any in-doubt units of work involving that database are completed. In general, this occurs automatically without the need for user intervention. The queue manager asks the database for a list of units of work that are in doubt. It then instructs the database to either commit or roll back each of these in-doubt units of work.

When a queue manager starts, it resynchronizes with each database. When an individual database becomes unavailable, only that database needs to be resynchronized the next time that the queue manager notices it is available again.

The queue manager attempts to regain contact with an unavailable database manager automatically as new global units of work are started. Alternatively, the **rsvmqtrn** command can be used to resolve explicitly all in-doubt units of work.

In-doubt units of work

A database manager may be left with in-doubt units of work if contact with the queue manager is lost after the database manager has been instructed to PREPARE. Until the database manager receives the COMMIT or ROLLBACK outcome from the queue manager, it needs to retain the database locks associated with the updates.

Because these locks prevent other applications from updating or reading database records, resynchronization needs to take place as soon as possible.

If for some reason you cannot wait for the queue manager to resynchronize with the database automatically, you could use facilities provided by the database manager to commit or rollback the database updates manually. This is called making a *heuristic* decision and should be used only as a last resort because of the possibility of compromising data integrity; you may end up committing the database updates when all of the other participants rollback, or vice versa.

It is far better to restart the queue manager, or use the `rsvmqtrn` command when the database has been restarted, to initiate automatic resynchronization.

Displaying outstanding units of work with the `dspmqtrn` command

While a database manager is unavailable it is possible to use the `dspmqtrn` command to check the state of outstanding units of work (UOWs) involving that database.

When a database manager becomes unavailable, before the two-phase commit process is entered, any in-flight UOWs in which it was participating are rolled back. The database manager itself rolls back its in-flight UOWs when it next restarts.

The `dspmqtrn` command displays only those units of work in which one or more participants are in doubt, awaiting the COMMIT or ROLLBACK from the queue manager.

For each of these units of work the state of each of the participants is displayed. If the unit of work did not update the resources of a particular resource manager, it is not displayed.

With respect to an in-doubt unit of work, a resource manager is said to have done one of the following things:

Prepared

The resource manager is prepared to commit its updates.

Committed

The resource manager has committed its updates.

Rolled-back

The resource manager has rolled back its updates.

Participated

The resource manager is a participant, but has not prepared, committed, or rolled back its updates.

The queue manager does not remember the individual states of the participants when the queue manager restarts. If the queue manager is restarted, but is unable

to contact a database manager, then the in-doubt units of work in which that database manager was participating are not resolved during restart. In this case, the database manager is reported as being in *prepared* state until such time as resynchronization has occurred.

Whenever the **dspmqrn** command displays an in-doubt UOW, it first lists all the possible resource managers that could be participating. These are allocated a unique identifier, *RMId*, which is used instead of the *Name* of the resource managers when reporting their state with respect to an in-doubt UOW.

Figure 12 shows the result of issuing the following command:

```
dspmqrn -m MY_QMGR

AMQ7107: Resource manager 0 is WebSphere MQ.
AMQ7107: Resource manager 1 is Oracle MQBankDB
AMQ7107: Resource manager 2 is Oracle MQFeeDB

AMQ7056: Transaction number 0,1.
      XID: formatID 5067085, gtrid_length 12, bqual_length 4
          gtrid [3291A5060000201374657374]
          bqual [00000001]
AMQ7105: Resource manager 0 has committed.
AMQ7104: Resource manager 1 has prepared.
AMQ7104: Resource manager 2 has prepared.
```

Figure 12. Sample *dspmqrn* output

The output from Figure 12 shows that there are three resource managers associated with the queue manager. The first is the resource manager 0, which is the queue manager itself. The other two resource manager instances are the MQBankDB and MQFeeDB Oracle databases.

The example shows only a single in-doubt unit of work. A message is issued for all three resource managers, which means that updates had been made to the queue manager and both Oracle databases within the unit of work.

The updates made to the queue manager, resource manager 0, have been *committed*. The updates to the Oracle databases are in *prepared* state, which means that Oracle must have become unavailable before it was called to commit the updates to the *MQBankDB* and *MQFeeDB* databases.

The in-doubt unit of work has an external identifier called an XID. This is the identifier that Oracle associates with the updates.

Resolving outstanding units of work with the **rsvmqtrn** command

The output shown in Figure 12 showed a single in-doubt UOW in which the commit decision had yet to be delivered to both Oracle databases.

In order to complete this unit of work, the queue manager and Oracle need to resynchronize when Oracle next becomes available. The queue manager uses the start of new units of work as an opportunity to attempt to regain contact with Oracle. Alternatively, you can instruct the queue manager to resynchronize explicitly using the **rsvmqtrn** command. You should do this soon after Oracle has been restarted so that any database locks associated with the in-doubt unit of work are released as quickly as possible.

This is achieved using the `-a` option which tells the queue manager to resolve all in-doubt units of work. In the following example, Oracle had been restarted so the queue manager was able to resolve the in-doubt unit of work:

```
rsvmqtrn -m MY_QMGR -a
```

Any in-doubt transactions have been resolved.

Mixed outcomes and errors

Although the queue manager uses a two-phase commit protocol, this does not completely remove the possibility of some units of work completing with mixed outcomes. This is where some participants commit their updates and some back out their updates.

Units of work that complete with a mixed outcome have serious implications because shared resources that should have been updated as a single unit of work are no longer in a consistent state.

Mixed outcomes are mainly caused when heuristic decisions are made about units of work instead of allowing the queue manager to resolve in-doubt units of work itself. Such decisions are outside the queue manager's control.

Whenever the queue manager detects a mixed outcome, it produces FFST™ information and documents the failure in its error logs, with one of two messages:

- If a database manager rolls back instead of committing:
AMQ7606 A transaction has been committed but one or more resource managers have rolled back.
- If a database manager commits instead of rolling back:
AMQ7607 A transaction has been rolled back but one or more resource managers have committed.

Further messages identify the databases that are heuristically damaged. It is then your responsibility to locally restore consistency to the affected databases. This is a complicated procedure in which you need first to isolate the update that has been wrongly committed or rolled back, then to undo or redo the database change manually.

Damage occurring due to software errors is less likely. Units of work affected in this way have their transaction number reported by message AMQ7112. The participants may be in an inconsistent state.

```

rsvmqtrn -m MY_QMGR

AMQ7107: Resource manager 0 is WebSphere MQ.
AMQ7107: Resource manager 1 is Oracle MQBankDB
AMQ7107: Resource manager 2 is Oracle MQFeeDB

AMQ7112: Transaction number 0,1 has encountered an error.
      XID: formatID 5067085, gtrid_length 12, bqual_length 4
          gtrid [3291A5060000201374657374]
          bqual [00000001]
AMQ7105: Resource manager 0 has committed.
AMQ7104: Resource manager 1 has prepared.
AMQ7104: Resource manager 2 has rolled back.

```

Figure 13. Sample `dspmqtrn` output for a transaction in error

The queue manager does not attempt to recover from such failures until the next queue manager restart. In Figure 13, this would mean that the updates to resource manager 1, the MQBankDB database, would be left in *prepared* state even if the `rsvmqtrn` was issued to resolve the unit of work.

Changing configuration information

After the queue manager has successfully started to coordinate global units of work you should take care when making changes to any of the XAResourceManager stanzas in the `qm.ini` file.

If you do need to change the `qm.ini` file you can do so at any time, but the changes do not take effect until after the queue manager has been restarted. For example, if you need to alter the XA open string passed to a database manager, you need to restart the queue manager for your change to take effect.

Note that if you remove an XAResourceManager stanza you are effectively removing the ability for the queue manager to contact that database manager.

Never change the *Name* attribute in any of your XAResourceManager stanzas. This attribute uniquely identifies that database manager instance to the queue manager. If this unique identifier is changed, the queue manager assumes that the database manager instance has been removed and a completely new instance has been added. The queue manager still associates outstanding units of work with the old *Name*, possibly leaving the database in an in-doubt state.

Removing database manager instances

If you do need to remove a database or database manager from your configuration permanently, you should first ensure that the database is not in doubt. You should perform this check before you restart the queue manager. Most database managers provide commands for listing in-doubt transactions. If there are any in-doubt transactions, first allow the queue manager to resynchronize with the database manager before you remove its XAResourceManager stanza.

If you fail to observe this procedure the queue manager still remembers all in-doubt units of work involving that database. A warning message, AMQ7623, is issued every time the queue manager is restarted. If you are never going to configure this database with the queue manager again, you can instruct the queue

manager to forget about the participation of the database in its in-doubt transactions using the `-r` option of the `rsvmqtrn` command.

Note: The queue manager finally forgets about transactions only when syncpoint processing has been completed with all participants.

There are times when you might need to remove an `XAResourceManager` stanza temporarily. This is best achieved by commenting out the stanza so that it can be easily reinstated at a later time. You may decide to take this action if you are suffering errors every time the queue manager contacts a particular database or database manager. Temporarily removing the `XAResourceManager` entry concerned allows the queue manager to start global units of work involving all of the other participants. An example of a commented out `XAResourceManager` stanza follows:

```
# This database has been temporarily removed
#XAResourceManager:
# Name=Oracle MQBankDB
# SwitchFile=sys$share:oraswit0
# XAOpenString=MQBankDB
```

Figure 14. Commented out XAResourceManager stanza

Chapter 9. The WebSphere MQ dead-letter queue handler

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

Messages can be put on the DLQ by queue managers, message channel agents (MCAs), and applications. All messages on the DLQ must be prefixed with a *dead-letter header* structure, MQDLH.

Messages put on the DLQ by a queue manager or a message channel agent always have an MQDLH; applications putting messages on the DLQ must supply an MQDLH. The *Reason* field of the MQDLH structure contains a reason code that identifies why the message is on the DLQ.

All WebSphere MQ environments need a routine to process messages on the DLQ regularly. WebSphere MQ 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 126
- “How the rules table is processed” on page 132
- “An example DLQ handler rules table” on page 133

Invoking the DLQ handler

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.ru1
```
- In the rules table. For example:

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

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

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

The **runmqdlq** command takes its input from SYS\$INPUT; you associate the rules table with **runmqdlq** by redirecting SYS\$INPUT from the rules table.

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. For the DLQ handler to put messages on queues with the authority of the user ID in the message context, you must also 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 255.

Attention: Running the DLQ handler without redirecting SYS\$INPUT to a rule file causes the DLQ handler to loop.

The sample DLQ handler, amqsdq

In addition to the DLQ handler invoked using the **runmqdlq** command, WebSphere MQ provides the source of a sample DLQ handler, **amqsdq**, whose function is similar to that provided by **runmqdlq**. You can customize **amqsdq** to provide a DLQ handler that meets your 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. 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*|' ')

The name of the DLQ you want to process:

1. Any INPUTQ value you supply as a parameter to the **runmqdlq** command 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* | ' ')

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

1. Any INPUTQM value you supply as a parameter to the **runmqdlq** command 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)

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

WAIT (YES | NO | *nnn*)

Whether the DLQ handler should wait for further messages to arrive on the DLQ when it detects that there are no further messages that it can process.

YES The DLQ handler waits indefinitely.

NO The DLQ handler ends when it detects that the DLQ is either empty or contains no messages that it can process.

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

Specify WAIT (YES) for busy DLQs, and WAIT (NO) or WAIT (*nnn*) for DLQs that have a low level of activity. If the DLQ handler is allowed to terminate, invoke it again using triggering. For more information about triggering, see the *WebSphere MQ Application Programming Guide*.

An alternative to including control data in the rules table is to supply the names of the DLQ and its queue manager as input parameters to the **runmqdlq** command. If you specify a value both in the rules table and as input to the **runmqdlq** command, the value specified on the **runmqdlq** command takes precedence.

If you include a control-data entry in the rules table, it must be the **first** entry in the table.

Rules (patterns and actions)

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

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

This rule instructs the DLQ handler to make three 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 rest of this section. Note the following:

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

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

The pattern-matching keywords

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

APPLIDAT (*ApplIdentityData* | *)

The *ApplIdentityData* value specified in the message descriptor, MQMD, of the message on the DLQ.

APPLNAME (*PutApplName* | *)

The name of the application that issued the **MQPUT** or **MQPUT1** call, as specified in the *PutApplName* field of the message descriptor MQMD of the message on the DLQ.

APPLTYPE (*PutApplType* | *)

The *PutApplType* value, specified in the message descriptor MQMD, of the message on the DLQ.

DESTQ (*QueueName* | *)

The name of the message queue for which the message is destined.

DESTQM (*QueueManagerName* | *)

The 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.

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

FORMAT (*Format* | *)

The name that the sender of the message uses to describe the format of the message data.

MSGTYPE (*MsgType* | *)

The message type of the message on the DLQ.

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

PERSIST (*Persistence* | *)

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

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

REASON (*ReasonCode* | *)

The reason code that describes why the message was put to the DLQ.

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

REPLYQ (*QueueName* | *)

The name of the reply-to queue specified in the message descriptor, MQMD, of the message on the DLQ.

REPLYQM (*QueueManagerName* | *)

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* | *)

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, used to describe how a matching message is to be processed, are described below.

ACTION (DISCARD | IGNORE | RETRY | FWD)

The action to be taken for any message on the DLQ that matches the pattern defined in this rule.

DISCARD

Delete the message from the DLQ.

IGNORE

Leave the message on the DLQ.

RETRY

If the first attempt to put the message on its destination queue fails, try again. The RETRY keyword sets the number of tries made to implement an action. The RETRYINT keyword of the control data controls the interval between attempts.

FWD Forward the message to the queue named on the FWDQ keyword.

You must specify the ACTION keyword.

FWDQ (*QueueName* | &DESTQ | &REPLYQ)

The name of the message queue to which to forward the message when ACTION (FWD) is requested.

QueueName

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

&DESTQ

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

&REPLYQ

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

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

FWDQM (QueueManagerName | &DESTQM | &REPLYQM | ' _')

The queue manager of the queue to which to forward a message.

QueueManagerName

The name of the queue manager of the queue to which to forward a message when ACTION (FWD) is requested.

&DESTQM

Take the queue manager name from the *DestQMGrName* field in the MQDLH structure.

&REPLYQM

Take the queue manager name from the *ReplyToQMGr* field in the message descriptor, MQMD.

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

HEADER (YES | NO)

Whether the MQDLH should remain on a message for which ACTION (FWD) is requested. By default, the MQDLH remains on the message. The HEADER keyword is not valid for actions other than FWD.

PUTAUT (DEF | CTX)

The authority with which messages should be put by the DLQ handler:

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

CTX Put the messages 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)

The number of times, in the range 1–999 999 999, to try an action (at the interval specified on the RETRYINT keyword of the control data). 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 only once 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.
- There can be any number of blanks 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 must not be greater than 72 characters.
- Use the plus sign (+) as the last nonblank character on a line to indicate that the rule continues from the first nonblank character in the next line. Use the minus sign (-) as the last nonblank character on a line to indicate that the rule continues from the start of the next line. Continuation characters can occur within keywords and parameters.

For example:

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

results in 'ABCD', and

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

results in 'ABC D'.

- Comment lines, which begin with an asterisk (*), can occur anywhere in the rules table.
- Blank lines are ignored.
- Each entry in the DLQ handler rules table comprises one or more keywords and their associated parameters. The parameters must follow these syntax rules:
 - Each parameter value must include at least one significant character. The delimiting quotation marks in quoted values are not considered significant. For example, these parameters are valid:

FORMAT('ABC')	3 significant characters
FORMAT(ABC)	3 significant characters
FORMAT('A')	1 significant character
FORMAT(A)	1 significant character
FORMAT(' ')	1 significant character

These parameters are invalid because they contain no significant characters:

```
FORMAT('')
FORMAT( )
FORMAT()
FORMAT
```

- Wildcard characters are supported. You can use the question mark (?) instead of any single character, except a trailing blank; you can use the asterisk (*) instead 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. You can use the asterisk (*) instead of an entire numeric parameter, but not as part of a numeric parameter. For example, these are valid numeric parameters:

MSGTYPE(2)	Only reply messages are eligible
------------	----------------------------------

MSGTYPE(*)	Any message type is eligible
MSGTYPE('*')	Any message type is eligible

However, MSGTYPE('2*') is not valid, because it includes an asterisk (*) as part of a numeric parameter.

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

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

- Enclose strings that contain blanks, lowercase characters, or special characters other than period (.), forward slash (/), underscore (_), and percent sign (%) in single quotation marks. Lowercase characters not enclosed in quotation marks are folded to uppercase. If the string includes a quotation, use two single quotation marks 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 the DLQ handler finds a rule with a matching pattern, it takes the action from that rule. The DLQ handler increments the retry count for a rule by 1 whenever it applies that rule. If the first try fails, the DLQ handler tries again until the number of tries 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.

Note:

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 can consist of an action only. Note, however, that action-only rules are applied to all messages on the queue that have MQDLHs and that have not already been processed in accordance with other rules in the table.
3. The rules table is validated when the DLQ handler starts, and errors are flagged at that time. (Error messages issued by the DLQ handler are described

in the *WebSphere MQ Messages* manual..) You can make changes to the rules table at any time, but those changes do not come into effect until the DLQ handler restarts.

4. The DLQ handler does not alter the content of messages, the MQDLH, or the message descriptor. The DLQ handler always puts messages to other queues with the message option MQPMO_PASS_ALL_CONTEXT.
5. Consecutive syntax errors in the rules table might not be recognized because the rules table is designed to eliminate the generation of repetitive errors during validation.
6. The DLQ handler opens the DLQ with the MQOO_INPUT_AS_Q_DEF option.
7. Multiple instances of the DLQ handler can run concurrently against the same queue, using the same rules table. However, it is more usual for there to be a one-to-one relationship between a DLQ and a DLQ handler.

Ensuring that all DLQ messages are processed

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

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

You can take specific measures to enable the DLQ handler to empty the DLQ. For example, try not to use ACTION (IGNORE), which simply leaves messages on the DLQ. (Remember that ACTION (IGNORE) is assumed for messages that are not explicitly addressed by other rules in the table.) Instead, for those messages that you would otherwise ignore, use an action that moves the messages to another queue. For example:

```
ACTION (FWD) FWDQ (IGNORED.DEAD.QUEUE) HEADER (YES)
```

Similarly, make the final rule in the table a catchall to process messages that have not been addressed by earlier rules in the table. For example, the final rule in the table could be something like this:

```
ACTION (FWD) FWDQ (REALLY.DEAD.QUEUE) HEADER (YES)
```

This forwards messages that fall through to the final rule in the table 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

The following example rules table contains a single control-data entry and several rules:

```
*****  
*           An example rules table for the runmqdlq command           *  
*****  
* Control data entry
```

```

* -----
* If no queue manager name is supplied as an explicit parameter to
* runmqdlq, use the default queue manager for the machine.
* If no queue name is supplied as an explicit parameter to runmqdlq,
* use the DLQ defined for the local queue manager.
*
inputqm(' ') inputq(' ')

* Rules
* -----
* We include rules with ACTION (RETRY) first to try to
* deliver the message to the intended destination.
* If a message is placed on the DLQ because its destination
* queue is full, attempt to forward the message to its
* destination queue. Make 5 attempts at approximately
* 60-second intervals (the default value for RETRYINT).

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

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

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

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

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

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

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

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

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

* Messages that are not persistent run the risk of being
* lost when a queue manager terminates. If an application
* is sending nonpersistent messages, it should be able
* to cope with the message being lost, so we can afford to
* discard the message. PERSIST(MQPER_NOT_PERSISTENT) ACTION(DISCARD)
* For performance and efficiency reasons, we like to keep
* the number of messages on the DLQ small.
* If we receive a message that has not been processed by
* an earlier rule in the table, we assume that it
* requires manual intervention to resolve the problem.
* Some problems are best solved at the node where the
* problem was detected, and others are best solved where
* the message originated. We don't have the message origin,
* but we can use the REPLYQM to identify a node that has
* some interest in this message.
* Attempt to put the message onto a manual intervention

```

```
* queue at the appropriate node. If this fails,  
* put the message on the manual intervention queue at  
* this node.
```

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

Chapter 10. WebSphere MQ for OpenVMS and clustering

OpenVMS clusters and *WebSphere MQ queue manager clusters* are two different things, independent of one another.

Note: When the term *cluster* is used, it refers to a WebSphere MQ queue manager cluster. An OpenVMS cluster is always referred to as *OpenVMS cluster*.

WebSphere MQ queue manager clusters do not necessarily use OpenVMS cluster intercommunication protocols, the OpenVMS cluster distributed lock manager or the OpenVMS cluster file system. All communication between queue managers in a WebSphere MQ cluster is via WebSphere MQ channels using one of the supported protocols. Thus it is possible to configure WebSphere MQ queue manager clusters with queue managers that run on OpenVMS systems that are not part of the same OpenVMS cluster.

If a WebSphere MQ queue manager is configured within an OpenVMS cluster, the WebSphere MQ queue manager can run only on one OpenVMS node (referred to as a node for the rest of this chapter) within the OpenVMS cluster at a time. The function of a single WebSphere MQ queue manager cannot be distributed across multiple OpenVMS nodes within an OpenVMS cluster. If an attempt is made to start a WebSphere MQ queue manager on more than one OpenVMS node, an error is returned. However, if there are multiple WebSphere MQ queue managers configured in an OpenVMS cluster they can be run on different OpenVMS nodes within the OpenVMS cluster.

In an OpenVMS cluster, failover sets provide a higher availability of WebSphere MQ queue managers. This enables a queue manager to be automatically restarted on another OpenVMS cluster node if a failure occurs. This feature can be used with or without WebSphere MQ queue manager clusters. (See “OpenVMS cluster failover sets” on page 138).

Installing WebSphere MQ in an OpenVMS cluster

Installing WebSphere MQ for HP OpenVMS, V6.0 in an OpenVMS cluster is very similar to installing WebSphere MQ on a standalone OpenVMS system. However, before installing, you need to consider the following:

- If there are multiple system disks in the OpenVMS cluster, WebSphere MQ needs to be installed on each system disk that has a node booted from it *and* that has to run WebSphere MQ. WebSphere MQ needs to be installed only once per system disk, not once per node.
- The disk holding the MQS_ROOT directory structure must be mounted system wide on the OpenVMS nodes that are to run the queue managers contained within the directory structure. It is possible to have different MQS_ROOT directory structures for each node. But, if failover sets are to be configured, each OpenVMS node in a failover set must refer to the same MQS_ROOT directory structure. When installing WebSphere MQ, you must specify the MQS_ROOT directory (in response to the question ‘Enter the root device for the WebSphere MQ datafiles:’) for each installation.
- If the disk containing the log files for a queue manager is different from the disk containing MQS_ROOT, the disk containing the log files must be mounted system-wide on all nodes in a Failover set.

- WebSphere MQ uses an account MQM which has a default directory SYS\$SPECIFIC:[MQS_SERVER]. This directory is created only for the node on which WebSphere MQ is installed. The directory must be created for each additional node that boots from the same system disk *and* that has to run WebSphere MQ. This can be done by executing the following DCL commands on each additional node:

```
$create/directory sys$specific:[mqc_server]/owner=[mqc_server] -
/protection=(s:rwed,o:rwed,g,w)
$set sec/acl=(identifier=mqm,options=default,access=r+w+e+d+c) -
sys$specific:[000000]mqc_server.dir
$set sec/acl=(identifier=mqm,access=r+w+e+d+c) -
sys$specific:[000000]mqc_server.dir
```

- On OpenVMS clusters sharing the same system disk, only one version of WebSphere MQ (V5.3 or V6.0) can be installed. On OpenVMS clusters where the cluster nodes have their own independent system disk, it is possible to install different versions of MQ running on different nodes. In that case cluster nodes with queue managers belonging to the same failover set must have the same version of WebSphere MQ, because they share a common MQS_ROOT directory.

OpenVMS cluster failover sets

Overview of OpenVMS cluster failover sets

OpenVMS cluster failover sets allow WebSphere MQ queue managers to be automatically restarted on another OpenVMS node in an OpenVMS cluster if the WebSphere MQ queue manager fails. The following types of failure are supported by OpenVMS cluster failover sets:

- Halt of an OpenVMS node running a WebSphere MQ queue manager
- System crash of an OpenVMS node running a WebSphere MQ queue manager
- Shutdown of an OpenVMS node running a WebSphere MQ queue manager without the clean ending of the WebSphere MQ queue manager
- The failure of a WebSphere MQ queue manager Execution Controller process

The following types of failure are not supported by OpenVMS cluster failover sets:

- A fault on an OpenVMS node running a WebSphere MQ queue manager that does not cause the node or the WebSphere MQ queue manager to fail.
- The failure of a WebSphere MQ queue manager process except for the Execution Controller process. A WebSphere MQ queue manager is never automatically restarted on the same node.
- A software or hardware failure of the disk holding the WebSphere MQ queue manager queue files and log data.
- Corruption of the WebSphere MQ queue manager queue files or log data.

OpenVMS cluster failover sets are supported only for queue managers that use the TCP/IP protocol for WebSphere MQ channels. The following TCP/IP stacks are supported:

- HP TCP/IP Services for OpenVMS V5.4-15 (Alpha) V5.5-81 (Itanium®)
- Process Software TCPware for OpenVMS V5.6 (Alpha) V5.6 (Itanium)
- Process Software Multinet for OpenVMS V5.0 (Alpha Only)

OpenVMS cluster failover set concepts

An OpenVMS cluster failover set is a collection of OpenVMS nodes that can potentially run a WebSphere MQ queue manager. There may be between one and four OpenVMS nodes in an OpenVMS cluster failover set and all the OpenVMS nodes must be members of the same OpenVMS cluster. An OpenVMS cluster failover set is specific to one WebSphere MQ queue manager. There may be more than one OpenVMS cluster failover set configured in an OpenVMS cluster. Note that the maximum length of a queue manager name supported by OpenVMS cluster failover sets is 25 characters.

Failover is the process by which a WebSphere MQ queue manager is restarted on another OpenVMS node when a supported failure occurs. After this process has completed the WebSphere MQ queue manager is said to have failed over.

Failback is the process by which a WebSphere MQ queue manager is restarted on its original OpenVMS node after a failure has been resolved. OpenVMS cluster failover sets do not support automatic failback but it can be performed manually. After this process has completed the WebSphere MQ queue manager is said to have failed back.

A *failover monitor* is a process that runs on each member of an OpenVMS cluster failover set. The failover monitors are responsible for performing all functions of the failover sets. The failover monitors within an OpenVMS cluster failover set cooperate with one another to provide these functions. A failover monitor is started using the `runmqfm` command.

One failover monitor is nominated as the *watcher failover monitor* and this failover monitor is said to be in a *watching* state. The first failover monitor to start in a failover set is the initial watcher failover monitor. A failover set becomes *live* when the first failover monitor is started. If the watcher failover monitor fails, or the OpenVMS node on which it is running fails, another failover monitor is automatically nominated as the watcher failover monitor. The watcher failover monitor is responsible for checking that the WebSphere MQ queue manager is running and for initiating a failover operation if a supported failure occurs. Any operation that must be performed on another OpenVMS node is forwarded by the watcher failover monitor to the failover monitor running on the relevant OpenVMS node which actually performs the operation.

OpenVMS cluster failover sets are administered using the DCL command **failover**. The **failover** command can be used from any node in the OpenVMS cluster failover set. All commands are sent to the watcher failover monitor which then decides which failover monitor should process the command and if necessary forwards it onto another failover monitor.

The OpenVMS cluster failover set configuration file holds the details of the OpenVMS cluster failover set including the number and names of the OpenVMS nodes. The file is called `FAILOVER.INI` and resides in the directory `MQS_ROOT:[MQM.QMGRS.queuemanagername]`. It is a text file which is modified with a text editor and must be created prior to starting the first failover monitor. A template configuration file called `FAILOVER.TEMPLATE` is provided in the directory `MQS_EXAMPLES`. The parameters in the configuration file cannot be changed dynamically. For a change to take effect all failover monitors must be stopped and then started again. Care must be taken when this is done because automatic failover of the WebSphere MQ queue manager cannot occur when the failover monitors are not started.

For a WebSphere MQ queue manager in a failover set, all WebSphere MQ commands continue to work as normal except for the **strmqm** and **endmqm** commands. These two commands return an error when a WebSphere MQ queue manager is in a live failover set. The **failover** command must be used to start and end the WebSphere MQ queue manager.

The OpenVMS node priority is the priority given to each OpenVMS node in the OpenVMS cluster failover set and is used to determine on which OpenVMS node the queue manager should be started after a failure has occurred. The OpenVMS node with the lowest numeric priority value has the highest priority.

The OpenVMS cluster failover set TCP/IP address is the TCP/IP address assigned to the failover set. All channels that refer to the failover set queue manager must be configured to specify this TCP/IP address in the connection name. Each OpenVMS cluster failover set must use a unique TCP/IP address. All OpenVMS nodes in the OpenVMS cluster failover set must have an interface TCP/IP address in the same subnet and the OpenVMS cluster failover set TCP/IP address must be in the same subnet.

Preparing to configure an OpenVMS cluster failover set

The following steps must be taken before configuring an OpenVMS cluster failover set:

1. Create the queue manager using **crtmqm** if it does not already exist.
2. Obtain a TCP/IP address for the OpenVMS cluster failover set.
3. Create or modify WebSphere MQ channels to use the OpenVMS cluster TCP/IP failover set TCP/IP address.
4. Decide on the OpenVMS nodes that are to be in the OpenVMS cluster failover set and decide their priorities.
5. Ensure that the MQS_ROOT logical refers to the same directory on all OpenVMS nodes in the OpenVMS cluster failover set and that the disk is mounted system-wide on all of the nodes. The disks containing the MQS_ROOT directory and the log files should not be MSCP served from one node in the failover set to another node because if the node serving the disk becomes unavailable, the node to which the disk is served is no longer able to access the disks.

Configuring an OpenVMS cluster failover set

The following steps are required to configure an OpenVMS cluster failover set:

1. Copy the MQS_EXAMPLES:FAILOVER.TEMPLATE file to MQS_ROOT:[MQM.QMGRS.queuemanagename] FAILOVER.INI.
2. Edit the MQS_ROOT:[MQM.QMGRS.queuemanagename] FAILOVER.INI file and modify for this OpenVMS cluster failover set configuration. (See "Editing the FAILOVER.INI configuration file" on page 141.)
3. Edit the **START_QM.COM**, **END_QM.COM** and **TIDY_QM.COM** command procedures. (See "Command procedures used by failover sets" on page 142.)
4. Set up ICC security for the ICC Association used by the failover monitors (See "Setting up security for ICC associations" on page 148.)
5. Start a failover monitor on each node in the OpenVMS cluster failover set using the **runmqfm -m queuemanagename** command.
6. Start the queue manager using the **failover -m queuemanagename -n nodename -s** command.

7. Modify the site specific shutdown to:
 - End or Move the queue manager if it is running on a node when it is shut down.
 - Halt the failover monitor.

OpenVMS cluster failover set post-configuration tasks

The following are tasks you can perform after your cluster failover set has been configured:

- Edit the system startup file to start the failover monitor processes on each node in the OpenVMS cluster failover set using the **runmqfm** command. The **runmqfm** command should be placed after the command to start WebSphere MQ.
- If it is required to start the queue manager automatically on system startup, place a command in the system startup on the relevant node to start the queue manager after the failover monitor has been started. The command to start the queue manager on a node is **failover -m queuemanagername -n nodename -s**.
- Modify the site specific shutdown to end the failover monitor on shutdown of the system. Also End or Move the queue manager if it is running on a node when it is shut down.

Editing the FAILOVER.INI configuration file

The FAILOVER.INI file must be customized for each OpenVMS cluster failover set. The meaning of each of the fields is listed in Table 7. The template configuration file supplied in MQS_EXAMPLES is included in the Appendix F, “OpenVMS cluster failover set templates,” on page 293. Any line in the file that begins with a '#' character is ignored when it is read by a failover monitor process. The character case of the field names within the file must be as specified in the template file. Each field name must be followed by an '=' character and then the associated value. All the fields in the template file are mandatory so no fields must be removed.

Table 7. Description of the fields within the FAILOVER.INI file

Field name	Description
IpAddress	The TCP/IP address to be used by the failover set
PortNumber	The TCP/IP port number used by the listener for the queue manager
TimeOut	This time out value is passed to the EndCommand procedure. See “Command procedures used by failover sets” on page 142.
StartCommand	The command procedure used to start the queue manager
EndCommand	The command procedure used to end the queue manager
TidyCommand	The command procedure used to tidy up on a node after a queue manager failure in which the OpenVMS node survives
LogDirectory	The directory which holds the log files created by the StartCommand, EndCommand and TidyCommand procedures
NodeCount	The number of nodes in the failover set. The number of node triplets defined after this field must correspond to this value. The maximum number of nodes supported is four.
NodeName	The node name of the node. That is the value specified for the SCSNODE OpenVMS system parameter.

Table 7. Description of the fields within the FAILOVER.INI file (continued)

Field name	Description
Interface	The TCP/IP interface name for the node when using the Digital TCP/IP Services for OpenVMS TCP/IP stack. This can be obtained from the output of the \$tcpip show interface command. This field is not used when using the TCPware for OpenVMS TCP/IP or Multinet for OpenVMS TCP/IP stacks but the default value of we0 should still be specified. (Do not remove the field from the configuration file.)
Priority	This is the priority given to this node within the failover set. The value must be between 1 and 10. A value of 1 is the highest priority. Multiple nodes can have the same priority. When a failure occurs or no specific node is specified in a failover -s or -f command, the queue manager is started on the highest priority node that is available.

Command procedures used by failover sets

Failover sets use three command procedures to implement some of its functions. The locations of these command procedures are specified in the FAILOVER.INI configuration file by the field names StartCommand, EndCommand and TidyCommand. Template files for these command procedures, with names **START_QM.TEMPLATE**, **END_QM.TEMPLATE** and **TIDY_QM.TEMPLATE** respectively, are provided in MQS_EXAMPLES. These files are listed in Appendix F, “OpenVMS cluster failover set templates,” on page 293.

The command procedures are passed five or six parameters. These are listed in Table 8:

Table 8. Parameters passed to command procedures

Parameter	Value
P1	Queue manager name
P2	Queue manager directory name
P3	Cluster TCP/IP address
P4	Node Interface name
P5	Listener port number
P6	End queue manager timeout (EndCommand procedure only)

The StartCommand procedure is used to start the queue manager in the following circumstances:

- When explicitly specified with the **-s** flag of the **failover** command
- When the queue manager is moved to another OpenVMS node using the **-f** flag of the **failover** command.
- When automatically restarted after a queue manager failure

By default the StartCommand procedure configures the failover set TCP/IP address on the node to run the Queue manager and then starts the queue manager using the **strmqm -m queuemanagername** command. Depending on system requirements the command procedure can be modified in the following ways:

- Change the **strmqm** command
- Add commands to start additional WebSphere MQ processes such as the listener

- Add commands to start application processes

The StartCommand procedure must exit with a status of 1 for the queue manager to be monitored after the queue manager has started.

The EndCommand procedure is used to end the queue manager in the following circumstances:

- When explicitly specified with the `-e` flag of the **failover** command
- When the queue manager is moved to another OpenVMS node using the `-f` flag of the **failover** command

By default the EndCommand procedure attempts to end the queue manager with the **endmqm -i** *queuemanagername* command. If the queue manager has not ended within the timeout period specified in the configuration file, the procedure attempts to end the queue manager with the **endmqm -p** *queuemanagername* command. If the queue has still not ended within another timeout period, the queue manager is ended by deleting the Execution Controller process. Once the Queue manager is ended, the failover set TCP/IP address is deconfigured. If the queue manager is successfully ended using the **endmqm** commands, the status `SS$_NORMAL` is returned. If the queue manager is ended by deleting the Execution Controller, the status `SS$_ABORT` is returned. If the queue manager is not ended after a third timeout period, the status `SS$_TIMEOUT` is returned. These statuses are used by the watcher failover monitor to determine the outcome of the EndCommand procedure and sets the state of the failover set accordingly. Depending on system requirements the command procedure can be modified in the following ways:

- Add commands to end additional WebSphere MQ processes such as the listener
- Add commands to end Application processes

The TidyCommand procedure is used to tidy up on an OpenVMS node if the queue manager fails but the OpenVMS node continues to run.

By default the TidyCommand procedure deconfigures the failover set TCP/IP address. Depending on system requirements the command procedure can be modified in the following ways:

- Add commands to end any WebSphere MQ processes that are still running such as the listener
- Add commands to end Application processes that are still running

The template files are set up by default to use Digital TCP/IP Services for OpenVMS commands to configure and de-configure the TCP/IP address. If you are using TCPware for OpenVMS or MultiNet for OpenVMS, comment out (deactivate) the Digital TCP/IP Services for OpenVMS commands and uncomment (activate) the TCPware for OpenVMS or MultiNet for OpenVMS commands.

Administration of failover sets

Failover sets must be managed from the SYSTEM account or from a WebSphere MQ Administration account. Failover sets are managed using two commands **DCL runmqfm** and **failover**. The **runmqfm** command is used to start the **failover** monitors and the **failover** command performs all other administration tasks.

Startup of failover monitors

Failover monitors are started by executing the **runmqfm** command on the OpenVMS node on which it is required to have the failover monitor started. For example to start a failover monitor for queue manager TESTQM, use the command:

```
$ runmqfm -m TESTQM
```

This creates a detached process with a name based on the queue manager name and ending in **_FM**. In this example the process name is TESTQM_FM. This process is listed in a monmq active display.

If a log file is required, this can be specified by redirecting output of the **runmqfm** command and additional debug information can be displayed in the log file by specifying the **-d** flag. For example:

```
$ runmqm -m TESTQM -d > sys$manager:fm.log
```

Note that the **runmqfm** command only starts failover monitor processes it does not start the queue manager.

Starting a queue manager within a failover set

To start a queue manager within a failover set, at least one failover monitor must be running and there must be a failover monitor running on the node on which you wish to start the queue manager. A queue manager is started using the **-s** flag of the **failover** command. The command can be executed from any OpenVMS node within the failover set. For example if you wish to start the queue manager TESTQM on node BATMAN, use the following command:

```
$ failover -m TESTQM -n BATMAN -s
```

If it is required to start the queue manager on the OpenVMS node with the highest priority available omit the **-n** flag from the command. For example:

```
$ failover -m TESTQM -s
```

Note that once a failover monitor is started for a queue manager (on any node), any attempt to use the **strmqm** command to start a queue manager fails. However, once all failover monitors have been stopped for a queue manager, the **strmqm** command can be used normally.

Ending a queue manager within a failover set

To end a queue manager within a failover set, there must be a failover monitor running on the node on which the queue manager is running. A queue manager is ended using the **-e** flag of the **failover** command. The command can be executed from any OpenVMS node within the failover set. For example if you wish to end

the queue manager TESTQM, use the following command:

```
$ failover -m TESTQM -e
```

Note that once a failover monitor is started for a queue manager (on any node), any attempt to use the **endmqm** command to end a queue manager fails. However, once all failover monitors have been stopped for a queue manager, the **endmqm** command can be used normally.

Moving a queue manager within a failover set

Moving a queue manager within a failover set means stopping the queue manager on the node on which it is currently running and then starting it again on another node within the failover set. To move a queue manager within a failover set, there must be a failover monitor running on the node on which the queue manager is currently running and there must be a failover monitor running on the node on which you wish to move the queue manager.

A queue manager is moved using the **-f** flag of the **failover** command. The command can be executed from any OpenVMS node within the failover set. For example if you wish to move the queue manager TESTQM to node ROBIN, use the following command:

```
$ failover -m TESTQM -n ROBIN -f
```

If it is required to move the queue manager to the OpenVMS node with the highest priority available, omit the **-n** flag from the command. For example:

```
$ failover -m TESTQM -f
```

Displaying the state of a failover set

There are three types of state that describe the overall state of the failover set:

- The failover set queue manager state
- The failover set node queue manager states (one for each node)
- The failover set Node Monitor states (one for each node)

The possible values of each of the states are described in the following three tables.

Table 9. Failover set queue manager states

State	Description
STOPPED	The queue manager has never been started in the failover set or has been cleanly shutdown
STARTED	The queue manager has been started in the failover set. The failover set tries to restart the queue manager if a queue manager failure occurs.

Table 10. Failover set node queue manager states

State	Description
AVAILABLE	The node is free to have the queue manager started on it if a failure occurs on another node.
RUNNING	The queue manager is running on this node.
EXCLUDED	The queue manager was stopped on this node in an unclean manner without the node itself failing. If a queue manager fails on another node it is not restarted on this node.

Table 11. Failover set node monitor states

State	Description
STARTED	A failover monitor is running on this node but it is not the watcher.
WATCHING	A failover monitor is running on this node and it is the watcher.
STOPPED	There is no failover monitor running on this node.

The state of a failover set is displayed using the `-q` flag of the **failover** command. There must be at least one failover monitor process running and the command can be executed from any node within the failover set. For example to display the state of the failover set for the queue manager TESTQM, use the following command:

```
$ failover -m TESTQM -q
```

Sample output from the command is shown below:

```
OpenVMS Cluster Failover Set - Configuration and State.
```

```
Queue Manager Name      : TESTQM
Sequence No             : 11
TCP/IP Address          : 10.20.30.40
Listener Port Number    : 1414
Timeout to end the Queue Manager : 30
Queue Manager state in Failover Set : STARTED
```

```
OpenVMS Node - Configuration and State
```

```
Node name               : BATMAN
Priority                 : 2
TCP/IP Interface        : we0
Queue Manager state     : RUNNING
Failover Monitor state  : WATCHING
```

```
Node name               : ROBIN
Priority                 : 1
TCP/IP Interface        : we0
Queue Manager state     : EXCLUDED
Failover Monitor state  : STARTED
```

Setting DCL symbols to the state of a failover set

In some cases it may be necessary to write DCL command procedures to control failover sets. The `-l` flag of the **failover** command sets three local DCL symbols to indicate the state of the failover set. These symbols can then be used to take

conditional actions based on the state of the queue manager. There must be at least one failover monitor process running and the command can be executed from any node within the failover set. The symbols that are set are shown in Table 12.

Table 12. DCL symbols and description

DCL symbol name	Description
MQS\$QMGR_NODE	Set to the OpenVMS node that is running the queue manager or a null string if there is no queue manager running
MQS\$AVAILABLE_NODES	Set to the list of OpenVMS nodes that are available to run the queue manager. That is the nodes that are in the queue manager AVAILABLE state and that have a failover monitor running.
MQS\$MONITOR_NODES	Set to the list of OpenVMS nodes that have a failover monitor running on them.

For example to set the symbols to the state of the failover set for the queue manager TESTQM, use the following command:

```
$ failover -m TESTQM -l
```

Example results for the setting of the symbols are shown below:

```
MQS$AVAILABLE_NODES = ""
MQS$MONITOR_NODES = "BATMAN,ROBIN"
MQS$QMGR_NODE = "BATMAN"
```

Halting a failover monitor process

The failover monitor process on an OpenVMS node can be halted using the `-h` flag of the **failover** command. The command can be executed from any node within the failover set. For example to halt the failover monitor for queue manager TESTQM on node BATMAN use the following command:

```
$ failover -m TESTQM -n BATMAN -h
```

If the failover monitor being halted is the watcher failover monitor, another failover monitor becomes the watcher if one exists. If the failover monitor being halted is the last failover monitor for the failover set, the failover set is no longer live. In this case, the queue manager can now be started and ended using the **strmqm** and **endmqm** commands. The `-h` flag of the **failover** command never ends a queue manager. If the queue manager is running on the OpenVMS node on which the failover monitor is being halted, the queue manager continues to run.

Executing commands while an update is in progress

The **failover** commands with flags `-s`, `-e`, `-f` and `-c` are considered updates. While these commands are in progress, an update in progress flag is set by the watcher failover monitor. When this flag is set, any other update and failover monitor halt

command fails because simultaneous updates are not allowed. Non-update commands such as the -q and -l flags continue to work when an update is in progress.

In rare circumstances, a failed update may leave the update in progress flag set. The -u flag of the **failover** command clears the update in progress flag. This command should be used with caution. For example to clear the update in progress flag for queue manager TESTQM, use the following command:

```
$ failover -m TESTQM -u
```

Changing the state of a failover set

In some circumstances it may be necessary to change the state of a failover set. This is achieved using the -c flag of the **failover** command. This is most likely needed when a queue manager state on a node is EXCLUDED after a failure and you want to change the state back to AVAILABLE after cleaning up the node. For example, to change the state to AVAILABLE for queue manager TESTQM on node BATMAN, use the following command:

```
$ failover -m TESTQM -n BATMAN -c -qmgr available
```

Also you may want to temporarily exclude a node from being considered as a candidate for running the queue manager by changing the Node queue manager state from AVAILABLE to EXCLUDED. For example to change the state to EXCLUDED for queue manager TESTQM on node BATMAN, use the following command:

```
$ failover -m TESTQM -n BATMAN -c -qmgr excluded
```

It is also possible to change all of the other states but any change takes effect only if the change requested is consistent with the running system. For instance, if a failover monitor is running on a node and you try to change the Monitor state to STOPPED, this change does not take effect. Apart from changing the Node queue manager states between EXCLUDED and AVAILABLE, it should not be necessary to use the change state command because every 30 seconds the watcher failover monitor performs an integrity check and makes any changes to the states if there is a discrepancy with the running system.

Setting up security for ICC associations

The failover set monitor and client programs use OpenVMS Intra Cluster Communication (ICC) calls to pass messages. To prevent unauthorized users sending messages to the failover monitor processes, the security for the ICC Associations should be configured in the **SYSS\$STARTUP:ICC\$SYSTARTUP.COM** command procedure.

Each failover set uses two association names: one with the name of the queue manager which is used for communication with the watcher failover monitor and the other with the name of the queue manager with **_MQ_FM** appended which is used to communicate with each failover monitor.

An example is shown in Figure 15 of the entries required in **ICC\$SYSTARTUP.COM** for each node in a failover set. There are two nodes in the failover set called **BATMAN** and **ROBIN** and the queue manager name is **TESTQM**.

```

$! ----- List Nodes with Special Actions -----
$!
$ nodeactions = "/BATMAN/ROBIN/"
$ if f$locate("/"+nodename+"/",nodeactions) .eq. f$length(nodeactions) -
  then goto exit ! No action for this node
$ goto 'nodename' ! Go to action code for this node
$!
$! ----- Major Nodes -----
$BATMAN:
$ROBIN:
$!
$! Place in here calls to @SYS$MANAGER:ICC$CREATE_SECURITY_OBJECT and
$! @SYS$MANAGER:ICC$ADD_REGISTRY_TABLE that apply to FAilover nodes in the
$! cluster
$!
$!
$ @SYS$MANAGER:ICC$CREATE_SECURITY_OBJECT ICC$::"TESTQM" -
"/owner=MQM/acl=((id=MQM,access=open+access),(id=*,access=none))"
$!
$ @SYS$MANAGER:ICC$CREATE_SECURITY_OBJECT 'nodename'::"TESTQM" -
"/owner=MQM/acl=((id=MQM,access=open+access),(id=*,access=none))"
$!
$ @SYS$MANAGER:ICC$CREATE_SECURITY_OBJECT 'nodename'::"TESTQM_MQ_FM" -
"/owner=MQM/acl=((id=MQM,access=open+access),(id=*,access=none))"
$!
$ set security/class=logical_name_table icc$registry_table -
/acl=(id=MQM,access=read+write)
$!
$ GOTO EXIT
$!

```

Figure 15. Sample entry required for **ICC\$SYSTARTUP.COM**

Note that ICC Association names are limited to 31 characters, therefore the maximum supported length of WebSphere MQ queue manager name is 25 characters when used in a failover set. Further information on setting up the security of ICC Associations can be found in the *OpenVMS System Manager's Manual*.

Troubleshooting problems with failover sets

When the **start_qm.com**, **end_qm.com** and **tidy_qm.com** procedures are executed, a log file is written to the LogDirectory specified in the **failover.ini** configuration file. The names of the log files are **qmgrname_procedurename.log**. For example, for a queue manager name of **TESTQM** the **start_qm.com** command procedure produces a log file with name **testqm_start_qm.log**.

By default the failover monitor does not produce a log file, but a log file can be specified using the redirection parameter on the **runmqfm** command. Additional debug information can be written to the file by specifying the **-d** parameter on the **runmqfm** command.

Check whether any FDC files have been generated in **MQS_ROOT:[MQM.ERRORS]**.

Using MultiNet for OpenVMS with failover sets

To use Multinet for OpenVMS with failover sets, the cluster alias service must be enabled. The cluster alias service is enabled with command:

```
$ MULTINET CONFIGURE/SERVERS
SERVER-CONFIG> ENABLE CLUSTERALIAS
SERVER-CONFIG> EXIT
```

The template command files assume that there is only one cluster alias address, and that is used by the failover set. However, if there are other cluster alias addresses being used, the command procedures need to be modified so that the other addresses remain in the MULTINET_CLUSTER_IP_ALIASES logical name.

An example of using failover sets

The following is an example of configuring two nodes, BATMAN and ROBIN, in an OpenVMS cluster into a failover set for queue manager TESTQM. The failover set TCP/IP address is 10.20.30.40 and the TCP/IP listener port is 1414. The node BATMAN is nominated as the primary node and ROBIN the secondary node. Initially, the queue manager is started on BATMAN and if the queue manager fails, it is restarted on ROBIN. If the queue manager is running on ROBIN the queue manager is not failed back onto BATMAN when ROBIN is rebooted. If the node running the queue manager is shutdown, the queue manager is ended and the failover monitor halted. If the node is not running, the queue manager is not shut down and only the failover monitor is halted.

Customizing failover.template

The failover.template file is modified as follows and copied as mqsr_root:[mqm.qmgrs.testqm] failover.ini.

```

# FAILOVER.TEMPLATE
# Template for creating a FAILOVER.INI configuration file
# All lines beginning with a '#' are treated as comments
#
# OpenVMS Cluster Failover Set Configuration information
# -----
#
# The TCP/IP address used by the OpenVMS Cluster Failover Set
#
IpAddress=10.20.30.40
#
# The TCP/IP port number used by the WebSphere MQ Queue Manager
#
PortNumber=1414
#
# The timeout used by the EndCommand command procedure
#
TimeOut=30
#
# The command procedure used to start the Queue Manager
#
StartCommand=@sys$manager:start_qm
#
# The command procedure used to end the Queue Manager
#
EndCommand=@sys$manager:end_qm
#
# The command procedure used to tidy up on a node after a
# Queue Manager failure but the OpenVMS node did not fail
#
TidyCommand=@sys$manager:tidy_qm
#
# The directory in which the log files for the start, end and
# tidy commands are written
#
LogDirectory=mqs_root:[mqm.errors]
#
# The number of nodes in the OpenVMS Cluster Failover Set. The
# number of nodes defined below must agree with this number
#
NodeCount=2
#
# The Name of the OpenVMS node
#
NodeName=BATMAN
#
# The TCP/IP interface name for the node
#Interface=we0
#
# The priority of the node
#
Priority=1
#
# The Name of the OpenVMS node
#
NodeName=ROBIN
#
# The TCP/IP interface name for the node
#
Interface=we0
#
# The priority of the node
#
Priority=2

```

Figure 16. Failover.template for creating a FAILOVER.INI configuration file

Modification of failover set command procedures

The command procedures are copied from the template files. The only modifications are that the start of the listener in **start_qm.com** and the end of the listener in **end_qm.com** are uncommented (activated). If there are applications to be stopped and started, the appropriate commands could be added to the command procedures.

Example failover set start command procedure, **start_failover_set.com**

The **start_failover_set.com** command procedure is used to start the failover monitor on each node and conditionally start the queue manager. The procedure is called from the system startup after the **MQS_STARTUP.COM** command procedure has been executed. The procedure is passed two parameters: the queue manager name and the primary node name. In this case it is called as follows:

```
$@start_failover_set testqm batman
```

The **start_failover_set.com** command procedure starts the failover monitor and then uses the **-l** parameter on the **failover** command to find out the state of the failover set. Note that the failover monitor may not have completely started when the **failover** command is executed so the command is retried up to three times with a second between each attempt. Then if the node is the primary node and the queue manager is not started, it is started using the **-s** parameter of the **failover** command.

```

$on error then exit
@$sy$manager:mqs_symbols
$!
$! start_failover_set.com
$! -----
$! Command procedure to start a Failover Set Queue Manager during startup
$!
$! p1 = Queue Manager name
$! p2 = Primary Node name
$!
$! Check that the Queue Manager has been specified
$!
$if p1 .eqs ""
$then
$ Write sys$output "Queue Manager name omitted"
$ exit
$else
$ qmgr_name = p1
$endif
$!
$! Check that the primary node name has been specified
$!
$if p2 .eqs ""
$then
$ Write sys$output "Node name omitted"
$ exit
$else
$ primary_node = p2
$endif
$!
$! Get the node name of this node
$!
$this_node=f$getsyi("nodename")
$!
$! Start the Failover Monitor on this node
$!
$runmqfm -m 'qmgr_name'
$!
$! Check that the Failover Monitor has fully started
$! Wait up to 3 seconds
$!
$count = 0
$check_start:
$on error then continue
$!
$! Set the MQS$* symbols to the state of Failover Set
$! Wait up to 3 seconds
$!
$failover -m 'qmgr_name' -l
$!
$! If an error is returned wait a second and try again
$!
$if ( ($status/8) .and %xffff ) .ne. 0 then goto wait
$!
$! If this node is not listed as running a monitor wait a second and try again
$!
$if f$locate( this_node, mqs$monitor_nodes ) .ne. f$length( mqs$monitor_nodes )
$then
$ goto start_qm
$endif
$wait:
$on error then exit
$count = count + 1
$!
$! If we have waited 3 seconds display an error and exit
$!
$if count .ge. 3
$then
$ write sys$output "Failover Monitor not started"
$ exit
$else
$ wait 00:00:01
$ goto check_start

```

Example failover set end command procedure, `end_failover_set.com`

The `end_failover_set.com` command procedure is used to conditionally end the queue manager and then the failover monitor on each node. The procedure is called from the site-specific shutdown before the `MQS_SHUTDOWN.COM` command procedure has been executed. The procedure is passed one parameter, the queue manager name. In this case it is called as follows:

```
$@start_failover_set testqm
```

The `end_failover_set.com` command procedure obtains the failover set state using the `-l` parameter of the `failover` command. Then if the queue manager is running on this node, it is ended. Then the failover monitor is halted.


```

on error then exit
$@sys$manager:mqs_symbols
$!
$! end_failover_set.com
$! -----
$! Command procedure to end a Failover Set Queue Manager during shutdown
$!
$! p1 = Queue Manager name
$!
$! Check that the Queue Manager has been specified
$!
$if p1 .eqs ""
$then
$ Write sys$output "Queue Manager name omitted"
$ exit
$else
$ qmgr_name = p1
$endif
$!
$! Get the node name of this node
$!
$this_node=f$getsyi("nodename")
$!
$! Set the MQS$* symbols to the state of the Failover Set
$!
$failover -m 'qmgr_name' -l
$!
$! If an error then exit
$!
$if ( ($status/8) .and %xffff ) .ne. 0
$then
$ write sys$output "Error querying Failover Set"
$ exit
$endif
$!
$! If the Queue Manager is not running on this node then exit
$!
$ if mqs$qmgr_node .nes. this_node
$then
$ write sys$output "Queue Manager not running on this node"
$ goto halt_fm
$endif
$!
$! End the Queue Manager
$!
$failover -m gjtest -e
$halt_fm:
$!
$! Halt the Failover Monitor
$!
$failover -m gjtest -n 'this_node' -h

```

Figure 18. end_failover_set command procedure

Chapter 11. 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.

WebSphere MQ ensures that messages are not lost by maintaining recovery 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 WebSphere MQ in a planned way.
2. *Crash recovery*, when a failure stops WebSphere MQ.
3. *Media recovery*, to restore damaged objects.

In all cases, the recovery restores the queue manager to the state it was in when the queue manager stopped, except that any in-flight transactions are rolled back, removing from the queues any updates that were in-flight at the time the queue manager stopped. Recovery restores all persistent messages; nonpersistent messages can be lost during the process.

The rest of this chapter introduces the concepts of recovery and restart in more detail, and tells you how to recover if problems occur. It covers the following topics:

- “Making sure that messages are not lost (logging)”
- “Checkpointing – ensuring complete recovery” on page 160
- “Calculating the size of the log” on page 162
- “Managing logs” on page 164
- “Using the log for recovery” on page 166
- “Protecting WebSphere MQ log files” on page 169
- “Backing up and restoring WebSphere MQ” on page 169
- “Recovery scenarios” on page 173
- “Dumping the contents of the log using the `dmpmqlog` command” on page 175

Making sure that messages are not lost (logging)

WebSphere MQ records all significant changes to the data controlled by the queue manager in a recovery log.

This includes creating and deleting objects, persistent message updates, transaction states, changes to object attributes, and channel activities. The log contains the information you need to recover all updates to message queues by:

- Keeping records of queue manager changes
- Keeping records of queue updates for use by the restart process
- Enabling you to restore data after a hardware or software failure

What logs look like

A WebSphere MQ log consists of two components:

1. One or more files of log data.
2. A log control file

A file of log data is also known as a log extent.

There are a number of log files that contain the data being recorded. You can define the number and size (as explained in Chapter 6, “Configuring WebSphere MQ,” on page 71), or take the system default of three files.

In WebSphere MQ for Windows, each of the three files defaults to 1 MB. In WebSphere MQ for UNIX systems, each of the three files defaults to 4 MB.

When you create a queue manager, the number of log files you define is the number of *primary* log files allocated. If you do not specify a number, the default value is used.

In WebSphere MQ for HP OpenVMS systems, if you have not changed the log path, log files are created in the directory:

```
MQS_ROOT:[MQM.LOG.QmName.ACTIVE]
```

WebSphere MQ starts with these primary log files, but if the primary log space is not sufficient, it allocates *secondary* log files. It does this dynamically and removes them when the demand for log space reduces. By default, up to two secondary log files can be allocated. You can change this default allocation, as described in Chapter 6, “Configuring WebSphere MQ,” on page 71.

The log control file

The log control file contains the information needed to control the use of log files, such as their size and location, the name of the next available file, and so on.

Note: 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 handle. You may need to change the default log numbers and sizes to meet your requirements. For more information, see “Calculating the size of the log” on page 162.

Types of logging

In WebSphere MQ, 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. For now, you should know that WebSphere MQ uses secondary log files to extend the log capacity as necessary. The mechanism for releasing log files for reuse is described in “Checkpointing – ensuring complete recovery” on page 160.

Linear logging

Use linear logging if you want both restart recovery and media 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 in any log extent that has not been deleted

As disk space is finite, you might have to think about some form of archiving. It is an administrative task to manage your disk space for the log, reusing or extending the existing space as necessary.

The number of log files used with linear logging can be very large, depending on your message flow and the age of your queue manager. However, there are a number of files that are said to be *active*. Active files contain the log entries required to restart the queue manager. Collectively, active log files are known as the *active log*. The number of active log files is usually less than the number of primary log files as defined in the configuration files. (See “Calculating the size of the log” on page 162 for information about defining the number.)

The key event that controls whether a log file is termed active or not is a *checkpoint*. A WebSphere MQ checkpoint is a point of consistency between the recovery log and object files. A checkpoint determines the set of log files needed to perform restart recovery. Log files that are not active are not required for restart recovery, and are termed inactive. In some cases inactive log files are required for media recovery. (See “Checkpointing – ensuring complete recovery” on page 160 for further information about checkpointing.)

Inactive log files can be archived as they are not required for restart recovery. Inactive log files that are not required for media recovery can be considered as superfluous log files. You can delete superfluous log files if they are no longer of interest to your operation. Refer to “Managing logs” on page 164 for further information about the disposition of log files.

If a new checkpoint is recorded in the second, or later, primary log file, the first file can become 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 arises when, probably due to long-lived

transactions, it is not possible to free a file from the active pool because it might 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 are refused with an MQRC_RESOURCE_PROBLEM return code 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, WebSphere MQ uses checkpoints. A checkpoint is a point in time when the record described in the log is the same as the record in the queue. The checkpoint itself consists of the series of log records needed to restart the queue manager; for example, the state of all transactions (that is, units of work) active at the time of the checkpoint.

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

WebSphere MQ 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.

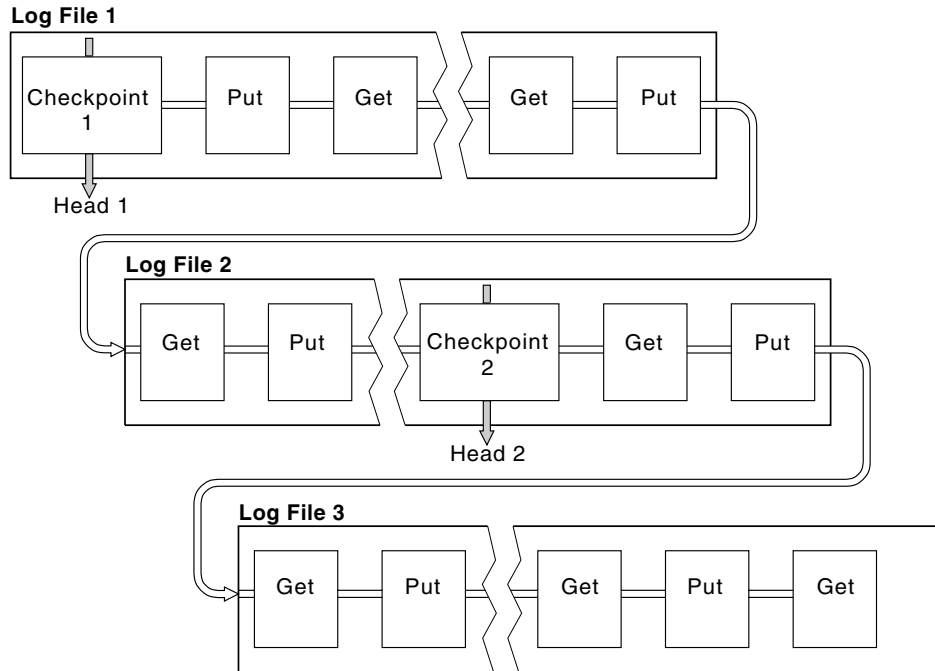


Figure 19. Checkpointing. For simplicity, only the ends of the log files are shown.

In Figure 19, all records before the latest checkpoint, checkpoint 2, are no longer needed by WebSphere MQ. 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.

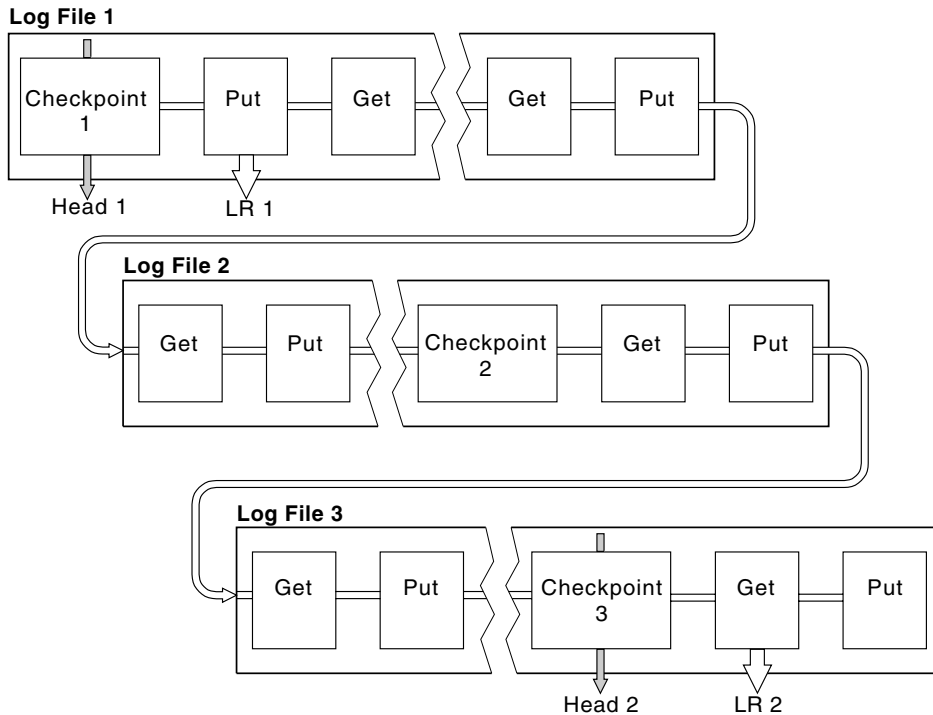


Figure 20. Checkpointing with a long-running transaction. For simplicity, only the ends of the log files are shown.

Figure 20 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 you are using linear logging, the log head is still moved down the active pool and the first file becomes inactive. A new primary file is formatted and added to the bottom of the pool in readiness for future logging activities.

Calculating the size of the log

After deciding whether the queue manager should use circular or linear logging, you need to estimate the size of the log that the queue manager needs. The size of the log is determined by the following log configuration parameters:

LogFilePages

The size of each primary and secondary log file in units of 4K pages

LogPrimaryFiles

The number of preallocated primary log files

LogSecondaryFiles

The number of secondary log files that can be created for use when the primary log files are full

Table 13 shows the amount of data the queue manager logs for various operations. Most queue manager operations need a minimal amount of log space. However, when a persistent message is put to a queue, **all** the message data must be written to the log to make it possible to recover the message. The size of the log depends, typically, on the number and size of the persistent messages the queue manager needs to handle.

Table 13. Log overhead sizes (all values are approximate)

Operation	Size
Put persistent message	750 bytes + message length If the message is large, it is divided into segments of 15700 bytes, each with a 300-byte overhead.
Get message	260 bytes
Syncpoint, commit	750 bytes
Syncpoint, rollback	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 260 000 bytes, each having a 300-byte overhead.
Checkpoint	750 bytes + 200 bytes for each active unit of work Additional data might be logged for any uncommitted puts or gets that have been buffered for performance reasons.

Note:

1. You can change the number of primary and secondary log files each time the queue manager starts.
2. You cannot change the log file size; you must determine it **before** creating the queue manager.
3. The number of primary log files and the log file size determine the amount of log space that is preallocated when the queue manager is created.
4. The total number of primary and secondary log files cannot exceed 63, which in the presence of long-running transactions, limits the maximum amount of log space available to the queue manager for restart recovery. The amount of log space the queue manager might need for media recovery does not share this limit.
5. When *circular* logging is being used, the queue manager reuses primary log space. This means that the queue manager's log can be smaller than the amount of data you have estimated that the queue manager needs to log. The queue manager allocates, up to a limit, a secondary log file when a log file becomes full, and the next primary log file in the sequence is not available.

6. Primary log files are made available for reuse during a checkpoint. The queue manager takes both the primary and secondary log space into consideration before taking a checkpoint because the amount of log space is running low. If you do not define more primary log files than secondary log files, the queue manager might allocate secondary log files before a checkpoint is taken. This makes the primary log files available for reuse.

Managing logs

Over time, some of the log records written become unnecessary for restarting the queue manager. If you are using circular logging, the queue manager reclaims freed space in the log files. This activity is transparent to the user and you do not usually see the amount of disk space used reduce because the space allocated is quickly reused.

Of the log records, only those written since the start of the last complete checkpoint, and those written by any active transactions, are needed to restart the queue manager. Thus, the log might 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 often enough 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 roll back 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 cannot 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 can then issue `MQCMIT`, which returns `MQRC_BACKED_OUT`, or `MQBACK` and start a new transaction. When the transaction consuming too much log space has been rolled back, its log space is released and the queue manager continues to operate normally.

If the log fills, message AMQ7463 is issued. In addition, if the log fills because a long-running transaction has prevented the space being released, message AMQ7465 is issued.

Finally, if records are being written to the log faster than the asynchronous housekeeping processes can handle them, message AMQ7466 is issued. If you see this message, 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 WebSphere MQ can run out of disk space only when it is creating a new file; it cannot run out of space when it is

writing a record to the log. WebSphere MQ 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 might be able to free some disk space. If you are using a linear log, there might be some inactive log files in the log directory, and you can copy these files to another drive or device. If you still run out of space, check that the configuration of the log in the queue manager configuration file is correct. You might be able to reduce the number of primary or secondary log files so that the log does not outgrow the available space. You cannot 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 when you configure your system (see “The LogDefaults stanza” on page 76 and “The Log stanza” on page 80). The amount of disk space used by the log does not increase beyond the configured size, including space for secondary files to be created when required.

If you are using a linear log, the log files are added continually as data is logged, and the amount of disk space used increases with time. If the rate of data being logged is high, disk space is consumed rapidly by new log files.

Over time, the older log files for a linear log are no longer needed to restart the queue manager or to perform media recovery of any damaged objects. The following are methods for determining which log files are still required:

Logger event messages

When enabled, logger event messages are generated when queue managers starts writing log records to a new log file. The contents of logger event messages specify the log files that are still required for queue manager restart, and media recovery. For more information on logger event messages, see the *Monitoring WebSphere MQ* manual.

Queue manager status

Executing the MQSC command, DISPLAY QMSTATUS, or the PCF command, Inquire Queue Manager Status, returns queue manager information, including details of the required log files. For more information on MQSC commands, see the *WebSphere MQ Script (MQSC) Command Reference* manual, and for information on PCF commands, see the *WebSphere MQ Programmable Command Formats and Administration Interface* manual.

Queue manager messages

Periodically, the queue manager issues a pair of messages to indicate which of the log files are needed:

- 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 for media recovery.

Only log files required for queue manager restart, active log files, need to be online. Inactive log files can be copied to an archive medium such as tape for disaster recovery, and removed from the log directory. Inactive log files that are not

required for media recovery can be considered as superfluous log files. You can delete superfluous log files if they are no longer of interest to your operation.

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 might wish to recover to avoid running out of disk space to hold all the required log files.

Messages AMQ7467 and AMQ7468 can also be issued at the time of running the **rcdmqimg** command. For more information about this command, see “**rcdmqimg** (record media image)” on page 248.

Log file location

When choosing a location for your log files, remember that operation is severely impacted if WebSphere MQ 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. 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, allow considerably more space; the space consumed by the log increases continuously as data is logged.

Ideally, place the log files on a separate disk drive from the queue manager data. This has benefits in terms of performance. It might also be possible to place the log files on multiple disk drives in a mirrored arrangement. This protects against failure of the drive containing the log. Without mirroring, you could be forced to go back to the last backup of your WebSphere MQ system.

Using the log for recovery

There are several ways that your data can be damaged. WebSphere MQ helps you to recover from:

- A damaged data object
- A power loss in the system
- A communications failure

This section looks at how the logs are used to recover from these problems.

Recovering from power loss or communications failures

WebSphere MQ can recover from both communications failures and loss of power. In addition, it can sometimes 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, you can usually restart the channels using the link that failed.

If you lose power, when the queue manager is restarted WebSphere MQ restores the queues to their committed state at the time of the failure. This ensures that no persistent messages are lost. Nonpersistent messages are discarded; they do not survive when WebSphere MQ stops abruptly.

Recovering damaged objects

There are ways in which a WebSphere MQ object can become unusable, for example because of 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 re-creates 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 re-create it. The information in the log required for media recovery of an object is called a *media image*. Media images can be recorded manually, using the **rcdmqimg** command, or automatically in some circumstances.

A media image is a sequence of log records containing an image of an object from which the object itself can be re-created.

The first log record required to re-create an object is known as its *media recovery record*; it is the start of the latest media image for the object. The media recovery record of each object is one of the pieces of information recorded during a checkpoint.

When an object is re-created 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 re-create 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 re-create the object. These records make up the object's first media image. Subsequently, at each shutdown, the queue manager records media images automatically as follows:

- Images of all process objects and queues that are not local
- Images of empty local queues

Media images can also be recorded manually using the **rcdmqimg** command, described in “**rcdmqimg** (record media image)” on page 248. This command writes a media image of the WebSphere MQ object. Once this has been done, only the logs that hold the media image, and all the logs created after this time, are needed

to re-create damaged objects. The benefit of doing this depends on such factors as the amount of free storage available, and the speed at which log files are created.

Recovering from media images

WebSphere MQ 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 when the queue manager last shutdown, any queue affected is also recovered automatically in order to complete the startup operation.

You must recover other objects manually, using the **rcrmqobj** command, which replays the records in the log to re-create the WebSphere MQ object. The object is re-created 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 re-create command was issued. If a WebSphere MQ object becomes damaged, the only valid actions that can be performed are either to delete it or to re-create it by this method. Nonpersistent messages **cannot** be recovered in this way.

See “rcrmqobj (recreate object)” on page 250 for further details of the **rcrmqobj** command.

The log file containing the media recovery record, and all subsequent log files, must be 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 want to re-create, you might have insufficient disk space to hold all the log files required to re-create an object.

Recovering damaged objects during start up

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, WebSphere MQ automatically tries to re-create the queue manager object from its media images. If the log method selected does not support media recovery, you can either restore a backup of the queue manager or delete the queue manager.

If any transactions were active when the queue manager stopped, the local queues containing the persistent, uncommitted messages put or got inside these transactions are also needed to start the queue manager successfully. If any of these local queues is found to be damaged, and the queue manager supports media recovery, it automatically tries to re-create them from their media images. If any of the queues cannot be recovered, WebSphere MQ 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 automatic only 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 can delete it or, if the queue manager is using a linear log, attempt to recover it from its media image using the `rcrmobj` command (see “`rcrmobj` (recreate object)” on page 250 for further details).

Protecting WebSphere MQ log files

Do not remove the active log files manually when a WebSphere MQ queue manager is running. If a user inadvertently deletes the log files that a queue manager needs to restart, WebSphere MQ **does not** issue any errors and continues to process data *including persistent messages*. The queue manager shuts down normally, but can fail to restart. Recovery of messages then becomes impossible.

Users with the authority to remove logs that are being used by an active queue manager also have authority to delete other important queue manager resources (such as queue files, the object catalog, and WebSphere MQ executables). They can therefore damage, perhaps through inexperience, a running or dormant queue manager in a way against which WebSphere MQ cannot protect itself.

Exercise caution when conferring super user or mqm authority.

Backing up and restoring WebSphere MQ

Periodically, you can take measures to protect queue managers against possible corruption caused by hardware failures. There are two ways of protecting a queue manager:

Backup the queue manager data

In the event of hardware failure, a queue manager can be forced to stop. If any queue manager log data is lost due to the hardware failure, the queue manager might be unable to restart. Through backing up queue manager data you may be able to recover some, or all, of the lost queue manager data.

In general, the more frequently you back up queue manager data, the less data you lose in the event of hardware failure resulting in loss of integrity in the recovery log.

To backup queue manager data, the queue manager must not be running.

For instructions of how to backup queue manager data, and how to restore queue manager data, see:

- “Backing up queue manager data” on page 170.
- “Restoring queue manager data” on page 170.

Using a backup queue manager

In the event of severe hardware failure, a queue manager can be unrecoverable. In this situation, if the unrecoverable queue manager has a dedicated backup queue manager, the backup queue manager can be activated in place of the unrecoverable queue manager. If it was updated

regularly, the backup queue manager log can contain log data up to, and including, the last complete log extent from the unrecoverable queue manager.

A backup queue manager can be updated while the existing queue manager is still running.

For instructions of how to create a backup queue manager, and how to activate a backup queue manager, see:

- “Creating a backup queue manager” on page 171.
- “Starting a backup queue manager” on page 172.

Backing up queue manager data

To take a backup copy of a queue manager’s data:

1. Ensure that the queue manager is not running. If you try to take a backup of a running queue manager, the backup might not be consistent because of updates in progress when the files were copied.

If possible, stop your queue manager in an orderly way. Try executing **endmqm -w** (a wait shutdown); only if that fails, use **endmqm -i** (an immediate shutdown).

2. Find the directories under which the queue manager places its data and its log files, using the information in the configuration files. For more information about this, see Chapter 6, “Configuring WebSphere MQ,” on page 71.

Note: You might have some difficulty in understanding the names that appear in the directory. The names are transformed to ensure that they are compatible with the platform on which you are using WebSphere MQ. For more information about name transformations, see “Understanding WebSphere MQ file names” on page 17.

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 files, especially the log control file and the configuration files (or equivalent Registry entries on Windows). Some of the directories might be empty, but you need them all to restore the backup at a later date, so save them too.

4. Preserve the ownerships of the files. For WebSphere MQ for UNIX systems, you can do this with the **tar** command.

Restoring queue manager data

To restore a backup of a queue manager’s data:

1. Ensure that the queue manager is not running.
2. Find the directories under which the queue manager places its data and its log files, using the information in the configuration files
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.
5. Update the configuration information files (or equivalent Registry entries on Windows).

Check the resulting directory structure to ensure that you have all the required directories.

See Appendix B, “Directory structure,” on page 275 for more information about WebSphere MQ directories and subdirectories.

Make sure that you have a log control file as well as the log files. Also check that the WebSphere MQ and queue manager configuration files are consistent so that WebSphere MQ can look in the correct places for the restored data.

If the data was backed up and restored correctly, the queue manager starts.

For circular logging, backup the queue manager data and log file directories at the same time as this should allow a consistent set of queue manager data and logs to be restored.

For linear logging, we recommend that you backup the queue manager data and log file directories at the same time. However, it is possible to restore only the queue manager data files if a corresponding complete sequence of log files is available.

Using a backup queue manager

An existing queue manager can have a dedicated backup queue manager. A backup queue manager is an inactive copy of the existing queue manager. If the existing queue manager becomes unrecoverable due to severe hardware failure, the backup queue manager can be brought online to replace the unrecoverable queue manager.

The existing queue manager log files must regularly be copied to the backup queue manager to ensure that the backup queue manager remains an effective method for disaster recovery. The existing queue manager does not need to be stopped for log files to be copied, however you should only copy a log file if the queue manager has finished writing to it. Because the existing queue manager log is continually updated, there is always a slight discrepancy between the existing queue manager log and the log data copied to the backup queue manager log. Regular updates to the backup queue manager minimizes the discrepancy between the two logs.

If a backup queue manager is required to be brought online it must be activated, and then started. The requirement to activate a backup queue manager before it is started is a preventative measure to protect against a backup queue manager being started accidentally. Once a backup queue manager is activated it can no longer be updated.

For information on how to create, update, and start a backup queue manager, see the following:

- “Creating a backup queue manager”
- “Updating a backup queue manager” on page 172
- “Starting a backup queue manager” on page 172

Creating a backup queue manager

You can only use a backup queue manager when using linear logging.

To create a backup queue manager for an existing queue manager, do the following:

1. Create a backup queue manager for the existing queue manager using the control command `crtmqm`. The backup queue manager requires the following:
 - To have the same attributes as the existing queue manager, for example the queue manager name, the logging type, and the log file size.
 - To be on the same platform as the existing queue manager.
 - To be at an equal, or higher, code level than the existing queue manager.
2. Take copies of all the existing queue manager's data and log file directories, including all subdirectories, as described in "Backing up queue manager data" on page 170.
3. Overwrite the backup queue manager's data and log file directories, including all subdirectories, with the copies taken from the existing queue manager.
4. Execute the following control command on the backup queue manager:

```
strmqm -r BackupQMName
```

This flags the queue manager as a backup queue manager within WebSphere MQ, and replays all the copied log extents to bring the backup queue manager in step with the existing queue manager.

Updating a backup queue manager

To ensure that a backup queue manager remains an effective method for disaster recovery it must be updated regularly. Regular updating lessens the discrepancy between the backup queue manager log, and the existing queue manager log. To update a backup queue manager, do the following:

1. Copy all the log extents that have been completed since the last update from the existing queue manager log directory to the backup queue manager log directory.
2. Execute the following control command on the backup queue manager:

```
strmqm -r BackupQMName
```

This replays all the copied log extents and brings the backup queue manager in step with the existing queue manager. Once complete, a message is generated which identifies all the log extents required for restart recovery, and all the log extents required for media recovery.

Warning: If you copy a **non-contiguous** set of logs to the backup queue manager log directory, only the logs up to the point where the first missing log is found are replayed.

Starting a backup queue manager

To substitute an unrecoverable queue manager with its backup queue manager, do the following:

1. Execute the following control command to activate the backup queue manager:

```
strmqm -a BackupQMName
```

The backup queue manager is activated. Now active, the backup queue manager can no longer be updated.

2. Execute the following control command to start the backup queue manager:

```
strmqm BackupQMName
```

WebSphere MQ regards this as restart recovery, and utilizes the log from the backup queue manager. During the last update to the backup queue manager replay occurred: therefore, only the active transactions from the last recorded checkpoint are rolled back.

When an unrecoverable queue manager is substituted for a backup queue manager some of the queue manager data from the unrecoverable queue manager can be lost. The amount of lost data is dependent on how recently the backup queue manager was last updated. The more recently the last update, the less queue manager data loss.

3. Restart all channels.

Check the resulting directory structure to ensure that you have all the required directories.

See Appendix B, "Directory structure," on page 275 for more information about WebSphere MQ directories and subdirectories.

Make sure that you have a log control file as well as the log files. Also check that the WebSphere MQ and queue manager configuration files are consistent so that WebSphere MQ can look in the correct places for the restored data.

If the data was backed up and restored correctly, the queue manager starts.

Note: Even though the queue manager data and log files are held in different directories, 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 may not start. If it does start, your data is likely to be corrupt.

Recovery scenarios

This section looks at a number of possible problems and indicates how to recover from them.

Disk drive failures

You might have 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 first check the directory structure for any damage and, if necessary, repair such damage. If you lose queue manager data, the queue manager directory structure might have been damaged. If so, re-create the directory tree manually before you restart the queue manager.

If damage has occurred to the queue manager data files, but not to the queue manager log files, then the queue manager is normally able to restart. If any damage has occurred to the queue manager log files, then it is likely that the queue manager is not able to restart.

Having checked for structural damage, there are a number of 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 restart the queue manager.
- **For linear logging with media recovery**, ensure that the directory structure is intact and restart the queue manager. If the queue manager restarts, check, using MQSC commands such as DISPLAY QUEUE, whether any other objects have been damaged. Recover those you find, using the **rccmqobj** command. For example:

```
rccmqobj -m QMgrName -t all *
```

where QMgrName is the queue manager being recovered. -t all * indicates that all damaged objects of any type are to be recovered. If only one or two objects have been reported as damaged, you can specify those objects by name and type here.

- **For linear logging with media recovery and with an undamaged log**, you might 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 things:

1. You must restore the checkpoint file 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.
2. You must have the oldest log file 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, restore a backup of both the queue manager data and the log, both of which were taken at the same time. This causes message integrity to be lost.

- **For circular logging**, if the queue manager log files are damaged, 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 re-creating the damaged objects.

If the queue manager log files are not damaged, the queue manager is normally able to restart. Following the restart you must identify all damaged objects, then delete and redefine them.

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**, manually delete the file containing the damaged object and restart the queue manager. (You can use the **dspmqls** command to determine the real, file-system name of the damaged object.) Media recovery of the damaged object is automatic.
- **For circular 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:

- For **linear logging**, re-create the object from its media image.
- For **circular logging**, we do not support re-creating a single object.

Automatic media recovery failure

If a local queue required for queue manager startup with a linear log is damaged, and the automatic media recovery fails, restore the last backup of the queue manager data and log and restart the queue manager.

Dumping the contents of the log using the `dmpmqlog` command

Use the `dmpmqlog` command to dump the contents of the queue manager log. By default all active log records are dumped, that is, the command starts dumping from the head of the log (usually the start of the last completed checkpoint).

The log can usually be dumped only when the queue manager is not running. Because the queue manager takes a checkpoint during shutdown, the active portion of the log usually contains a small number of log records. However, you can use the `dmpmqlog` command to dump more log records using one of the following options to change the start position of the dump:

- Start dumping from the *base* of the log. The base of the log is the first log record in the log file that contains the head of the log. The amount of additional data dumped in this case depends on where the head of the log is positioned in the log file. If it is near the start of the log file, only a small amount of additional data is dumped. If the head is near the end of the log file, significantly more data is dumped.
- Specify the start position of the dump as an individual log record. Each log record is identified by a unique *log sequence number (LSN)*. In the case of circular logging, this starting log record cannot be before the base of the log; this restriction does not apply to linear logs. You might need to reinstate inactive log files before running the command. You must specify a valid LSN, taken from previous `dmpmqlog` output, as the start position.

For example, with linear logging you can specify the `nextlsn` from your last `dmpmqlog` output. The `nextlsn` appears in Log File Header and indicates the LSN of the next log record to be written. Use this as a start position to format all log records written since the last time the log was dumped.

- For **linear logs only**, you can instruct `dmpmqlog` to start formatting log records from any given log file extent. In this case, `dmpmqlog` expects to find this log file, and each successive one, in the same directory as the active log files. This option does not apply to circular logs, where `dmpmqlog` cannot access log records prior to the base of the log.

The output from the `dmpmqlog` command is the Log File Header and a series of formatted log records. The queue manager uses several log records to record changes to its data.

Some of the information that is formatted is only of use internally. The following list includes the most useful log records:

Log File Header

Each log has a single log file header, which is always the first thing formatted

by the **dmpmqlog** command. It contains the following fields:

<i>logactive</i>	The number of primary log extents.
<i>loginactive</i>	The number of secondary log extents.
<i>logsize</i>	The number of 4 KB pages per extent.
<i>baselsn</i>	The first LSN in the log extent containing the head of the log.
<i>nextlsn</i>	The LSN of the next log record to be written.
<i>headlsn</i>	The LSN of the log record at the head of the log.
<i>tailsn</i>	The LSN identifying the tail position of the log.
<i>hflag1</i>	Whether the log is CIRCULAR or LOG RETAIN [®] (linear).
<i>HeadExtentID</i>	The log extent containing the head of the log.

Log Record Header

Each log record within the log has a fixed header containing the following information:

<i>LSN</i>	The log sequence number.
<i>LogRecdType</i>	The type of the log record.
<i>XTranid</i>	The transaction identifier associated with this log record (if any). A <i>TranType</i> of MQI indicates a WebSphere MQ-only transaction. A <i>TranType</i> of XA is involved with other resource managers. Updates involved within the same unit of work have the same <i>XTranid</i> .
<i>QueueName</i>	The queue associated with this log record (if any).
<i>Qid</i>	The unique internal identifier for the queue.
<i>PrevLSN</i>	The LSN of the previous log record within the same transaction (if any).

Start Queue Manager

This logs that the queue manager has started.

<i>StartDate</i>	The date that the queue manager started.
<i>StartTime</i>	The time that the queue manager started.

Stop Queue Manager

This logs that the queue manager has stopped.

<i>StopDate</i>	The date that the queue manager stopped.
<i>StopTime</i>	The time that the queue manager stopped.
<i>ForceFlag</i>	The type of shutdown used.

Start Checkpoint

This denotes the start of a queue manager checkpoint.

End Checkpoint

This denotes the end of a queue manager checkpoint.

<i>ChkPtLSN</i>	The LSN of the log record that started this checkpoint.
-----------------	---

Put Message

This logs a persistent message put to a queue. If the message was put under syncpoint, the log record header contains a non-null *XTranid*. The remainder of the record contains:

<i>SpcIndex</i>	An identifier for the message on the queue. It can be used to match the corresponding MQGET that was used to get this message from the queue. In this case a subsequent <i>Get Message</i> log record can be found containing the same <i>QueueName</i> and <i>SpcIndex</i> . At this point the <i>SpcIndex</i> identifier can be reused for a subsequent put message to that queue.
<i>Data</i>	Contained in the hex dump for this log record is various internal data followed by the Message Descriptor (eyecatcher MD) and the message data itself.

Put Part

Persistent messages that are too large for a single log record are logged as a single *Put Message* record followed by multiple *Put Part* log records.

<i>Data</i>	Continues the message data where the previous log record left off.
-------------	--

Get Message

Only gets of persistent messages are logged. If the message was got under syncpoint, the log record header contains a non-null *XTranid*. The remainder of the record contains:

<i>SpcIndex</i>	Identifies the message that was retrieved from the queue. The most recent <i>Put Message</i> log record containing the same <i>QueueName</i> and <i>SpcIndex</i> identifies the message that was retrieved.
<i>QPriority</i>	The priority of the message retrieved from the queue.

Start Transaction

Indicates the start of a new transaction. A *TranType* of MQI indicates a WebSphere MQ-only transaction. A *TranType* of XA indicates one that involves other resource managers. All updates made by this transaction have the same *XTranid*.

Prepare Transaction

Indicates that the queue manager is prepared to commit the updates associated with the specified *XTranid*. This log record is written as part of a two-phase commit involving other resource managers.

Commit Transaction

Indicates that the queue manager has committed all updates made by a transaction.

Rollback Transaction

This denotes the queue manager's intention to roll back a transaction.

End Transaction

This denotes the end of a rolled-back transaction.

Transaction Table

This record is written during syncpoint. It records the state of each transaction that has made persistent updates. For each transaction the following information is recorded:

<i>XTranid</i>	The transaction identifier.
<i>FirstLSN</i>	The LSN of the first log record associated with the transaction.
<i>LastLSN</i>	The LSN of the last log record associated with the transaction.

Transaction Participants

This log record is written by the XA Transaction Manager component of the queue manager. It records the external resource managers that are participating in transactions. For each participant the following is recorded:

<i>RMName</i>	The name of the resource manager.
<i>RMID</i>	The resource manager identifier. This is also logged in subsequent <i>Transaction Prepared</i> log records that record global transactions in which the resource manager is participating.
<i>SwitchFile</i>	The switch load file for this resource manager.
<i>XAOpenString</i>	The XA open string for this resource manager.
<i>XACloseString</i>	The XA close string for this resource manager.

Transaction Prepared

This log record is written by the XA Transaction Manager component of the queue manager. It indicates that the specified global transaction has been successfully prepared. Each of the participating resource managers is instructed to commit. The *RMID* of each prepared resource manager is recorded in the log record. If the queue manager itself is participating in the transaction a *Participant Entry* with an *RMID* of zero is present.

Transaction Forget

This log record is written by the XA Transaction Manager component of the queue manager. It follows the *Transaction Prepared* log record when the commit decision has been delivered to each participant.

Purge Queue

This logs the fact that all messages on a queue have been purged, for example, using the MQSC command CLEAR QUEUE.

Queue Attributes

This logs the initialization or change of the attributes of a queue.

Create Object

This logs the creation of a WebSphere MQ object.

<i>ObjName</i>	The name of the object that was created.
<i>UserId</i>	The user ID performing the creation.

Delete Object

This logs the deletion of a WebSphere MQ object.

<i>ObjName</i>	The name of the object that was deleted.
----------------	--

Chapter 12. Problem determination

This chapter suggests reasons for some of the problems you might experience using WebSphere MQ. You usually start with a symptom, or set of symptoms, and trace them back to their cause.

Problem determination is not problem solving. However, the process of problem determination often enables you to solve a problem. For example, if you find that the cause of the problem is an error in an application program, you can solve the problem by correcting the error.

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 WebSphere MQ code, contact your IBM Support Center. This chapter contains these sections:

- "Preliminary checks"
- "Looking at problems in more detail" on page 183
- "Application design considerations" on page 188
- "Error logs" on page 189
- "Dead-letter queues" on page 194
- "Configuration files and problem determination" on page 194
- "Using WebSphere MQ trace" on page 194
- "First-failure support technology (FFST)" on page 195
- "Problem determination with WebSphere MQ clients" on page 198
- "Error messages with clients" on page 198

Preliminary checks

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

The cause of your problem could be in:

- WebSphere MQ
- The network
- The application

The sections that follow raise some fundamental questions that you need to consider. As you work through the questions, make a note of anything that might be relevant to the problem. Even if your observations do not suggest a cause immediately, they could be useful later if you have to carry out a systematic problem determination exercise.

Has WebSphere MQ run successfully before?

If WebSphere MQ has not run successfully before, it may not have been set up correctly. See *WebSphere MQ for HP OpenVMS Quick Beginnings* to check that WebSphere MQ has been installed and set up correctly.

Are there any error messages?

WebSphere MQ uses error logs to capture messages concerning its own operation, 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 189 for information about the locations and contents of the error logs.

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 *WebSphere MQ 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 WebSphere MQ 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 WebSphere MQ 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 might have been made to either WebSphere MQ channel definitions or any underlying communications definitions required by your application.
- Do your applications deal with return codes that they might get as a result of any changes you have made?
- Have you changed any component of the operating system that could affect the operation of WebSphere MQ? For example, have you modified the Windows Registry.

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 invokes a rarely-used path in the program.
- Does the application check all return codes?
Has your WebSphere MQ 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 WebSphere MQ systems?
Could it be that there is something different about the way that this WebSphere MQ system is set up that 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, examine it carefully to see if you can find any errors.

Before you look at the code, and depending upon which programming language the code is written in, examine the output from the translator, or the compiler and linkage editor, to see if any errors have been reported.

If your application fails to translate, compile, or link-edit into the load library, it also fails to run if you attempt to invoke it. See the *WebSphere MQ Application Programming Guide* for information about building your application.

If the documentation shows that each of these steps was accomplished without error, consider the coding logic of the application. Do the symptoms of the problem indicate the function that is failing and, therefore, the piece of code in error? See “Common programming errors” for some examples of common errors that cause problems with WebSphere MQ applications.

Common programming errors

The errors in the following list illustrate the most common causes of problems encountered while running WebSphere MQ programs. Consider the possibility that the problem with your WebSphere MQ 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 might mean that WebSphere MQ cannot set up completion and reason codes for your application to process.
- Failing to check return codes from MQI requests.
- Passing variables with incorrect lengths specified.
- Passing parameters in the wrong order.
- Failing to initialize *MsgId* and *CorrelId* correctly.
- Failing to initialize *Encoding* and *CodedCharSetId* following MQRC_TRUNCATED_MSG_ACCEPTED.

Problems with commands

Be careful when including special characters, for example, back slash (\) and double quote (") characters, in descriptive text for some commands. If you use either of these characters in descriptive text, precede them with a \, that is, enter \\ or \" if you want \ or " in your text.

Does the problem affect specific parts of the network?

You might be able to identify specific parts of the network that are affected by the problem (remote queues, for example). If the link to a remote message queue manager is not working, the messages cannot flow to a remote queue.

Check that the connection between the two systems is available, and that the intercommunication component of WebSphere MQ has 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 WebSphere MQ 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 depends 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 WebSphere MQ 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 the way that processes can run independently of each other. For example, a program might issue an MQGET call without specifying a wait option before an earlier process has completed. An intermittent problem might 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 you have applied a service update to WebSphere MQ, 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 was applied correctly and completely?
- Does the problem still exist if WebSphere MQ is restored to the previous service level?
- If the installation was successful, check with the IBM Support Center for any maintenance package errors.
- If a maintenance package has been applied to any other program, consider the effect it might have on the way WebSphere MQ interfaces with it.

Looking at problems in more detail

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 WebSphere MQ library and in the libraries of other licensed programs.

If you have not yet found the cause, start to look at the problem in greater detail. 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?” on page 186
- “Are some of your queues failing?” on page 187
- “Does the problem affect only remote queues?” on page 187
- “Is your application or system running slowly?” on page 187

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.

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?
 - Has another application got exclusive access to the queue?
- Are you able to get any messages from the queue?
 - Do you need to take a syncpoint?

If messages are being put or retrieved within syncpoint, they are not available to other tasks until the unit of recovery has been committed.

- Is your wait interval long enough?
You can set the wait interval as an option for the MQGET call. 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 might 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 WebSphere MQ has been restarted, the message has been lost.
- Has another application got exclusive access to the queue?

If you cannot find anything wrong with the queue, and WebSphere MQ is running, check the process that you expected to put the message onto the queue for the following:

- Did the application start?
If it should have been triggered, check that the correct trigger options were specified.
- Did the application stop?
- Is a trigger monitor running?
- Was the trigger process defined correctly?
- Did the application complete correctly?
Look for evidence of an abnormal end in the job log.
- Did the application commit its changes, or were they backed out?

If multiple transactions are serving the queue, they can conflict with one another. For example, suppose one transaction issues an MQGET call with a buffer length of zero to find out the length of the message, and then issues a specific MQGET call specifying the *MsgId* of that message. However, in the meantime, another transaction issues a successful MQGET call for that message, so the first application receives a reason code of MQRC_NO_MSG_AVAILABLE. Applications that are expected to run in a multiple server environment must be designed to cope with this situation.

Consider that the message could have been received, but that your application failed to process it in some way. For example, did an error in the expected format of the message cause your program to reject it? If this is the case, refer to “Messages that contain unexpected or corrupted information.”

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:

- 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 might 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 appears corrupt to the other.

- Is an application sending messages to the wrong queue?

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

If your application uses 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 started; or should a different application have started?

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

Problems with incorrect output when using distributed queues

If your application uses distributed queues, consider the following points:

- Has WebSphere MQ 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 WebSphere MQ. Check that the connection between the two systems is active.

You can use the MQSC command PING against either the queue manager (PING QMGR) or the channel (PING CHANNEL) to verify that the link is operable.

- Is triggering set on in the sending system?
- Is the message for which you are waiting a reply message from a remote system?

Check that triggering is activated in the remote system.

- Is the queue already full?

If 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 *WebSphere MQ 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 can stop the distributed queuing component. See the *WebSphere MQ Intercommunications* manual 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 call is issued if the format is recognized as one of the built-in formats.

If the data format is not recognized for conversion, the data conversion exit is taken to allow you to perform the translation with your own routines.

Refer to the *WebSphere MQ Application Programming Guide* for further details of data conversion.

Have you failed to receive a response from a PCF command?

If you have issued a command but have not received a response, consider the following:

- Is the command server running?
Work with the **dspmqcsv** command to check the status of the command server.
 - If the response to this command indicates that the command server is not running, use the **strmqcsv** command to start it.
 - If the response to the command indicates that the SYSTEM.ADMIN.COMMAND.QUEUE is not enabled for MQGET requests, enable the queue for MQGET requests.
- Has a reply been sent to the dead-letter queue?
The dead-letter queue header structure contains a reason or feedback code describing the problem. See the *WebSphere MQ 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 189 for further information.
- Are the queues enabled for put and get operations?
- Is the *WaitInterval* long enough?
If your MQGET call has timed out, a completion code of MQCC_FAILED and a reason code of MQRC_NO_MSG_AVAILABLE are returned. (See the *WebSphere MQ Application Programming Reference* 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 receiving 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 WebSphere MQ system. First, try stopping individual queue managers to isolate a failing queue manager. If this does not reveal the problem, try stopping and restarting WebSphere MQ, responding to any messages that are produced in the error log.

If the problem still occurs after restart, contact your IBM Support Center for help.

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`, 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. The values might have changed; the queue might have been 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 that required channels have started, can be triggered, and any required initiators are running.
- Check that the programs that should be putting messages to the remote queues have not reported problems.
- If you use triggering to start the distributed queuing process, check that the transmission queue has triggering set on. Also, check that the trigger monitor is running.
- Check the error logs for messages indicating channel errors or problems.
- If necessary, start the channel manually. See the *WebSphere MQ Intercommunications* manual for information about starting channels.

Is your application or system running slowly?

If your application is running slowly, it might be in a loop or waiting for a resource that is not available.

This might also indicate a performance problem. Perhaps your system is operating near the limits of its capacity. This type of problem is probably worst at peak system load times, typically at mid-morning and mid-afternoon. (If your network extends across more than one time zone, peak system load might seem to occur at some other time.)

A performance problem might 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 appear to be a problem that only occurs when certain queues are accessed.

The following symptoms might indicate that WebSphere MQ is running slowly:

- Your system is slow to respond to MQSC 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 might lie with WebSphere MQ itself. If you suspect this, contact your IBM Support Center for help.

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 itself, but affect the performance of other tasks. Several problems specific to programs making WebSphere MQ calls are discussed in the following sections.

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

Effect of message length

The amount of data in a message can affect the performance of the application that processes the message. To achieve the best performance from your application, send only the essential data in a message. For example, in a request to debit a bank account, the only information that might need to be passed from the client to the server application is the account number and the amount of the debit.

Effect of message persistence

Persistent messages are usually logged. Logging messages reduces the performance of your application, so use persistent messages for essential data only. If the data in a message can be discarded if the queue manager stops or fails, use a nonpersistent message.

Searching for a particular message

The MQGET call usually retrieves the first message from a queue. If you use the message and correlation identifiers (*MsgId* and *CorrelId*) in the message descriptor

to specify a particular message, the queue manager has to search the queue until it finds that message. Using the MQGET call in this way affects the performance of your application.

Queues that contain messages of different lengths

If your application cannot use messages of a fixed length, grow and shrink the buffers dynamically to suit the typical message size. If the application issues an MQGET call that fails because the buffer is too small, the size of the message data is returned. Add code to your application so that the buffer is resized accordingly and the MQGET call is re-issued.

Note: if you do not set the *MaxMsgLength* attribute explicitly, it defaults to 4 MB, which might be very inefficient if this is used to influence the application buffer size.

Frequency of syncpoints

Programs that issue very large numbers of MQPUT or MQGET 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.

Error logs

WebSphere MQ uses a number of error logs to capture messages concerning its own operation of WebSphere MQ, 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, the error log is shown here:

```
MQS_ROOT: [MQM.QMGRS.QMgrName.ERRORS]AMQERR01.LOG
```

- If the queue manager is not available, the error log is shown here:

```
MQS_ROOT: [MQM.QMGRS.$SYSTEM.ERRORS]AMQERR01.LOG
```

- If an error has occurred with a client application, the error log is shown here:

```
MQS_ROOT: [MQM.ERRORS]AMQERR01.LOG
```

Error 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 256 KB. The files are placed in the errors subdirectory of each queue manager that you create.

As error messages are generated they are placed in AMQERR01. When AMQERR01 gets bigger than 256 KB it is copied to AMQERR02. Before the copy, AMQERR02 is copied to AMQERR03.LOG. The previous contents, if any, of AMQERR03 are discarded.

The latest error messages are thus always placed in AMQERR01, the other files being used to maintain a history of error messages.

All messages relating to channels are also placed in the appropriate queue manager's errors files unless the name of their queue manager is unknown or the queue manager is unavailable. When the queue manager name is unavailable or its name cannot be determined, channel-related messages are placed in the [MQM.QMGRS.\$SYSTEM.ERRORS] subdirectory.

To examine the contents of any error log file, use your usual OpenVMS editor.

Early errors

There are a number of special cases where the above error logs have not yet been established and an error occurs. WebSphere MQ attempts to record any such errors in an error log. The location of the log depends on how much of a queue manager has been established.

If, 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 WebSphere MQ 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 6, "Configuring WebSphere MQ," on page 71.

Example error log

This example shows part of a WebSphere MQ for HP OpenVMS error log:

```
...
06/29/00 09:41:39 AMQ7467: The oldest log file required to start queue
manager BKM1 is S0000000.LOG.
```

EXPLANATION: The log file S0000000.LOG contains the oldest log record required to restart the queue manager. Log records older than this may be required for media recovery.

ACTION: You can move log files older than S0000000.LOG 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.

----- --

06/29/00 09:41:39 AMQ7468: The oldest log file required to perform media recovery of queue manager BKM1 is S0000000.LOG.

EXPLANATION: The log file S0000000.LOG 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.
ACTION: You can move log files older than S0000000.LOG to an archive medium to release space in the log directory.

----- --
06/29/00 09:42:05 AMQ7467: The oldest log file required to start queue manager BKM1 is S0000000.LOG.

EXPLANATION: The log file S0000000.LOG contains the oldest log record required to restart the queue manager. Log records older than this may be required for media recovery.
ACTION: You can move log files older than S0000000.LOG 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.

----- --
06/29/00 09:42:05 AMQ7468: The oldest log file required to perform media recovery of queue manager BKM1 is S0000000.LOG.

EXPLANATION: The log file S0000000.LOG 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.
ACTION: You can move log files older than S0000000.LOG to an archive medium to release space in the log directory.

----- --
06/29/00 09:42:06 AMQ8003: WebSphere MQ queue manager started.

EXPLANATION: WebSphere MQ queue manager BKM1 started.
ACTION: None.

----- --
06/29/00 09:42:06 AMQ7467: The oldest log file required to start queue manager BKM1 is S0000000.LOG.

EXPLANATION: The log file S0000000.LOG contains the oldest log record required to restart the queue manager. Log records older than this may be required for media recovery.
ACTION: You can move log files older than S0000000.LOG 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.

----- --
06/29/00 09:42:06 AMQ7468: The oldest log file required to perform media recovery of queue manager BKM1 is S0000000.LOG.

EXPLANATION: The log file S0000000.LOG 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.
ACTION: You can move log files older than S0000000.LOG to an archive medium to release space in the log directory.

----- --
06/29/00 09:46:27 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.

----- --
06/29/00 09:46:43 AMQ7467: The oldest log file required to start queue manager BKM1 is S0000000.LOG.

EXPLANATION: The log file S0000000.LOG contains the oldest log record required to restart the queue manager. Log records older than this may be required for media recovery.

ACTION: You can move log files older than S0000000.LOG 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.

----- --
06/29/00 09:46:43 AMQ7468: The oldest log file required to perform media recovery of queue manager BKMI is S0000000.LOG.

EXPLANATION: The log file S0000000.LOG 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.

ACTION: You can move log files older than S0000000.LOG to an archive medium to release space in the log directory.

----- --
06/29/00 09:46:44 AMQ8004: WebSphere MQ queue manager ended.

EXPLANATION: WebSphere MQ queue manager BKMI ended.

ACTION: None.

----- --
06/29/00 09:46:59 AMQ7467: The oldest log file required to start queue manager BKMI is S0000000.LOG.

EXPLANATION: The log file S0000000.LOG contains the oldest log record required to restart the queue manager. Log records older than this may be required for media recovery.

ACTION: You can move log files older than S0000000.LOG 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.

----- --
06/29/00 09:47:00 AMQ7468: The oldest log file required to perform media recovery of queue manager BKMI is S0000000.LOG.

EXPLANATION: The log file S0000000.LOG 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.

ACTION: You can move log files older than S0000000.LOG to an archive medium to release space in the log directory.

----- --
06/29/00 09:47:08 AMQ7472: Object TEST1, type queue damaged.

EXPLANATION: Object TEST1, type queue 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.

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.

----- --
06/29/00 09:47:09 AMQ8003: WebSphere MQ queue manager started.

EXPLANATION: WebSphere MQ queue manager BKMI started.

ACTION: None.

----- --
06/29/00 09:47:09 AMQ7467: The oldest log file required to start queue manager BKMI is S0000000.LOG.

EXPLANATION: The log file S0000000.LOG contains the oldest log record required to restart the queue manager. Log records older than this may be required for media recovery.

ACTION: You can move log files older than S0000000.LOG 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.

----- --
06/29/00 09:47:10 AMQ7468: The oldest log file required to perform media recovery of queue manager BKM1 is S0000000.LOG.

EXPLANATION: The log file S0000000.LOG 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.

ACTION: You can move log files older than S0000000.LOG to an archive medium to release space in the log directory.

----- --
06/29/00 09:47:47 AMQ7081: Object TEST1, type queue recreated.

EXPLANATION: The object TEST1, type queue was recreated from its media image.

ACTION: None.

----- --
06/29/00 11:22:10 AMQ7467: The oldest log file required to start queue manager BKM1 is S0000000.LOG.

EXPLANATION: The log file S0000000.LOG contains the oldest log record required to restart the queue manager. Log records older than this may be required for media recovery.

ACTION: You can move log files older than S0000000.LOG 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.

----- --
06/29/00 11:22:10 AMQ7468: The oldest log file required to perform media recovery of queue manager BKM1 is S0000000.LOG.

EXPLANATION: The log file S0000000.LOG 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.

ACTION: You can move log files older than S0000000.LOG to an archive medium to release space in the log directory.

----- --
06/29/00 11:22:11 AMQ8004: WebSphere MQ queue manager ended.

EXPLANATION: WebSphere MQ queue manager BKM1 ended.

ACTION: None.

----- --
...

Operator messages

Operator messages identify normal errors, typically caused directly by users doing things like using parameters that are not valid on a command. Operator messages are national-language enabled, with message catalogs installed in standard locations.

These messages are written to the associated window, if any. In addition, some operator messages are written to the AMQERR01.LOG file in the queue manager directory, and others to the equivalent file in the system error log directory.

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 the MQSC command `DISPLAY QUEUE`. If the queue contains messages, use the provided browse sample application (`amqsbcbg`) 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 each message. See “Browsing queues” on page 41 for more information about running this sample and about the kind of output it produces.

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

Problems might occur if you do not associate a dead-letter queue with each queue manager. For more information about dead-letter queues, see Chapter 9, “The WebSphere MQ dead-letter queue handler,” on page 125.

Configuration files and problem determination

Configuration file errors typically prevent queue managers from being found, and result in *queue manager unavailable* errors. Ensure that the configuration files exist, and that the WebSphere MQ configuration file references the correct queue manager and log directories.

Using WebSphere MQ trace

WebSphere MQ for HP OpenVMS uses the following commands for the trace facility:

- **dspmqrtrc** – see “dspmqrtrc (Display WebSphere MQ formatted trace output)” on page 230
- **endmqrtrc** – see “endmqrtrc (End WebSphere MQ trace)” on page 237

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:

`AMQppppppppp.TRC`

where *ppppppppp* is the process identifier (PID) of the process producing the trace.

Note:

1. In WebSphere MQ for HP OpenVMS, the value of the process identifier is always eight characters long.

2. There is one trace file for each process running as part of the entity being traced.

Sample trace data

The following sample is an extract from an OpenVMS trace:

```

...
20:02:16.652000 538969462.1      component:24 pointer:8B9FA0
20:02:16.652000 538969462.1      -----} xcsFreeMem rc=OK
20:02:16.652000 538969462.1      This set has 1 connections left (including us)
20:02:16.652000 538969462.1      -----{ xcsFreeMem
20:02:16.652000 538969462.1      component:24 pointer:62720D0
20:02:16.652000 538969462.1      -----} xcsFreeMem rc=OK
20:02:16.652000 538969462.1      -----{ xcsReleaseThreadMutexSem
20:02:16.652000 538969462.1      -----} xcsReleaseThreadMutexSem rc=OK
20:02:16.652000 538969462.1      -----} xcsDisconnectSharedMemSet rc=OK
20:02:16.652000 538969462.1      -----{ xcsDisconnectSharedMemSet
20:02:16.653000 538969462.1      About to disconnect set 1::0::0-460
20:02:16.653000 538969462.1      -----{ xcsRequestThreadMutexSem
20:02:16.653000 538969462.1      -----} xcsRequestThreadMutexSem rc=OK
20:02:16.653000 538969462.1      -----{ xcsFreeMem
20:02:16.653000 538969462.1      component:24 pointer:7C1D60
20:02:16.653000 538969462.1      -----} xcsFreeMem rc=OK ...

```

Figure 21. Sample WebSphere MQ for HP OpenVMS trace

Note:

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 that is normally recorded in FFST logs is, on WebSphere MQ for HP 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 a WebSphere MQ internal error.

How to examine the FFSTs

The files are named AMQnnnnnnnn_mm.FDC, where:

nnnnnnnn

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.

```

+-----+
| WebSphere MQ First Failure Symptom Report |
|=====|
| Date/Time      :- Wednesday October 28 13:26:33 GMT 2009 |
| Host Name      :- IPF3 (OpenVMS V8.3-1H1) |
+-----+

```

```

PIDS          :- 5697175
LVLS          :- 6.0.1.2
Product Long Name :- WebSphere MQ for OpenVMS Itanium
Vendor        :- IBM
Probe Id      :- ZX005020
Application Name :- MQM
Component     :- zxcProcessChildren
SCCS Info     :- cmd/zmain/amqzymb0.c, 1.377.1.11
Line Number   :- 6272
Build Date    :- Mar  4 2009
CMVC level    :- p600-101-060504
Build Type    :- IKAP - (Production)
Userid        :- [400,400] (MQMQMQ)
Program Name  :- AMQZXMA0.EXE
Addressing mode :- 64-bit
Program Name  :- AMQZXMA0.EXE
Process       :- 538969253
Thread        :- 1
QueueManager  :- test28
ConnId(1) IPCC :- 2
ConnId(2) QM   :- 2
ConnId(3) QM-P :- 2
ConnId(4) App  :- 2
Major Errorcode :- zrcX_PROCESS_MISSING
Minor Errorcode :- OK
Probe Type     :- MSGAMQ5008
Probe Severity :- 2
Probe Description :- AMQ5008: An essential WebSphere MQ process 538969255
(zllCRIT) cannot be found and is assumed to be terminated.
FDCSequenceNumber :- 0
Arith1         :- 538969255 202004a7
Comment1       :- zllCRIT
VMS Errorcode  :- No message (00000000)

```

JPI Quota information:

```

=====
ASTCNT=4115/4120(99%) *      BIOCNT=2048/2048(100%) *
BYTCNT=29981504/29981504(4294967253%) *  DIOCNT=4096/4096(100%) *
ENQCNT=15917/16000(99%) *    FILCNT=1014/1024(99%) *
PAGFILCNT=9916864/10000000(99%) *      TQCNT=794/800(99%) *
FREPTCNT=2147483647          APTCNT=0
GPGCNT=30544                 PPGCNT=5328
VIRTPEAK=346112              DFWSCNT=16384
WSAUTH=32768                  WSAUTHEXT=784384
WSEXTENT=784384              WSPEAK=35872
WSQUOTA=32768                 WSSIZE=48768
CPULIM=0                       MAXDETACH=0
MAXJOBS=0                       JOBPRCNT=5
PAGEFLTS=2998                 PRCNT=5/300(1%) +
(*) - % resource remaining, (+) - % resource used

```

Privilege and rights information:

```

=====
CURPRIV=acct allspool altpri audit bugchk bypass cmexec cmkrnl detach
diagnose downgrade exquota group grpnam grprpv import log_io
mount netmbx oper pfnmap phy_io prmceb prmgbl prmbx
pswamp readall security setprv share shmem sysgbl syslck
sysnam sysprv tmpmbx upgrade volpro world
IMAGPRIV=bugchk prmgbl sysgbl world
AUTHPRIV=acct allspool altpri audit bugchk bypass cmexec cmkrnl detach
diagnose downgrade exquota group grpnam grprpv import log_io
mount netmbx oper pfnmap phy_io prmceb prmgbl prmbx
pswamp readall security setprv share shmem sysgbl syslck
sysnam sysprv tmpmbx upgrade volpro world
MAHESH                          INTERA
REMOTE                           SYS$NO
IMAGE_RIGHTS=

```

```
SYS$NODE_IPF3
```

```
SYI information:
```

```
=====
```

```
ACTIVE CPU=2/2(100%) +          CLUSTER NODES=1
FREE_GBLPAGES=3003952/3447272(87%) *  GBLPAGFIL=197084
FREE_GBLSECTS=251/1600(15%) *      MEMSIZE=262096
PAGEFILE_FREE=262600/262600(100%) *  PAGE_SIZE=8192
SWAPFILE_FREE=5512/5512(100%) *     MAXPROCESSCNT=695
PROCSECTCNT=128                   BALSETCNT=693
WSMAX=784384                       NPAGEDYN=17973248
NPAGEVIR=98287616                  PAGEDYN=10158080
VIRTUALPAGECNT=2147483647          LOCKIDTBL_MAX=1842328
PQL_DASTLM=24                       PQL_MASTLM=100
PQL_DBIOLM=32                       PQL_MBIOLM=100
PQL_DBYTLM=262144                   PQL_MBYTLM=128000
PQL_DCPULM=0                         PQL_MCPULM=0
PQL_DDIOLM=32                       PQL_MDIOLM=100
PQL_DFILLM=128                      PQL_MFILLM=100
PQL_DPGFLQUOTA=700000               PQL_MPGFLQUOTA=512000
PQL_DPRCLM=32                       PQL_MPRCLM=10
PQL_DTQELM=16                       PQL_MTQELM=0
PQL_DWSDEFAULT=16384                PQL_MWSDEFAULT=16384
PQL_DWSQUOTA=32768                  PQL_MWSQUOTA=32768
PQL_DWSEXTENT=784384                PQL_MWSEXTENT=784384
PQL_DENQLM=2048                     PQL_MENQLM=300
PQL_DJTQUOTA=8192                   PQL_MJTQUOTA=0
CLISYMTBL=750                       DEFMBXMSG=256
DEFMBXBUFQUO=1056                   CHANNELCNT=512
DLCKEXTRASTK=2560                   PIOPAGES=975
CTLPAGES=1056                       CTLIMGLIM=35
(*) - % resource remaining, (+) - % resource used
```

```
MQM Function Stack
ExecCtrlrMain
zxcProcessChildren
xcsFFST
```

```
MQM Trace History
---{ kill
Data: 0x00000000 0x00000000
---} kill rc=OK
--} xcsCheckProcess rc=OK
--{ zdmHealthCheck
---{ xcsCheckProcess
Data: 0x202004aa
----{ kill
.
.
.
.
```

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 Chapter 6, "Configuring WebSphere MQ," on page 71 for further details.

Problem determination with WebSphere MQ clients

An MQI client application receives MQRC_* reason codes in the same way as non-client MQI applications. However, there are additional reason codes for error conditions associated with clients. For example:

- Remote machine not responding
- Communications line error
- Invalid machine address

The most common time for errors to occur is when an application issues an MQCONN and receives the response MQRC_Q_MQR_NOT_AVAILABLE. An error message, written to the client log file, explains the cause of the error. Messages might also be logged at the server, depending on the nature of the failure.

Terminating clients

Even though a client has terminated, the process at the server can still hold its queues open. Normally, this is only 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.

OpenVMS clients

Error messages for OpenVMS systems clients are placed in the error logs on their respective WebSphere MQ server systems. Typically, these files appear in the MQS_ROOT:[MQM.ERRORS] directory on OpenVMS systems.

The names of the default files held in this directory are:

AMQERR01.LOG

For error messages.

AMQERR01.FDC

For First Failure Data Capture messages.

Chapter 13. How to use WebSphere MQ control commands

This chapter describes how to use the WebSphere MQ control commands. If you want to issue control commands, your user ID must be a member of the mqm group.

All commands in this chapter can be issued from an OpenVMS DCL prompt. Command names and their flags are not case sensitive: you can enter them in upper case, lower case, or a combination of upper case and lower case. However, parameters to control commands (such as queue names) can be case sensitive. See “Case sensitivity in control commands” on page 18 for more information.

Before using any control command, **mqm_startup** must have been run once since the last restart.

Names of WebSphere MQ objects

In general, the names of WebSphere MQ objects can have up to 48 characters. This rule applies to all the following objects:

- Queue managers
- Queues
- Process definitions
- Namelists
- Clusters
- Listeners
- Services
- Authentication information objects

The maximum length of channel, and client connection channel names is 20 characters.

The characters that can be used for all WebSphere MQ names are:

- Uppercase A–Z
- Lowercase a–z
- Numerics 0–9
- Period (.)
- Underscore (_)
- Forward slash (/) (see note 1)
- Percent sign (%) (see note 1)

Note:

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 can be enclosed in double quotation marks, but this is essential only if special characters are included in the name.

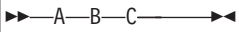
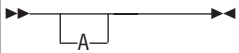

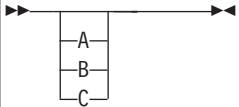
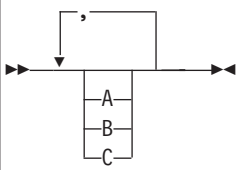
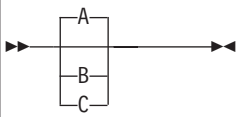
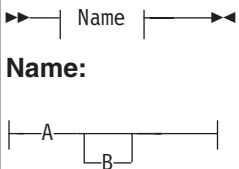
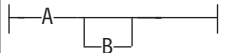
How to read syntax diagrams

This section describes the interpretation of 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:

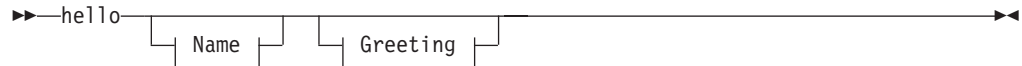
Table 14. How to read syntax diagrams

Convention	Meaning
	You must specify values A, B, and C. Required values are shown on the main line of a syntax diagram.
	You may specify value A. Optional values are shown below the main line of a syntax diagram.
	Values A, B, and C are alternatives, one of which you must specify.
	Values A, B, and C are alternatives, one of which you may specify.
	You may specify one or more of the values A, B, and C. Any required separator for multiple or repeated values (in this example, the comma (,)) is shown on the arrow.
	Values A, B, and C are alternatives, one of which you may specify. If you specify none of the values shown, the default A (the value shown above the main line) is used.
 Name: 	The syntax fragment Name is shown separately from the main syntax diagram.

Example syntax diagram

Here is an example syntax diagram that describes the **hello** command:

Hello Command



Name



Greeting



Notes:

- 1 You can code up to three names.

According to the syntax diagram, these are all valid versions of the **hello** command:

```
hello
hello name
hello name, name
hello name, name, name
hello, how are you?
hello name, how are you?
hello name, name, how are you?
hello name, name, name, how are you?
```

The space before the *name* value is significant: if you do not code *name* at all, you must still code the comma before how are you?.

Syntax help

You can obtain help for the syntax of any control command by entering the command followed by a question mark. WebSphere MQ responds by listing the syntax required for the selected command.

The syntax shows all the parameters and variables associated with the command. Different forms of parentheses are used to indicate whether a parameter is required. For example:

```
CmdName [-x OptParam ] ( -c | -b ) argument
```

where:

CmdName

Is the command name for which you have requested help.

[-x OptParam]

Square brackets enclose one or more optional parameters. Where square brackets enclose multiple parameters, you can select no more than one of them.

(-c | -b)

Brackets enclose multiple values, one of which you must select. In this example, you must select either flag c or flag b.

argument

A mandatory argument.

Examples

1. Result of entering endmqm ?

```
endmqm [-z] [-c | -w | -i | -p] QMgrName
```

2. Result of entering rcdmqimg ?

```
rcdmqimg [-z] [-m QMgrName] -t ObjType [GenericObjName]
```

Chapter 14. The control commands

This section provides reference information for each of the following WebSphere MQ control commands:

Command name	Purpose
crtmqcvx	Convert data
crtmqm	Create a local queue manager
dltmqm	Delete a queue manager
dmpmqaut	Dump authorizations to an object
dmpmqlog	Dump a log
dspmq	Display queue managers
dspmqaut	Display authorizations to an object
dspmqcsv	Display the status of a command server
dspmqfls	Display file names
dspmqrte	WebSphere MQ display route application
dspmqtrc	Display formatted trace output (UNIX systems only)
dspmqtrn	Display details of transactions
dspmqver	Display version number
endmqcsv	Stop the command server on a queue manager
endmqlsr	Stop the listener process on a queue manager
endmqm	Stop a local queue manager
endmqtrc	Stop tracing for an entity
mqftapp	Run the File Transfer Application
mqftrev	Receive file using the File Transfer Application (server)
mqftrevc	Receive file using the File Transfer Application (client)
mqftsnd	Send file using the File Transfer Application (server)
mqftsndc	Send file using the File Transfer Application (client)
rcdmqimg	Write an image of an object to the log
rcrmqobj	Recreate an object from their image in the log
rsvmqtrn	Commit or back out a transaction
runmqchi	Start a channel initiator process
runmqchl	Start a sender or requester channel
runmqdlq	Start the dead-letter queue handler
runmqlsr	Start a listener process
runmqsc	Issue MQSC commands to a queue manager
runmqtmc	Invoke the client trigger monitor
runmqtrm	Invoke a trigger monitor for a server
setmqaut	Change authorizations to an object
setmqprd	Enroll production license
strmqcsv	Start the command server for a queue manager

strmqm	Start a local queue manager
--------	-----------------------------

crtmqcvx (data conversion)

Purpose

Use the **crtmqcvx** command to create a fragment of code that performs data conversion on data type structures. The command generates a C function that can be used in an exit to convert C structures.

The command reads an input file containing structures to be converted, and writes an output file containing code fragments to convert those structures.

For information about using this command, see the *WebSphere MQ Application Programming Guide*.

Syntax

►► **crtmqcvx**—*SourceFile*—*TargetFile*—►►

Required parameters

SourceFile

The input file containing the C structures to convert.

TargetFile

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, the fragment uses macros supplied in the header file `amqsvmha.h`.

```

MQLONG Convertmy_structure(
    PMQBYTE *in_cursor,
    PMQBYTE *out_cursor,
    PMQBYTE in_lastbyte,
    PMQBYTE out_lastbyte,
    MQHCONN hConn,
    MQLONG opts,
    MQLONG MsgEncoding,
    MQLONG ReqEncoding,
    MQLONG MsgCCSID,
    MQLONG ReqCCSID,
    MQLONG CompCode,
    MQLONG Reason)
{
    MQLONG ReturnCode = MQRC_NONE;

    ConvertLong(1); /* code */

    AlignLong();
    ConvertLong(1); /* value */

Fail:
    return(ReturnCode);
}

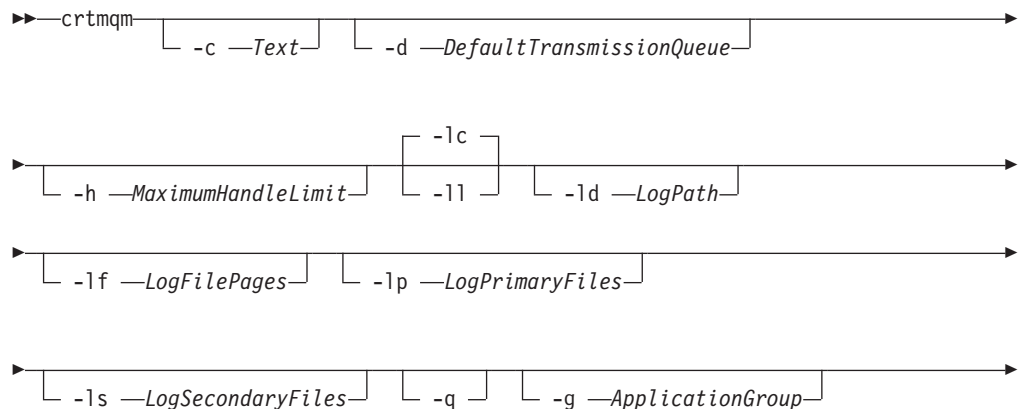
```

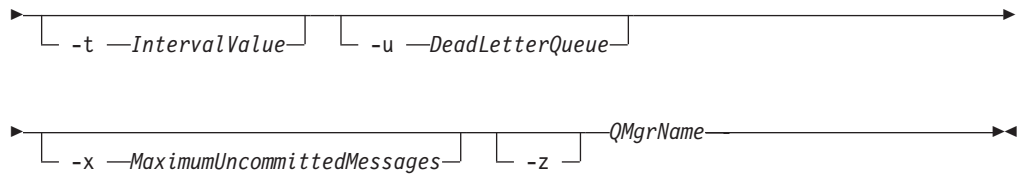
crtmqm (create queue manager)

Purpose

Use the **crtmqm** command to create a local queue manager and define the default and system objects. The objects created by **crtmqm** are listed in Appendix A, “System and default objects,” on page 273. When a queue manager has been created, use the **strmqm** command to start it.

Syntax





Required parameters

QMgrName

The name of the queue manager to create. The name can contain up to 48 characters. This must be the last item in the command.

Optional parameters

-c *Text*

Descriptive text for this queue manager. You can use up to 64 characters; the default is all blanks.

If you include special characters, enclose the description in double quotes. The maximum number of characters is reduced if the system is using a double-byte character set (DBCS).

-d *DefaultTransmissionQueue*

The name of the local transmission queue where remote messages are put if a transmission queue is not explicitly defined for their destination. There is no default.

-h *MaximumHandleLimit*

The maximum number of handles that any one application can have open at the same time.

Specify a value in the range 1 through 999 999 999. The default value is 256.

The next six parameter descriptions relate to logging, which is described in “Using the log for recovery” on page 166.

Note: Choose the logging arrangements with care, because some cannot be changed once they are committed.

-lc Use circular logging. This is the default logging method.

-ll Use linear logging.

-ld *LogPath*

The directory used to hold log files.

In WebSphere MQ for Windows, the default is `C:\Program Files\IBM\WebSphere MQ\log` (assuming that C is your data drive).

In WebSphere MQ for UNIX systems, the default is `/var/mqm/log`.

User ID `mqm` and group `mqm` must have full authorities to the log files. If you change the locations of these files, you must give these authorities yourself.

This occurs automatically if the log files are in their default locations.

-lf *LogFilePages*

The log data is held in a series of files called log files. The log file size is specified in units of 4 KB pages.

In WebSphere MQ for UNIX systems, the default number of log file pages is 1024, giving a log file size of 4 MB. The minimum number of log file pages is 64 and the maximum is 65 535.

In WebSphere MQ for Windows, the default number of log file pages is 256, giving a log file size of 1 MB. The minimum number of log file pages is 32 and the maximum is 65 535.

Note: The size of the log files specified during queue manager creation cannot be changed for a queue manager.

-lp *LogPrimaryFiles*

The log files allocated when the queue manager is created.

The minimum number of primary log files you can have is 2 and the maximum is 254 on Windows, or 510 on UNIX systems. The default is 3.

The total number of primary and secondary log files must not exceed 255 on Windows, or 511 on UNIX systems, and must not be less than 3.

Operating system limits can reduce the maximum possible log size.

The value is examined when the queue manager is created or started. You can change it after the queue manager has been created. However, a change in the value is not effective until the queue manager is restarted, and the effect might not be immediate.

For more information on primary log files, see “What logs look like” on page 158.

To calculate the size of the primary log files, see “Calculating the size of the log” on page 162.

-ls *LogSecondaryFiles*

The log files allocated when the primary files are exhausted.

The minimum number of secondary log files is 1 and the maximum is 253 on Windows, or 509 on UNIX systems. The default number is 2.

The total number of primary and secondary log files must not exceed 255 on Windows, or 511 on UNIX systems, and must not be less than 3.

Operating system limits can reduce the maximum possible log size.

The value is examined when the queue manager is started. You can change this value, but changes do not become effective until the queue manager is restarted, and even then the effect might not be immediate.

For more information on the use of secondary log files, see “What logs look like” on page 158.

To calculate the size of the secondary log files, see “Calculating the size of the log” on page 162.

-q Makes this queue manager the default queue manager. The new queue manager replaces any existing default queue manager.

If you accidentally use this flag and want to revert to an existing queue manager as the default queue manager, change the default queue manager as described in “Making an existing queue manager the default” on page 25.

-g *ApplicationGroup*

The name of the group containing members allowed to:

- Run MQI applications

- Update all IPCC resources
- Change the contents of some queue manager directories

This option applies only to WebSphere MQ for AIX, Solaris, HP-UX, and Linux®.

The default value is `-g all`, which allows unrestricted access.

The `-g ApplicationGroup` value is recorded in the queue manager configuration file, `qm.ini`.

The `mqm` user ID and the user executing the command **must** belong to the specified `ApplicationGroup`.

-t *IntervalValue*

The trigger time interval in milliseconds for all queues controlled by this queue manager. This value specifies the time after receiving 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 might choose to see all trigger events that happen; if so, set a low or zero value in this field.

Specify a value in the range 0 through 999 999 999. The default is 999 999 999 milliseconds, a time of more than 11 days. Allowing the default to be used effectively means that triggering is disabled after the first trigger message. However, an application can enable triggering again by servicing the queue using a command to alter the queue to reset the trigger attribute.

-u *DeadLetterQueue*

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 is no dead-letter queue.

-x *MaximumUncommittedMessages*

The maximum number of uncommitted messages under any one syncpoint. That is, the sum of:

- The number of messages that can be retrieved from queues
- The number of messages that can be put on queues
- Any trigger messages generated within this unit of work

This limit does not apply to messages that are retrieved or put outside a syncpoint.

Specify a value in the range 1 through 999 999 999. The default value is 10 000 uncommitted messages.

-z Suppresses error messages.

This flag is used within WebSphere MQ to suppress unwanted error messages. Because using this flag can result in loss of information, do not use it when entering commands on a command line.

Return codes

0 Queue manager created

8	Queue manager already exists
49	Queue manager stopping
69	Storage not available
70	Queue space not available
71	Unexpected error
72	Queue manager name error
100	Log location invalid
111	Queue manager created. However, there was a problem processing the default queue manager definition in the product configuration file. The default queue manager specification might be incorrect.
115	Invalid log size
119	Permission denied (Windows only)

Examples

1. This command creates a default queue manager called `Paint.queue.manager`, with a description of `Paint shop`, and creates the system and default objects. It also specifies that linear logging is to be used:

```
crtmqm -c "Paint shop" -ll -q Paint.queue.manager
```

2. This command creates a default queue manager called `Paint.queue.manager`, creates the system and default objects, and requests two primary and three secondary log files:

```
crtmqm -c "Paint shop" -ll -lp 2 -ls 3 -q Paint.queue.manager
```

3. This command creates a queue manager called `travel`, creates the system and default objects, sets the trigger interval to 5000 milliseconds (or 5 seconds), and specifies `SYSTEM.DEAD.LETTER.QUEUE` as its dead-letter queue.

```
crtmqm -t 5000 -u SYSTEM.DEAD.LETTER.QUEUE travel
```

Related commands

<code>strmqm</code>	Start queue manager
<code>endmqm</code>	End queue manager
<code>dltmqm</code>	Delete queue manager

dltmqm (delete queue manager)

Purpose

Use the **dltmqm** command to delete a specified queue manager and all objects associated with it. Before you can delete a queue manager you must end it using the **endmqm** command.

In WebSphere MQ for Windows, it is an error to delete a queue manager when queue manager files are open. If you get this error, close the files and reissue the command.

Syntax

```

>> dltmqm [ -z ] QMgrName <<<

```

Required parameters

QMgrName

The name of the queue manager to delete.

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
24	A process that was using the previous instance of the queue manager has not yet disconnected.
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 might be incorrect.
119	Permission denied (Windows only)

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

<code>crtmqm</code>	Create queue manager
<code>strmqm</code>	Start queue manager
<code>endmqm</code>	End queue manager

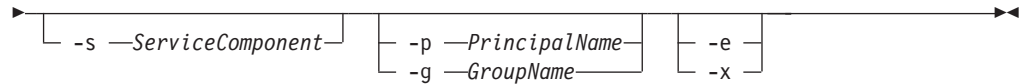
dmpmqaut (dump authority)

Purpose

Use the **dmpmqaut** command to dump the current authorizations to a specified object.

Syntax

```
▶▶ dmpmqaut [-m QMgrName] [-n Profile] [-t ObjectType]
```

Optional parameters

-m *QMGrName*

Dump authority records only for the queue manager specified. If you omit this parameter, only authority records for the default queue manager are dumped.

-n *Profile*

The name of the profile for which to dump authorizations.

-l Dump only the profile name and type. Use this option to generate a *terse* list of all defined profile names and types.

-t *ObjectType*

The type of object for which to dump authorizations. Possible values are:

authinfo	Authentication information object, for use with Secure Sockets Layer (SSL) channel security
channel or chl	A channel
clntconn or clcn	A client connection channel
listener or lstr	A listener
namelist or nl	A namelist
process or prcs	A process
queue or q	A queue or queues matching the object name parameter
qmgr	A queue manager
service or srvc	A service

-s *ServiceComponent*

If installable authorization services are supported, specifies the name of the authorization service for which to dump authorizations. This parameter is optional; if you omit it, the authorization inquiry is made to the first installable component for the service.

-p *PrincipalName*

This parameter applies to WebSphere MQ for Windows only; UNIX systems keep only group authority records.

The name of a user for whom to dump authorizations to the specified object. The name of the principal can optionally include a domain name, specified in the following format:

userid@domain

For more information about including domain names on the name of a principal, see Chapter 7, “WebSphere MQ security,” on page 91.

-g *GroupName*

The name of the user group for which to dump authorizations. You can specify only one name, which must be the name of an existing user group. On Windows systems, you can use only local groups.

-e Display all profiles used to calculate the cumulative authority that the entity has to the object specified in **-n** *Profile*. The variable *Profile* must not contain any wildcard characters.

The following parameters must also be specified:

- `-m QMgrName`
- `-n Profile`
- `-t ObjectType`

and either `-p PrincipalName`, or `-g GroupName`.

- `-x` Display all profiles with exactly the same name as specified in `-n Profile`. This option does not apply to the **QMGR** object, so a dump request of the form `dmpmqaut -m QM -t QMGR ... -x` is not valid.

Examples

The following examples show the use of `dmpmqaut` to dump authority records for generic profiles:

1. This example dumps all authority records with a profile that matches queue `a.b.c` for principal `user1`.

```
dmpmqaut -m qm1 -n a.b.c -t q -p user1
```

The resulting dump would look something like this:

```
profile:    a.b.*
object type: queue
entity:     user1
type:       principal
authority:  get, browse, put, inq
```

Note: UNIX users cannot use the `-p` option; they must use `-g` `groupname` instead.

2. This example dumps all authority records with a profile that matches queue `a.b.c`.

```
dmpmqaut -m qmgr1 -n a.b.c -t q
```

The resulting dump would look something like this:

```
profile:    a.b.c
object type: queue
entity:     Administrator
type:       principal
authority:  all
-----
profile:    a.b.*
object type: queue
entity:     user1
type:       principal
authority:  get, browse, put, inq
-----
profile:    a.**
object type: queue
entity:     group1
type:       group
authority:  get
```

3. This example dumps all authority records for profile `a.b.*`, of type `queue`.

```
dmpmqaut -m qmgr1 -n a.b.* -t q
```

The resulting dump would look something like this:

```

profile:    a.b.*
object type: queue
entity:     user1
type:       principal
authority:  get, browse, put, inq

```

4. This example dumps all authority records for queue manager qmX.

```
dmpmqaut -m qmX
```

The resulting dump would look something like this:

```

profile:    q1
object type: queue
entity:     Administrator
type:       principal
authority:  all
-----
profile:    q*
object type: queue
entity:     user1
type:       principal
authority:  get, browse
-----
profile:    name.*
object type: namelist
entity:     user2
type:       principal
authority:  get
-----
profile:    pr1
object type: process
entity:     group1
type:       group
authority:  get

```

5. This example dumps all profile names and object types for queue manager qmX.

```
dmpmqaut -m qmX -l
```

The resulting dump would look something like this:

```

profile: q1, type: queue
profile: q*, type: queue
profile: name.*, type: namelist
profile: pr1, type: process

```

Note:

1. For WebSphere MQ for Windows only, all principals displayed include domain information, for example:

```

profile:    a.b.*
object type: queue
entity:     user1@domain1
type:       principal
authority:  get, browse, put, inq

```

2. Each class of object has authority records for each group or principal. These records have the profile name @CLASS and track the crt (create) authority common to all objects of that class. If the crt authority for any object of that class is changed then this record is updated. For example:

```

profile:    @class
object type: queue
entity:     test
entity type: principal
authority:  crt

```

This shows that members of the group test have crt authority to the class queue.

3. For WebSphere MQ for Windows only, members of the “Administrators” group are by default given full authority. This authority, however, is given automatically by the OAM, and is not defined by the authority records. The **dmpmqaut** command displays authority defined only by the authority records. Unless an authority record has been explicitly defined, therefore, running the **dmpmqaut** command against the “Administrators” group displays no authority record for that group.

Related commands

dspmqaout	Display authority
setmqaut	Set or reset authority

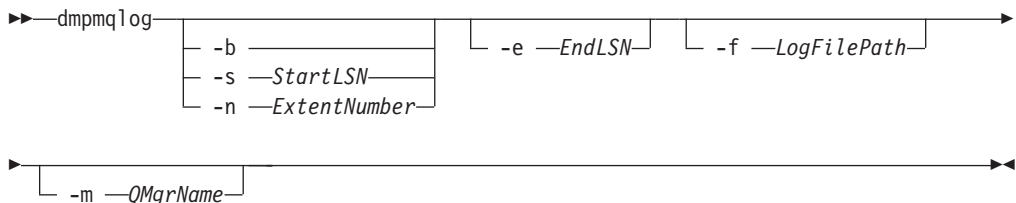
dmpmqlog (dump log)

Purpose

Use the **dmpmqlog** command to dump a formatted version of the WebSphere MQ system log.

The log to be dumped must have been created on the same type of operating system as that being used to issue the command.

Syntax



Optional parameters

Dump start point

Use one of the following parameters to specify the log sequence number (LSN) at which the dump should start. If you omit this, dumping starts by default from the LSN of the first record in the active portion of the log.

-b Start dumping from the base LSN. The base LSN identifies the start of the log extent that contains the start of the active portion of the log.

-s *StartLSN*

Start dumping from the specified LSN. The LSN is specified in the format *nnnn:nnnn:nnnn:nnnn*.

If you are using a circular log, the LSN value must be equal to or greater than the base LSN value of the log.

-n *ExtentNumber*

Start dumping from the specified extent number. The extent number must be in the range 0–9 999 999.

This parameter is valid only for queue managers using linear logging.

-e EndLSN

End dumping at the specified LSN. The LSN is specified in the format nnnn:nnnn:nnnn:nnnn.

-f LogFilePath

The absolute (rather than relative) directory path name to the log files. The specified directory must contain the log header file (amqh1ctl.lfh) and a subdirectory called active. The active subdirectory must contain the log files. By default, log files are assumed to be in the directories specified in the WebSphere MQ configuration information. If you use this option, queue names associated with queue identifiers are shown in the dump only if you use the -m option to name a queue manager name that has the object catalog file in its directory path.

On a system that supports long file names this file is called qmqmobjcat and, to map the queue identifiers to queue names, it must be the file used when the log files were created. For example, for a queue manager named qm1, the object catalog file is located in the directory ..\qmgrs\qm1\qmanager\. To achieve this mapping, you might need to create a temporary queue manager, for example named tmpq, replace its object catalog with the one associated with the specific log files, and then start dmpmqlog, specifying -m tmpq and -f with the absolute directory path name to the log files.

-m QMgrName

The name of the queue manager. If you omit this parameter, the name of the default queue manager is used.

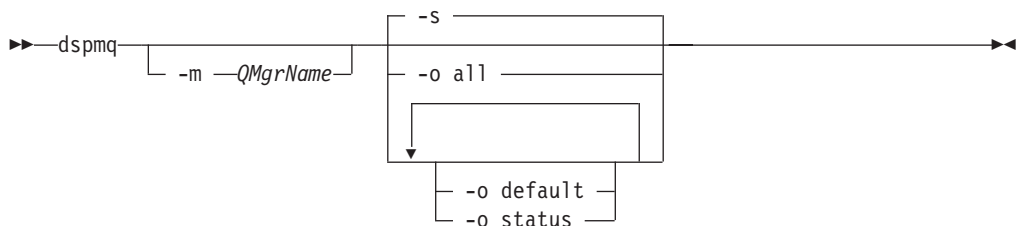
The queue manager must not be running when the **dmpmqlog** command is issued. Similarly, the queue manager must not be started while **dmpmqlog** is running.

dspmq (display queue managers)

Purpose

Use the **dspmq** command to display names and details of the queue managers on a system.

Syntax



Required parameters

None

Optional parameters

-m *QMgrName*

The queue manager for which to display details. If you give no name, all queue manager names are displayed.

-s Displays the operational status of the queue managers. This is the default status setting.

The parameter *-o status* is equivalent to *-s*.

-o *all*

Displays the operational status of the queue managers, and whether any are the default queue manager.

-o *default*

Displays whether any of the queue managers are the default queue manager.

-o *status*

Displays the operational status of the queue managers.

Queue Manager States

The following is a list of the different states a queue manager can be in:

Starting
Running
Quiescing
Ending immediately
Ending preemptively
Ended normally
Ended immediately
Ended unexpectedly
Ended preemptively

Return codes

0	Command completed normally
36	Invalid arguments supplied
71	Unexpected error
72	Queue manager name error

dspmqaout (display authority)

Purpose

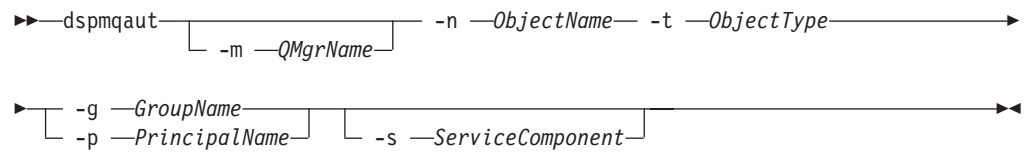
Use the **dspmqaout** command to display the current authorizations to a specified object.

If a user ID is a member of more than one group, this command displays the combined authorizations of all the groups.

Only one group or principal can be specified.

For more information about authorization service components, see “The Service stanza” on page 79 and “The ServiceComponent stanza” on page 80.

Syntax



Required parameters

-n *ObjectName*

The name of the object on which to make the inquiry.

This parameter is required, unless you are displaying the authorizations of a queue manager, in which case you *must not* include it and instead specify the queue manager name using the *-m* parameter.

-t *ObjectType*

The type of object on which to make the inquiry. Possible values are:

authinfo	Authentication information object, for use with Secure Sockets Layer (SSL) channel security
channel or chl	A channel
clntconn or clcn	A client connection channel
listener or lstr	A Listener
namelist or nl	A namelist
process or prcs	A process
queue or q	A queue or queues matching the object name parameter
qmgr	A queue manager
service or srvc	A service

Optional parameters

-m *QMgrName*

The name of the queue manager on which to make the inquiry. This parameter is optional if you are displaying the authorizations of your default queue manager.

-g *GroupName*

The name of the user group on which to make the inquiry. You can specify only one name, which must be the name of an existing user group. On Windows systems, you can use only local groups.

-p *PrincipalName*

The name of a user for whom to display authorizations to the specified object.

For WebSphere MQ for Windows only, the name of the principal can optionally include a domain name, specified in the following format:

```
userid@domain
```

For more information about including domain names on the name of a principal, see Chapter 7, “WebSphere MQ security,” on page 91.

-s *ServiceComponent*

If installable authorization services are supported, specifies the name of the

authorization service to which the authorizations apply. This parameter is optional; if you omit it, the authorization inquiry is made to the first installable component for the service.

Returned parameters

Returns an authorization list, which can contain none, one, or more authorization values. Each authorization value returned means that any user ID in the specified group or principal has the authority to perform the operation defined by that value.

Table 15 shows the authorities that can be given to the different object types.

Table 15. Specifying authorities for different object types

Authority	Queue	Process	Queue manager	Namelist	Auth info	Clntconn	Channel	Listener	Service
all	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
alladm	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
allmqi	Yes	Yes	Yes	Yes	Yes	No	No	No	No
none	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
altusr	No	No	Yes	No	No	No	No	No	No
browse	Yes	No	No	No	No	No	No	No	No
chg	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
clr	Yes	No	No	No	No	No	No	No	No
connect	No	No	Yes	No	No	No	No	No	No
crt	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
ctrl	No	No	No	No	No	No	Yes	Yes	Yes
ctrlx	No	No	No	No	No	No	Yes	No	No
dlt	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
dsp	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
get	Yes	No	No	No	No	No	No	No	No
put	Yes	No	No	No	No	No	No	No	No
inq	Yes	Yes	Yes	Yes	Yes	No	No	No	No
passall	Yes	No	No	No	No	No	No	No	No
passid	Yes	No	No	No	No	No	No	No	No
set	Yes	Yes	Yes	No	No	No	No	No	No
setall	Yes	No	Yes	No	No	No	No	No	No
setid	Yes	No	Yes	No	No	No	No	No	No

The following list defines the authorizations associated with each value:

all Use all operations relevant to the object.
alladm Perform all administration operations relevant to the object.
allmqi Use all MQI calls relevant to the object.
altusr Specify an alternate user ID on an MQI call.
browse Retrieve a message from a queue by issuing an **MQGET** call with the **BROWSE** option.

chg	Change the attributes of the specified object, using the appropriate command set.
clr	Clear a queue (PCF command Clear queue only).
ctrl	Start, and stop the specified channel, listener, or service. And ping the specified channel.
ctrlx	Reset or resolve the specified channel.
connect	Connect the application to the specified queue manager by issuing an MQCONN call.
crt	Create objects of the specified type using the appropriate command set.
dlr	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
```

- The following example displays the authorities user1 has for queue a.b.c:

```
dspmqaout -m qmgr1 -n a.b.c -t q -p user1
```

The results from this command are:

```
Entity user1 has the following authorizations for object:
get
put
```

Related commands

dmpmqaut	Dump authority
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

None

Optional parameters

QMgrName

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

<code>strmqcsv</code>	Start a command server
<code>endmqcsv</code>	End a command server

dspmqls (display files)

Purpose

Use the **dspmqls** command to display the real file system name for all WebSphere MQ objects that match a specified criterion. You can use this command to identify the files associated with a particular object. This is useful for backing up specific objects. See “Understanding WebSphere MQ file names” on page 17 for information about name transformation.

Syntax

```
▶▶ dspmqls [-m QMgrName] [-t ObjType] GenericObjName ▶▶
```

Required parameters

GenericObjName

The name of the object. The name is a string with no flag and is a required parameter. Omitting the name returns an error.

This parameter supports a wild card character `*` at the end of the string.

Optional parameters

-m *QMgrName*

The name of the queue manager for which to examine files. If you omit this name, the command operates on the default queue manager.

-t *ObjType*

The 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
authinfo	Authentication information object, for use with Secure Sockets Layer (SSL) channel security
channel or chl	A channel
clntconn or clcn	A client connection channel
catalog or ctlg	An object catalog
namelist or nl	A namelist
listener or lstr	A listener

process or prcs	A process
queue or q	A queue or queues matching the object name parameter
qalias or qa	An alias queue
qlocal or ql	A local queue
qmodel or qm	A model queue
qremote or qr	A remote queue
qmgr	A queue manager object
service or srvc	A service

Note:

1. The **dspmqls** command displays the name of the directory containing the queue, *not* the name of the queue itself.
2. In WebSphere MQ for UNIX systems, you need to prevent the shell from interpreting the meaning of special characters, for example, *. The way you do this depends on the shell you are using, but might involve the use of single quotation marks, double quotation marks, or a backslash.

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 defined on the default queue manager.

```
dspmqls SYSTEM.ADMIN*
```
2. The following command displays file details for all processes with names beginning PROC defined on queue manager RADIUS.

```
dspmqls -m RADIUS -t prcs PROC*
```

dspmqrte (WebSphere MQ display route application)

Purpose

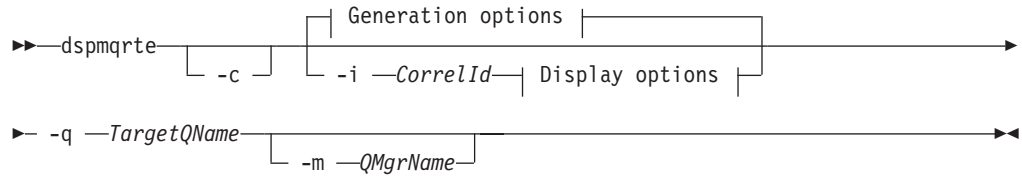
The WebSphere MQ display route application (dspmqrte) can be executed on all WebSphere MQ Version 6.0 queue managers, with the exception of WebSphere MQ for z/OS queue managers. You can execute the WebSphere MQ display route application as a client to a WebSphere MQ for z/OS Version 6.0 queue manager by specifying the -c parameter when issuing the dspmqrte command.

Note: To run a client application against a WebSphere MQ for z/OS queue manager, the client attachment feature must be installed.

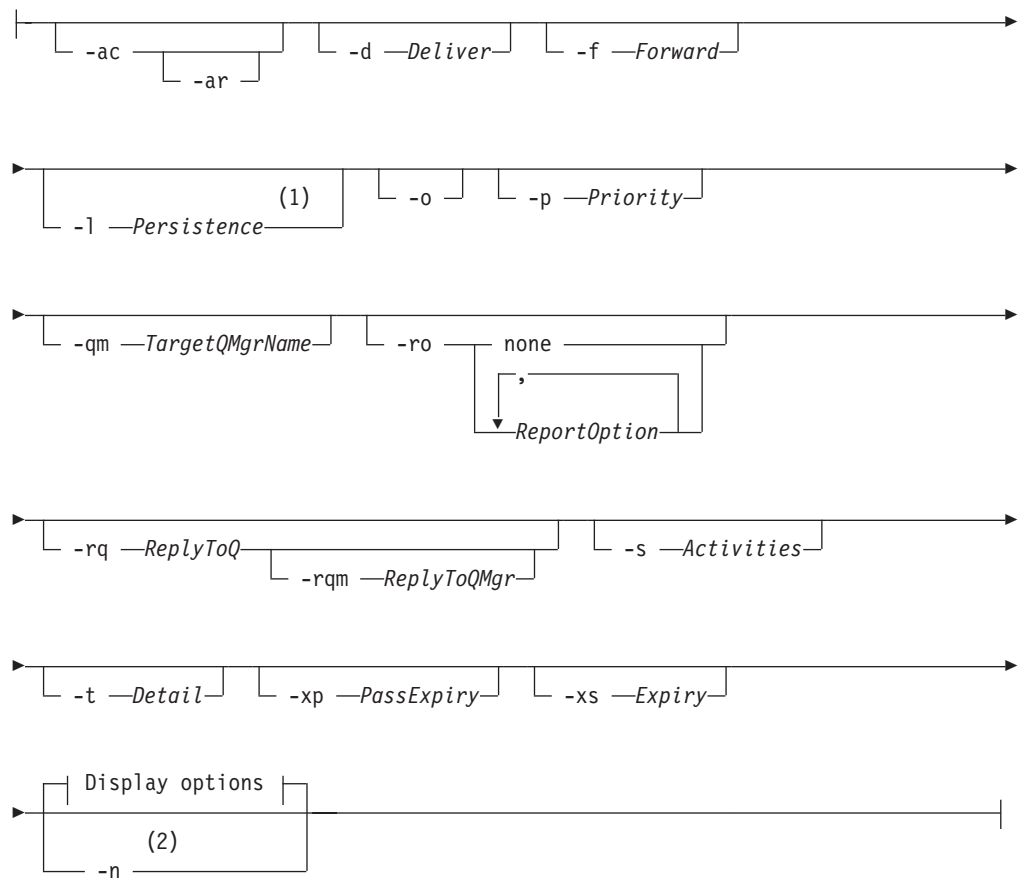
Use **dspmqrte** to help determine the route a message has taken through a queue manager network. The WebSphere MQ display route application generates and puts a trace-route message into a queue manager network. As the trace-route message travels through the queue manager network, activity information is recorded. When the trace-route message reaches it's target queue the activity information is collected by the WebSphere MQ display route application and

displayed. For more information, and examples of using the WebSphere MQ display route application, see the Monitoring WebSphere MQ *WebSphere MQ Monitoring* book.

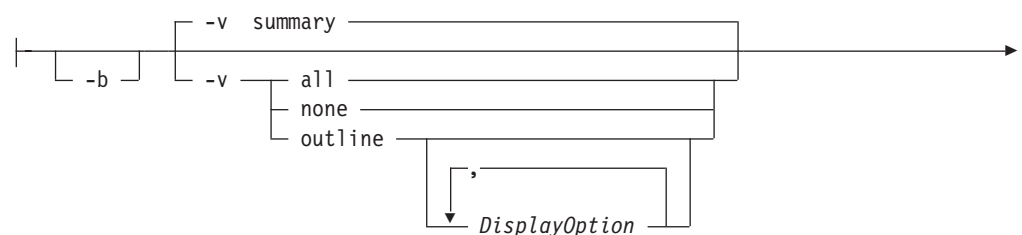
Syntax



Generation options:



Display options:



└─w *WaitTime*─┘

Notes:

- 1 If *Persistence* is specified as *yes*, and is accompanied by a request for a trace-route reply message (*-ar*), or any report generating options (*-ro ReportOption*), then you must specify the parameter *-rq ReplyToQ*. The reply-to queue must not resolve to a temporary dynamic queue.
- 2 If this parameter is accompanied by a request for a trace-route reply message (*-ar*), or any of the report generating options (*-ro ReportOption*), then a specific (non-model) reply-to queue must be specified using *-rq ReplyToQ*. By default, activity report messages are requested.

Required parameters

-q *TargetQName*

If the WebSphere MQ display route application is being used to send a trace-route message into a queue manager network, *TargetQName* specifies the name of the target queue.

If the WebSphere MQ display route application is being used to view previously gathered activity information, *TargetQName* specifies the name of the queue where the activity information is stored.

Optional parameters

- c Specifies that the WebSphere MQ display route application connects as a client application. For more information on how to set up client machines, see the WebSphere MQ Clients *WebSphere MQ Clients* book.

If you do not specify this parameter, the WebSphere MQ display route application does not connect as a client application.

-i *CorrelId*

This parameter is used when the WebSphere MQ display route application is used to display previously accumulated activity information only. There can be many activity reports and trace-route reply messages on the queue specified by *-q TargetQName*. *CorrelId* is used to identify the activity reports, or a trace-route reply message, related to a trace-route message. Specify the message identifier of the original trace-route message in *CorrelId*.

The format of *CorrelId* is a 48 character hexadecimal string.

-m *QMGrName*

The name of the queue manager to which the WebSphere MQ display route application connects. The name can contain up to 48 characters.

If you do not specify this parameter, the default queue manager is used.

Generation Options

The following parameters are used when the WebSphere MQ display route application is used to put a trace-route message into a queue manager network.

-ac

Specifies that activity information is to be accumulated within the trace-route message.

If you do not specify this parameter, activity information is **not** accumulated within the trace-route message.

- ar Requests that a trace-route reply message containing all accumulated activity information is generated in the following circumstances:
 - The trace-route message is discarded by a WebSphere MQ Version 6 queue manager.
 - The trace-route message is put to a local queue (target queue or dead-letter queue) by a WebSphere MQ Version 6 queue manager.
 - The number of activities performed on the trace-route message exceeds the value of specified in *-s Activities*.

For more information on trace-route reply messages, see the Monitoring WebSphere MQ *WebSphere MQ Monitoring* book.

If you do not specify this parameter, a trace-route reply message is **not** requested.

-d *Deliver*

Specifies whether the trace-route message is to be delivered to the target queue on arrival. Possible values for *Deliver* are:

- | | |
|------------|---|
| yes | On arrival, the trace-route message is put to the target queue, even if the queue manager does not support trace-route messaging. |
| no | On arrival, the trace-route message is not put to the target queue. |

If you do not specify this parameter, the trace-route message is **not** put to the target queue.

-f *Forward*

Specifies the type of queue manager that the trace-route message can be forwarded to. Queue managers use an algorithm when determining whether to forward a message to a remote queue manager. For details of this algorithm, see Monitoring WebSphere MQ *WebSphere MQ Monitoring*. The possible values for *Forward* are:

- | | |
|------------------|--|
| all | The trace-route message is forwarded to any queue manager.
Warning: If forwarded to a WebSphere MQ queue manager prior to Version 6.0, the trace-route message is not recognized and can be delivered to a local queue despite the value of the <i>-d Deliver</i> parameter. |
| supported | The trace-route message is forwarded only to a queue manager that honors the <i>Deliver</i> parameter from the <i>TraceRoute</i> PCF group. |

If you do not specify this parameter, the trace-route message is forwarded only to a queue manager that honors the *Deliver* parameter.

-l *Persistence*

Specifies the persistence of the generated trace-route message. Possible values for *Persistence* are:

- | | |
|------------|---|
| yes | The generated trace-route message is persistent. (MQPER_PERSISTENT). |
| no | The generated trace-route message is not persistent. (MQPER_NOT_PERSISTENT). |

q The generated trace-route message inherits its persistence value from the queue specified by *-q TargetQName*. (MQPER_PERSISTENCE_AS_Q_DEF).

A trace-route reply message, or any report messages, returned share the same persistence value as the original trace-route message.

If *Persistence* is specified as **yes**, you must specify the parameter *-rq ReplyToQ*. The reply-to queue must not resolve to a temporary dynamic queue.

If you do not specify this parameter, the generated trace-route message is **not** persistent.

-o Specifies that the target queue is not bound to a specific destination. Typically this parameter is used when the trace-route message is to be put across a cluster. The target queue is opened with option MQOO_BIND_NOT_FIXED.

If you do not specify this parameter, the target queue is bound to a specific destination.

-p *Priority*

Specifies the priority of the trace-route message. The value of *Priority* is either greater than or equal to 0, or MQPRI_PRIORITY_AS_Q_DEF.

MQPRI_PRIORITY_AS_Q_DEF specifies that the priority value is taken from the queue specified by *-q TargetQName*.

If you do not specify this parameter, the priority value is taken from the queue specified by *-q TargetQName*.

-qm *TargetQMgrName*

Qualifies the target queue name; normal queue manager name resolution applies. The target queue is specified with *-q TargetQName*.

If you do not specify this parameter, the queue manager to which the WebSphere MQ display route application is connected is used as the reply-to queue manager.

-ro **none** | *ReportOption*

none Specifies no report options are set.

ReportOption Specifies report options for the trace-route message. Multiple report options can be specified using a comma as a separator. Possible values for *ReportOption* are:

activity The report option MQRO_ACTIVITY is set.

coa The report option MQRO_COA_WITH_FULL_DATA is set.

cod The report option MQRO_COD_WITH_FULL_DATA is set.

exception The report option MQRO_EXCEPTION_WITH_FULL_DATA is set.

expiration The report option MQRO_EXPIRATION_WITH_FULL_DATA is set.

discard The report option MQRO_DISCARD_MSG is set.

If neither *-ro ReportOption* nor *-ro none* are specified, then the MQRO_ACTIVITY and MQRO_DISCARD_MSG report options are specified.

-rq *ReplyToQ*

Specifies the name of the reply-to queue that all responses to the trace-route message are sent to. If the trace-route message is persistent, or if the *-n* parameter is specified, a reply-to queue must be specified that is **not** a temporary dynamic queue.

If you do not specify this parameter, the system default model queue, SYSTEM.DEFAULT.MODEL.QUEUE is used as the reply-to queue. Using this model queue causes a temporary dynamic queue, for the WebSphere MQ display route application, to be created.

-rqm *ReplyToQMgr*

Specifies the name of the queue manager where the reply-to queue resides. The name can contain up to 48 characters.

If you do not specify this parameter, the queue manager to which the WebSphere MQ display route application is connected is used as the reply-to queue manager.

-s *Activities*

Specifies the maximum number of recorded activities that can be performed on behalf of the trace-route message before it is discarded. This prevents the trace-route message from being forwarded indefinitely if caught in an infinite loop. The value of *Activities* is either greater than or equal to 1, or MQROUTE_UNLIMITED_ACTIVITIES. MQROUTE_UNLIMITED_ACTIVITIES specifies that an unlimited number of activities can be performed on behalf of the trace-route message.

If you do not specify this parameter, an unlimited number of activities can be performed on behalf of the trace-route message.

-t *Detail*

Specifies the activities that are recorded. The possible values for *Detail* are:

low	Activities performed by user-defined application are recorded only.
medium	Activities specified in low are recorded. Additionally, activities performed by MCAs are recorded.
high	Activities specified in low , and medium are recorded. MCAs do not expose any further activity information at this level of detail. This option is available to user-defined applications that are to expose further activity information only. For example, if a user-defined application determines the route a message takes by considering certain message characteristics, the routing logic could be included with this level of detail.

If you do not specify this parameter, medium level activities are recorded.

-xp *PassExpiry*

Specifies whether the report option MQRO_DISCARD_MSG and the remaining expiry time from the trace-route message is passed on to the trace-route reply message. Possible values for *PassExpiry* are:

yes	<p>The report option MQRO_PASS_DISCARD_AND_EXPIRY is specified in the message descriptor of the trace-route message.</p> <p>If a trace-route reply message, or activity reports, are generated for the trace-route message, the MQRO_DISCARD_MSG report option (if specified), and the remaining expiry time are passed on.</p> <p>This is the default value.</p>
no	<p>The report option MQRO_PASS_DISCARD_AND_EXPIRY is not specified.</p> <p>If a trace-route reply message is generated for the trace-route message, the discard option and remaining expiry time from the trace-route message are not passed on.</p>

If you do not specify this parameter, the MQRO_PASS_DISCARD_AND_EXPIRY report option is not specified in the trace-route message.

-xs Expiry

Specifies the expiry time for the trace-route message, in seconds.

If you do not specify this parameter, the expiry time is specified as 60 seconds.

-n Specifies that activity information returned for the trace-route message is not to be displayed.

If this parameter is accompanied by a request for a trace-route reply message, (*-ar*), or any of the report generating options from (*-ro ReportOption*), then a specific (non-model) reply-to queue must be specified using *-rq ReplyToQ*. By default, activity report messages are requested.

After the trace-route message is put to the specified target queue, a 48 character hexadecimal string is returned containing the message identifier of the trace-route message. The message identifier can be used by the WebSphere MQ display route application to display the activity information for the trace-route message at a later time, using the *-i CorrelId* parameter.

If you do not specify this parameter, activity information returned for the trace-route message is displayed in the form specified by the *-v* parameter.

Display options

The following parameters are used when the WebSphere MQ display route application is used to display collected activity information.

-b Specifies that the WebSphere MQ display route application browses only activity reports or a trace-route reply message related to a message. This allows activity information to be displayed again at a later time.

If you do not specify this parameter, the WebSphere MQ display route application destructively gets activity reports or a trace-route reply message related to a message.

-v summary | all | none | outline DisplayOption

summary	The queues that the trace-route message was routed through are displayed.
all	All available information is displayed.
none	No information is displayed.

outline *DisplayOption* Specifies display options for the trace-route message. Multiple display options can be specified using a comma as a separator.

If no values are supplied the following is displayed:

- The application name
- The type of each operation
- Any operation specific parameters

Possible values for *DisplayOption* are:

activity All non-PCF group parameters in *Activity* PCF groups are displayed.

identifiers

Values with parameter identifiers MQBACF_MSG_ID or MQBACF_CORREL_ID are displayed. This overrides *msgdelta*.

message

All non-PCF group parameters in *Message* PCF groups are displayed. When this value is specified, you cannot specify *msgdelta*.

msgdelta

All non-PCF group parameters in *Message* PCF groups, that have changed since the last operation, are displayed. When this value is specified, you cannot specify *message*.

operation

All non-PCF group parameters in *Operation* PCF groups are displayed.

traceroute

All non-PCF group parameters in *TraceRoute* PCF groups are displayed.

If you do not specify this parameter, a summary of the message route is displayed.

-w *WaitTime*

Specifies the time, in seconds, that the WebSphere MQ display route application waits for activity reports, or a trace-route reply message, to return to the specified reply-to queue.

If you do not specify this parameter, the wait time is specified as the expiry time of the trace-route message, plus 60 seconds.

Return codes

0	Command completed normally
10	Invalid arguments supplied
20	An error occurred during processing

Examples

1. The following command puts a trace-route message into a queue manager network with the target queue specified as TARGET.Q. Providing queue managers on route are enabled for activity recording, activity reports are generated. Depending on the queue manager attribute, ACTIVREC, activity

reports are either delivered to the reply-to queue ACT.REPORT.REPLY.Q, or are delivered to a system queue. The trace-route message is discarded on arrival at the target queue.

```
dspmqrte -q TARGET.Q -rq ACT.REPORT.REPLY.Q
```

Providing one or more activity reports are delivered to the reply-to queue, ACT.REPORT.REPLY.Q, the WebSphere MQ display route application orders and displays the activity information.

2. The following command puts a trace-route message into a queue manager network with the target queue specified as TARGET.Q. Activity information is accumulated within the trace-route message, but activity reports are not generated. On arrival at the target queue the trace-route message is discarded. Depending on the value of the target queue manager attribute, ROUTEREC, a trace-route reply message can be generated and delivered to either the reply-to queue, TRR.REPLY.TO.Q, or to a system queue.

```
dspmqrte -ac -ar -ro discard -rq TRR.REPLY.TO.Q -q TARGET.Q
```

Providing a trace-route reply message is generated and is delivered to the reply-to queue TRR.REPLY.TO.Q, the WebSphere MQ display route application orders and displays the activity information that was accumulated in the trace-route message.

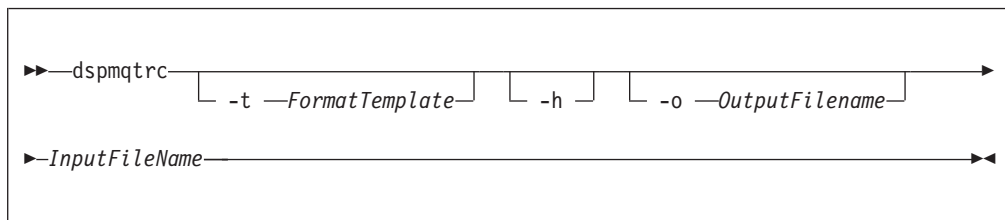
For more examples of using the WebSphere MQ display route application and its output, see the Monitoring WebSphere MQ *WebSphere MQ Monitoring* book.

dspmqtrc (Display WebSphere MQ formatted trace output)

Purpose

Use the **dspmqtrc** command to display WebSphere MQ formatted trace output.

Syntax



Required parameters

InputFileName

Specifies the name of the file containing the unformatted trace. For example
MQS_ROOT:[MQM.TRACE]AMQ20202345.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.

-h Omit header information from the report.

-o *output_filename*

The name of the file into which to write formatted data.

Examples

1. The following command shows the redirection of output:

```
dspmqtrc mqs_root:[mqm.trace]amq20202345.trc > mqs_root:[mqm.trace]amq20202345.fmt
```

Related commands

endmqtrc

End WebSphere MQ trace

strmqtrc

Start WebSphere MQ trace

dspmqtrn (display transactions)

Purpose

Use the **dspmqtrn** command to display details of in-doubt transactions. This includes transactions coordinated by WebSphere MQ and by an external transaction manager.

For each in-doubt transaction, a transaction number (a human-readable transaction identifier), the transaction state, and the transaction ID are displayed. (Transaction IDs can be up to 128 characters long, hence the need for a transaction number.)

Syntax

```
▶▶ dspmqtrn [ -e ] [ -i ] [ -m QMgrName ] ▶▶
```

Optional parameters

- e** Requests details of externally coordinated, in-doubt transactions. Such transactions are those for which WebSphere MQ has been asked to prepare to commit, but has not yet been informed of the transaction outcome.
- i** Requests details of internally coordinated, in-doubt transactions. Such transactions are those for which each resource manager has been asked to prepare to commit, but WebSphere MQ has yet to inform the resource managers of the transaction outcome.

Information about the state of the transaction in each of its participating resource managers is displayed. This information can help you assess the affects of failure in a particular resource manager.

Note: If you specify neither **-e** nor **-i**, details of both internally and externally coordinated in-doubt transactions are displayed.

-m *QMgrName*

The name of the queue manager for which to display transactions. If you omit the name, the default queue manager's transactions are displayed.

Return codes

0	Successful operation
36	Invalid arguments supplied
40	Queue manager not available
49	Queue manager stopping
69	Storage not available
71	Unexpected error
72	Queue manager name error
102	No transactions found

Related commands

rsvmqtrn	Resolve transaction
----------	---------------------

dspmqver (display version information)

Purpose

Use the **dspmqver** command to display WebSphere MQ version and build information.

Syntax

```
▶▶ dspmqver [ -p Components ] [ -f Fields ] [ -b ] [ -v ] ▶▶▶
```

Optional parameters

-p *Components*

Display information for the components specified by *Component*. Either a single component, or multiple components can be specified. To specify multiple components, sum the values of the required components, then specify *Component* as the total of the summation. Available components and related values follow:

1	WebSphere MQ server, or client.
2	WebSphere MQ classes for Java™.
4	WebSphere MQ classes for Java Message Service.
8	WebScale Distribution Hub

The default value is 1.

-f *Fields*

Display information for the fields specified by *Field*. Either a single field, or multiple fields can be specified. To specify multiple fields, sum the values of the required fields, then specify *Field* as the total of the summation. Available fields and related values follow:

1	Name
---	------

2	Version, in the form V.R.M.F: Where V=Version, R=Release, M=Modification, and F=Fix pack
4	CMVC level
8	Build type

Information for each selected field is displayed on a separate line when the `dspmqr` command is executed.

The default value is 15. This displays information for all fields.

- b** Omit header information from the report.
- v** Display verbose output.

Return codes

0	Command completed normally.
10	Command completed with unexpected results.
20	An error occurred during processing.

Examples

The following command displays WebSphere MQ version and build information, using the default settings for *-p Components* and *-f Fields*:

```
dspmqr
```

The following command displays version and build information for the WebSphere MQ classes for Java:

```
dspmqr -p 2
```

The following command displays the name and version of the WebSphere MQ classes for Java Message Service:

```
dspmqr -p 4 -f 3
```

The following command displays the build level of the WebScale Distribution Hub:

```
dspmqr -p 8 -f 4
```

endmqcsv (end command server)

Purpose

Use the **endmqcsv** command to stop the command server on the specified queue manager.

If the queue manager attribute, `SCMDSERV`, is specified as `QMGR` then changing the state of the command server using **endmqcsv** does not effect how the queue manager acts upon the `SCMDSERV` attribute at the next restart.

Syntax

```
▶▶ endmqcsv [ -c ] [ -i ] QMgrName ▶▶
```

Required parameters

QMgrName

The name of the queue manager for which to end the command server.

Optional parameters

-c Stops the command server 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 Stops the command server immediately. Actions associated with a command message currently being processed might not complete.

Return codes

0	Command completed normally
10	Command completed with unexpected results
20	An error occurred during processing

Examples

1. The following command stops the command server on queue manager saturn.queue.manager:

```
endmqcsv -c saturn.queue.manager
```

The command server can complete processing any command it has already started before it stops. Any new commands received remain unprocessed in the command queue until the command server is restarted.

2. The following command stops the command server on queue manager pluto immediately:

```
endmqcsv -i pluto
```

Related commands

strmqcsv	Start a command server
dspmqcsv	Display the status of a command server

endmqlsr (end listener)

Purpose

The endmqlsr command ends all listener processes for the specified queue manager.

You do not need to stop the queue manager before issuing the endmqlsr command. If any of the listeners are configured to have inbound channels running within the runmqlsr listener process, rather than within a pool process, the request

to end that listener might fail if channels are still active. In this case a message is written indicating how many listeners were successfully ended and how many listeners are still running.

If the listener attribute, CONTROL, is specified as QMGR then changing the state of the listener using endmq1sr does not effect how the queue manager acts upon the CONTROL attribute at the next restart.

Syntax

```
►► endmq1sr [ -w ] [ -m QMgrName ]
```

Optional parameters

-m *QMgrName*

The name of the queue manager. If you omit this, the command operates on the default queue manager.

-w Wait before returning control.

Control is returned to you only after all listeners for the specified queue manager have stopped.

Return codes

0	Command completed normally
10	Command completed with unexpected results
20	An error occurred during processing

endmqm (end queue manager)

Purpose

Use the **endmqm** command to end (stop) a specified local queue manager. This command stops a queue manager in one of three modes:

- Controlled or quiesced shutdown
- Immediate shutdown
- Preemptive shutdown

The attributes of the queue manager and the objects associated with it are not affected. You can restart the queue manager using the **strmqm** (Start queue manager) command.

To delete a queue manager, stop it and then use the **dltmqm** (Delete queue manager) command.

Issuing the **endmqm** command affects any client application connected through a server-connection channel. The effect varies depending on the parameter used, but it is as though a STOP CHANNEL command was issued in one of the three possible modes. See *WebSphere MQ Clients* for information on the effects of STOP CHANNEL modes on server-connection channels. The **endmqm** optional parameter descriptions state which STOP CHANNEL mode they are equivalent to.

Syntax



Required parameters

QMgrName

The name of the message queue manager to be stopped.

Optional parameters

-c Controlled (or quiesced) shutdown. This is the default.

The queue manager stops, but only after all applications have disconnected. Any MQI calls currently being processed are completed.

Control is returned to you immediately and you are not notified of when the queue manager has stopped.

The effect on any client applications connected through a server-connection channel is equivalent to a STOP CHANNEL command issued in QUIESCE mode.

-w Wait shutdown.

This type of shutdown is equivalent to a controlled shutdown except that control is returned to you only after the queue manager has stopped. You receive the message *Waiting for queue manager qmName to end while shutdown progresses*.

The effect on any client applications connected through a server-connection channel is equivalent to a STOP CHANNEL command issued in QUIESCE mode.

-i Immediate shutdown. The queue manager stops after it has completed all the MQI calls currently being processed. Any MQI requests issued after the command has been issued fail. Any incomplete units of work are rolled back when the queue manager is next started.

Control is returned after the queue manager has ended.

The effect on any client applications connected through a server-connection channel is equivalent to a STOP CHANNEL command issued in FORCE mode.

-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 might stop without waiting for applications to disconnect or for MQI calls to complete. This can give unpredictable results for WebSphere MQ applications. The shutdown mode is set to *immediate shutdown*. If the queue manager has not stopped after a few seconds, the shutdown mode is escalated, and all remaining queue manager processes are stopped.

The effect on any client applications connected through a server-connection channel is equivalent to a STOP CHANNEL command issued in TERMINATE mode.

-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
119	Permission denied (Windows only)

Examples

The following examples show commands that 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

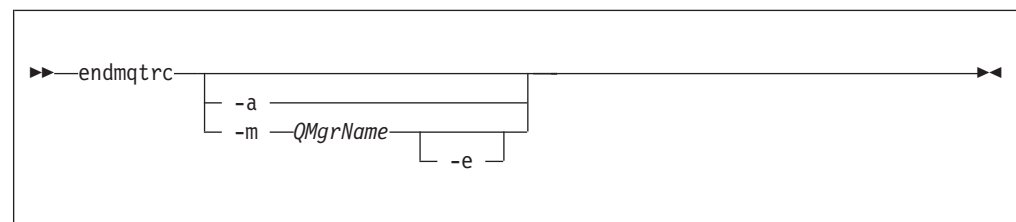
<code>crtmqm</code>	Create a queue manager
<code>strmqm</code>	Start a queue manager
<code>dlmqm</code>	Delete a queue manager

endmqtrc (End WebSphere MQ 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 WebSphere MQ trace

mqftapp (run File Transfer Application GUI)

Purpose

The **mqftapp** command is available with the **File Transfer Application on WebSphere MQ for Windows, and WebSphere MQ for Linux (x86 platform) servers only**.

Use the **mqftapp** command to run the File Transfer Application graphical user interface (GUI).

Alternatively, on WebSphere MQ for Windows you can start the File Transfer Application by selecting it through the start menu.

When run for the first time, the graphical user interface must be configured. For instructions of how to do this, see Appendix F, "OpenVMS cluster failover set templates," on page 293.

Syntax

The syntax of this command follows:

▶▶—mqftapp—▶▶

Related commands

mqftrcv Receive file on server

mqftrcvc Receive file on client

mqftsnd	Send file from server
mqftsndc	Send file from client

mqftrcv (receive file on server)

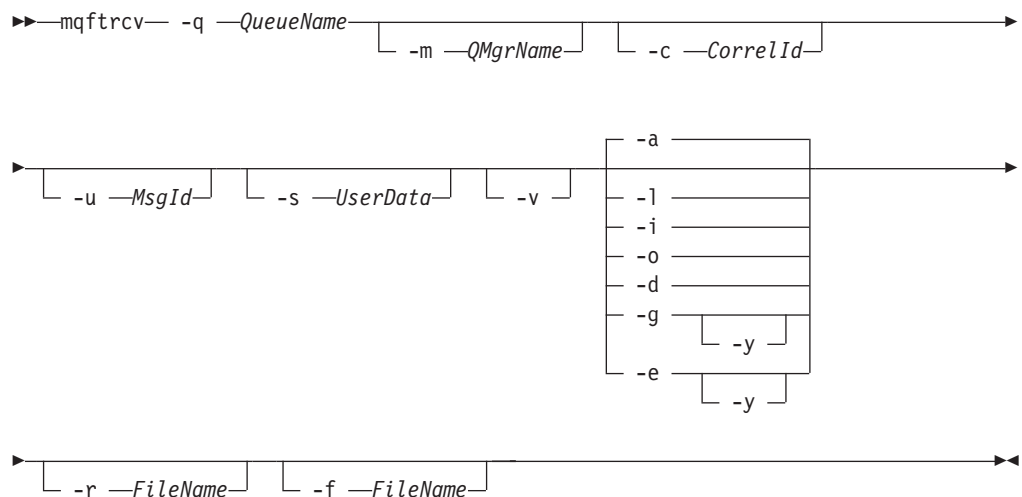
Purpose

The **mqftrcv** command is available with the File Transfer Application on WebSphere MQ for Windows, and WebSphere MQ for Linux (x86 platform) servers only.

Use the **mqftrcv** command to do one of the following:

- Receive a file.
- Extract a file.
- Delete a file.
- View sent files.

Syntax



Required parameters

-q *QueueName*
The local name of the destination queue.

Optional parameters

-m *QMGrName*
The name of the queue manager that hosts the destination queue. A queue manager that does not have the File Transfer Application installed can be specified. If you omit this parameter, the default queue manager is used.

-c *CorrelId*
Select all files matching *CorrelId*. Selection can be combined with **-s** *UserData*, and **-f** *FileName*.

- u *MsgID***
Select the message that has a message ID that matches *MsgID*. Used to select other messages.
- s *UserData***
Select files by locating any occurrence of the character string *UserData*, in part or all of the file's **UserData**. The comparison is case sensitive, and wildcard characters cannot be used.

Selection can be combined with *-c CorrelId*, and *-f FileName*.
- v** Return the *CorrelId*, and *MsgId* of the file.
- a** List all files and messages, in the following order:
 1. Complete files, ordered by queue name
 2. Incomplete files, ordered by queue name
 3. Other messages, ordered by queue name
 This is the default value. For more information on file status see Appendix G, "MONMQ diagnostic utility," on page 299.
- l** List all complete files, ordered by queue name.
- i** List all incomplete files, ordered by queue name.
- o** List all other messages, ordered by queue name.
- d** Delete the specified file, or the group of messages. If more than one file matches the selection criteria, no files are deleted and a return code is returned.
- g** Receive a complete file. Message associated with the file are removed. If a file already exists of the same name, do one of the following:
 - Specify the *-y* parameter, so that the existing file is overwritten.
 - Specify the *-r FileName* parameter, so that the file is renamed.
- e** Extract a complete, or incomplete file. Messages associated with the file are not removed. If a file already exists of the same name, do one of the following:
 - Specify the *-y* parameter, so that the existing file is overwritten.
 - Specify the *-r FileName* parameter, so that the file is renamed.
- y** Replace an existing file of the same name. Used with optional parameters *-g*, and *-e*.
- r *FileName***
Assign new file name and/or file location.

Used to rename, or to relocate a file. The file is assigned the name specified in *FileName*. A fully qualified file name can be specified to relocate the file. If the file name, or path, contains embedded spaces, it must be specified in double quotes. One file can be specified only, and you cannot use wildcard characters.
- f *FileName***
Select all files matching *FileName*. The fully qualified file name can be specified. If the file name contains embedded spaces, it must be specified in double quotes. You cannot use wildcard characters.

Selection can be combined with *-c CorrelId*, and *-s UserData*.

Return codes

- 0 Successful operation

36	Invalid arguments supplied
40	Queue manager not available
69	Storage not available
71	Unexpected error
163	Queue name required
164	Cannot open queue
165	Cannot open file
166	Cannot put to queue
167	No file name specified (Send)
168	Message length is too small to send data
169	Sending file has changed
170	Cannot get from queue
171	Cannot write to file
172	CorrelId is invalid
173	MsgId is invalid
174	No messages to receive
175	File for delete is not unique

Examples

This command lists all files and messages on the queue, MY.QUEUE, located on the default queue manager:

```
mqftrcv -q MY.QUEUE -a
```

This command gets the first complete file on the queue, MY.QUEUE, located on queue manager QM1:

```
mqftrcv -q MY.QUEUE -m QM1 -g
```

This command gets the complete file, named My document.txt, on the queue, MY.QUEUE, located on the default queue manager:

```
mqftrcv -q MY.QUEUE -g -f "My document.txt"
```

This command gets the complete file, named My document.txt, also marked URGENT, on the queue, MY.QUEUE, located on queue manager QM1 :

```
mqftrcv -q MY.QUEUE -m QM1 -g -f "My document.txt" -s "URGENT"
```

Related commands

mqftapp	Run File Transfer Application
mqftrcv	Receive file on client
mqftsnd	Send file from server
mqftsndc	Send file from client

mqftrcv (receive file on client)

Purpose

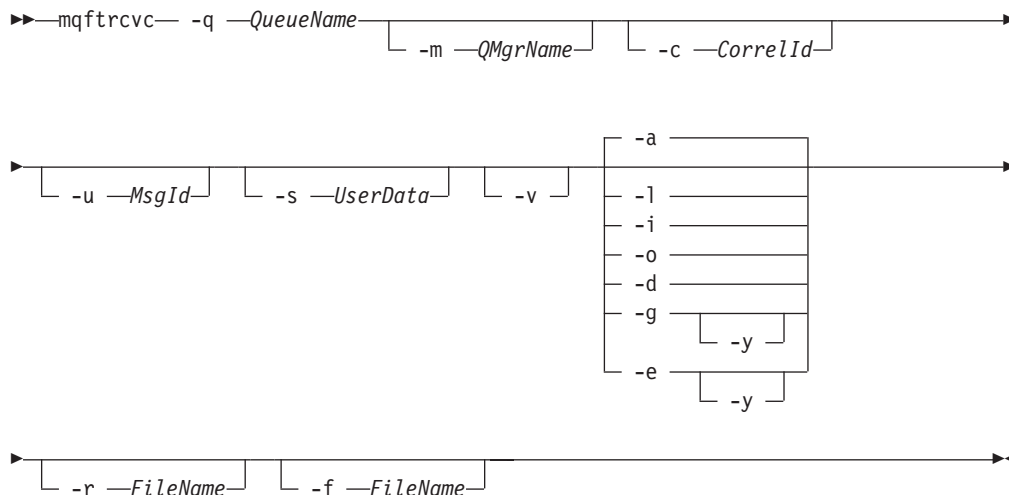
The **mqftrcv** command is available with the File Transfer Application on WebSphere MQ for Windows, and WebSphere MQ for Linux (x86 platform) clients only.

Use the **mqftrcv** command to do one of the following:

- Receive a file from a connected server.

- Extract a file from a connected server.
- Delete a file from a connected server.
- View sent files on a connected server.

Syntax



Required parameters

- q *QueueName***
The local name of the destination queue.

Optional parameters

- m *QMgrName***
The name of the queue manager that hosts the destination queue. A queue manager that does not have the File Transfer Application installed can be specified. If you omit this parameter, the default queue manager is used.
- c *CorrelId***
Select all files matching *CorrelId*. Selection can be combined with `-s UserData`, and `-f FileName`.
- u *MsgID***
Select the message that has a message ID that matches *MsgID*. Used to select other messages.
- s *UserData***
Select files by locating any occurrence of the character string *UserData*, in part or all of the file's **UserData**. The comparison is case sensitive, and wildcard characters cannot be used.

Selection can be combined with `-c CorrelId`, and `-f FileName`.
- v** Return the *CorrelId*, and *MsgId* of the file.
- a** List all files and messages, in the following order:
 1. Complete files, ordered by queue name
 2. Incomplete files, ordered by queue name
 3. Other messages, ordered by queue name

This is the default value. For more information on file status see Appendix G, "MONMQ diagnostic utility," on page 299.

- l List all complete files, ordered by queue name.
- i List all incomplete files, ordered by queue name.
- o List all other messages, ordered by queue name.
- d Delete the specified file, or the group of messages. If more than one file matches the selection criteria, no files are deleted and a return code is returned.
- g Receive a complete file. Message associated with the file are removed. If a file already exists of the same name, do one of the following:
 - Specify the *-y* parameter, so that the existing file is overwritten.
 - Specify the *-r FileName* parameter, so that the file is renamed.
- e Extract a complete, or incomplete file. Messages associated with the file are not removed. If a file already exists of the same name, do one of the following:
 - Specify the *-y* parameter, so that the existing file is overwritten.
 - Specify the *-r FileName* parameter, so that the file is renamed.
- y Replace an existing file of the same name. Used with optional parameters *-g*, and *-e*.

-r *FileName*

Assign new file name and/or file location.

Used to rename, or to relocate a file. The file is assigned the name specified in *FileName*. A fully qualified file name can be specified to relocate the file. If the file name, or path, contains embedded spaces, it must be specified in double quotes. One file can be specified only, and you cannot use wildcard characters.

-f *FileName*

Select all files matching *FileName*. The fully qualified file name can be specified. If the file name contains embedded spaces, it must be specified in double quotes. You cannot use wildcard characters.

Selection can be combined with *-c CorrelId*, and *-s UserData*.

Return codes

0	Successful operation
36	Invalid arguments supplied
40	Queue manager not available
69	Storage not available
71	Unexpected error
163	Queue name required
164	Cannot open queue
165	Cannot open file
166	Cannot put to queue
167	No file name specified (Send)
168	Message length is too small to send data
169	Sending file has changed
170	Cannot get from queue
171	Cannot write to file
172	CorrelId is invalid
173	MsgId is invalid
174	No messages to receive

Examples

This command lists all files and messages on the queue, MY.QUEUE, located on the default queue manager:

```
mqftrcvc -q MY.QUEUE -a
```

This command gets the first complete file on the queue, MY.QUEUE, located on queue manager QM1:

```
mqftrcvc -q MY.QUEUE -m QM1 -g
```

This command gets the complete file, named My document.txt, on the queue, MY.QUEUE, located on the default queue manager:

```
mqftrcvc -q MY.QUEUE -g -f "My document.txt"
```

This command gets the complete file, named My document.txt, also marked URGENT, on the queue, MY.QUEUE, located on queue manager QM1 :

```
mqftrcvc -q MY.QUEUE -m QM1 -g -f "My document.txt" -s "URGENT"
```

Related commands

mqftapp	Run File Transfer Application
mqftrcv	Receive file on server
mqftsnd	Send file from server
mqftsndc	Send file from client

mqftsnd (send file from server)

Purpose

The **mqftsnd** command is available with the File Transfer Application on WebSphere MQ for Windows, and WebSphere MQ for Linux (x86 platform) servers only.

Use the **mqftsnd** command to send a file from a WebSphere MQ server using the File Transfer Application.

Syntax

```

>> mqftsnd -q QueueName [ -m QMGrName ] [ -t TargetQMGrName ]
[ -v ] [ -l MsgLength ] [ -p yes ] [ -p no ] [ -p queue ] [ -s UserData ]
>> -f FileName

```

Required parameters

-q *QueueName*

The local name of the destination queue.

-f *FileName*

The name of the file to be transmitted. The fully qualified file name can be specified. If the file name contains embedded spaces, it must be specified in double quotes. One file can be specified only, and you cannot use wildcard characters.

Note: The file is not deleted from its original location during a send.

Optional parameters

-m *QMgrName*

The name of the queue manager that has access to the file at its origin. If you omit this parameter, the default queue manager is used.

-t *TargetQMgrName*

The name of the queue manager that hosts the destination queue. If you omit this parameter, the queue manager specified by *QMgrName* is used.

-v Return the *CorrelId* of the file.

-l *MessageSize*

The maximum size of a segmented message in bytes.

If a file is too large to be sent as a single message, the file is segmented into a number smaller messages, known as segments, and all these segments are transmitted instead. When all the segments reach their destination, the target queue manager reassembles them to form the original file.

Specify a value between 250 and the queue manager's maximum message length. To determine the maximum message length, use the MQIA_MAX_MSG_LENGTH selector with the MQINQ call.

The default value is 100000.

-p *yes*

Messages are persistent. This is the default value.

-p *no*

Messages are not persistent.

-p *queue*

Messages persistence is defined by the queue.

-s *UserData*

An character string that contains user information relevant to the file being sent. The content of this data is of no significance to the target queue manager.

Return codes

0	Successful operation
36	Invalid arguments supplied
40	Queue manager not available
69	Storage not available
71	Unexpected error
163	Queue name required
164	Cannot open queue
165	Cannot open file

166	Cannot put to queue
167	No file name specified (Send)
168	Message length is too small to send data
169	Sending file has changed
170	Cannot get from queue
171	Cannot write to file
172	CorrelId is invalid
173	MsgId is invalid
174	No messages to receive
175	File for delete is not unique

Examples

This command sends a file from the default queue manager, to the queue DEST.Q, located on queue manager QM2:

```
mqftsnd -q DEST.Q -t QM2 -f "My document.txt"
```

This command sends a file as non-persistent messages from queue manager QM1, to the queue DEST.Q, located on the default queue manager, setting the maximum segment size to 50000 bytes:

```
mqftsnd -q DEST.Q -m QM1 -l 50000 -p no -f "C:\My Downloads\My document.idd"
```

Related commands

mqftapp	Run File Transfer Application
mqftcv	Receive file on server
mqftcvc	Receive file on client
mqftsndc	Send file from client

mqftsndc (send file from client)

Purpose

The **mqftsndc** command is available with the File Transfer Application on WebSphere MQ for Windows, and WebSphere MQ for Linux (x86 platform) clients only.

Use the **mqftsndc** command to send a file from a WebSphere MQ client using the File Transfer Application.

Syntax

```

>> mqftsndc -q QueueName [-m QMgrName] [-t TargetQMgrName]
    [-v] [-l MsgLength] [-p yes] [-p no] [-p queue] [-s UserData]

```

► -f *FileName* ◀

Required parameters

-q *QueueName*

The local name of the destination queue.

-f *FileName*

The name of the file to be transmitted. The fully qualified file name can be specified. If the file name contains embedded spaces, it must be specified in double quotes. One file can be specified only, and you cannot use wildcard characters.

Note: The file is not deleted from its original location during a send.

Optional parameters

-m *QMgrName*

The name of the queue manager that has access to the file at its origin. If you omit this parameter, the default queue manager is used.

-t *TargetQMgrName*

The name of the queue manager that hosts the destination queue. If you omit this parameter, the queue manager specified by *QMgrName* is used.

-v Return the *CorrelId* of the file.

-l *MessageSize*

The maximum size of a segmented message in bytes.

If a file is too large to be sent as a single message, the file is segmented into a number smaller messages, known as segments, and all these segments are transmitted instead. When all the segments reach their destination, the target queue manager reassembles them to form the original file.

Specify a value between 250 and the queue manager's maximum message length. To determine the maximum message length, use the MQIA_MAX_MSG_LENGTH selector with the MQINQ call.

The default value is 100000.

-p *yes*

Messages are persistent. This is the default value.

-p *no*

Messages are not persistent.

-p *queue*

Messages persistence is defined by the queue.

-s *UserData*

An character string that contains user information relevant to the file being sent. The content of this data is of no significance to the target queue manager.

Return codes

0	Successful operation
36	Invalid arguments supplied
40	Queue manager not available
69	Storage not available

71	Unexpected error
163	Queue name required
164	Cannot open queue
165	Cannot open file
166	Cannot put to queue
167	No file name specified (Send)
168	Message length is too small to send data
169	Sending file has changed
170	Cannot get from queue
171	Cannot write to file
172	CorrelId is invalid
173	MsgId is invalid
174	No messages to receive
175	File for delete is not unique

Examples

This command sends a file from the default queue manager, to the queue DEST.Q, located on queue manager QM2:

```
mqftsndc -q DEST.Q -t QM2 -f "My document.txt"
```

This command sends a non-persistent file from queue manager QM1, to the queue DEST.Q, located on the default queue manager, setting the maximum segment size to 50000 bytes:

```
mqftsndc -q DEST.Q -m QM1 -l 50000 -p no -f "C:\My Downloads\My document.idd"
```

Related commands

mqftapp	Run File Transfer Application
mqftrcv	Receive file on server
mqftrcvc	Receive file on client
mqftsnd	Send file from server

rcdmqimg (record media image)

Purpose

Use the **rcdmqimg** command to write an image of an object, or group of objects, to the log for use in media recovery. This command can only be used when using linear logging. Use the associated command **rcrmqobj** to recreate the object from the image.

You use this command 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

```

>> rcdmqimg [ -m QMGrName ] [ -z ] [ -l ] -t ObjectType

```

Required parameters

GenericObjName

The name of the object to record. This parameter can have a trailing asterisk to record that any objects with names matching the portion of the name before the asterisk.

This parameter is required *unless* you are recording a queue manager object or the channel synchronization file. Any object name you specify for the channel synchronization file is ignored.

-t *ObjectType*

The types of object for which to record images. Valid object types are:

* or all	All the object types
authinfo	Authentication information object, for use with Secure Sockets Layer (SSL) channel security
channel or chl	Channels
clntconn or clcn	Client connection channels
catalog or ctlg	An object catalog
listener or lstr	Listeners
namelist or nl	Namelists
process or prcs	Processes
queue or q	All types of queue
qalias or qa	Alias queues
qlocal or ql	Local queues
qmodel or qm	Model queues
qremote or qr	Remote queues
qmgr	Queue manager object
service or srvc	Service
syncfile	Channel synchronization file.

Note: When using WebSphere MQ for UNIX systems, you need to prevent the shell from interpreting the meaning of special characters, for example, *. How you do this depends on the shell you are using, but might involve the use of single quotation marks, double quotation marks, or a backslash.

Optional parameters

-m *QMgrName*

The name of the queue manager for which to record images. If you omit this, the command operates on the default queue manager.

-z Suppresses error messages.

-l Writes messages containing the names of the oldest log files needed to restart the queue manager and to perform media recovery. The messages are written to the error log and the standard error destination. (If you specify both the **-z** and **-l** parameters, the messages are sent to the error log, but not to the standard error destination.)

When issuing a sequence of **rcdmqimg** commands, include the **-l** parameter only on the last command in the sequence, so that the log file information is gathered only once.

Return codes

0	Successful operation
36	Invalid arguments supplied
40	Queue manager not available
49	Queue manager stopping
68	Media recovery 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 this command to recreate an object, or group of objects, from their images contained in the log. This command can only be used when using linear logging. Use the associated command, **rcdmqimg**, to record the object images to the log.

Use this command on a running queue manager. All activity on the queue manager after the image was recorded is logged. To recreate an object, replay the log to recreate events that occurred after the object image was captured.

Syntax

```
►► rcrmqobj [ -m QMGrName ] [ -z ] -t ObjectType GenericObjName ►►
```

Required parameters

GenericObjName

The name of the object to re-create. This parameter can have a trailing asterisk to re-create any objects with names matching the portion of the name before the asterisk.

This parameter is required *unless* the object type is the channel synchronization file; any object name supplied for this object type is ignored.

-t *ObjectType*

The types of object to re-create. Valid object types are:

* or all	All object types
authinfo	Authentication information object, for use with Secure Sockets Layer (SSL) channel security
channel or chl	Channels
clntconn or clcn	Client connection channels
clchltab	Client channel table
listener or lstr	Listener
namelist or nl	Namelists
process or prcs	Processes
queue or q	All types of queue
qalias or qa	Alias queues
qlocal or ql	Local queues
qmodel or qm	Model queues
qremote or qr	Remote queues
service or srvc	Service
syncfile	Channel synchronization file.

You can use this option when circular logs are configured but syncfile fails if the channel scratchpad files, which are used to rebuild syncfile, are damaged or missing.

Note: When using WebSphere MQ for UNIX systems, you need to prevent the shell from interpreting the meaning of special characters, for example, *. How you do this depends on the shell you are using, but might involve the use of single quotation marks, double quotation marks, or a backslash.

Optional parameters

-m *QMgrName*

The name of the queue manager for which to recreate objects. If omitted, the command operates on the default queue manager.

-z Suppresses error messages.

Return codes

0	Successful operation
36	Invalid arguments supplied
40	Queue manager not available
49	Queue manager stopping
66	Media image not available
68	Media recovery 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

`rcdmqing` Record an object in the log

rsvmqtrn (resolve transactions)

Purpose

Use the **rsvmqtrn** command to commit or back out internally or externally coordinated in-doubt transactions.

Use this command only when you are certain that transactions cannot be resolved by the normal protocols. Issuing this command might result in the loss of transactional integrity between resource managers for a distributed transaction.

Syntax

```
▶▶ rsvmqtrn -a _____ -m QMgrName ▶▶
      |_____|
      | -b _____ Transaction |
      | -c _____ |
      | -r RMID      |
```

Required parameters

-m *QMgrName*
The name of the queue manager.

Optional parameters

- a** The queue manager resolves all internally-coordinated, in-doubt transactions (that is, all global units of work).
- b** Backs out the named transaction. This flag is valid for externally-coordinated transactions (that is, for external units of work) only.
- c** Commits the named transaction. This flag is valid for externally-coordinated transactions (that is, external units of work) only.
- r** *RMID*
The resource manager whose participation in the in-doubt transaction can be ignored. This flag is valid for internally-coordinated transactions only, and for resource managers that have had their resource manager configuration entries removed from the queue manager configuration information.

Note: The queue manager does not call the resource manager. Instead, it marks the resource manager's participation in the transaction as being complete.

Transaction

The transaction number of the transaction being committed or backed out. Use the **dspmqrn** command to find the relevant transaction number. This parameter is required with the **-b**, **-c**, and **-r** *RMID* parameters.

Return codes

0	Successful operation
32	Transactions could not be resolved
34	Resource manager not recognized
35	Resource manager not permanently unavailable
36	Invalid arguments supplied
40	Queue manager not available
49	Queue manager stopping
69	Storage not available
71	Unexpected error
72	Queue manager name error
85	Transactions not known

Related commands

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 *WebSphere MQ Intercommunication* *WebSphere MQ Intercommunications*.

The channel initiator is started by default as part of the queue manager.

Syntax

```
runmqchi [-q InitiationQName] [-m QMGrName]
```

Optional parameters

-q *InitiationQName*

The name of the initiation queue to be processed by this channel initiator. If you omit it, SYSTEM.CHANNEL.INITQ is used.

-m *QMGrName*

The name of the queue manager on which the initiation queue exists. If you omit the name, the default queue manager is used.

& To run the channel initiator as a background process add ampersand (&) at the end of the **runmqchi** control command line.

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, review the queue manager error log that the channel is associated with for the error messages, and the system error log for records of problems that occur before the channel is associated with the queue manager. For more information about error logs, see "Error logs" on page 189.

runmqchl (run channel)

Purpose

Use the **runmqchl** command to run either a sender (SDR) or a requester (RQSTR) channel.

The channel runs synchronously. To stop the channel, issue the MQSC command STOP CHANNEL.

Syntax

```
runmqchl -c ChannelName [-m QMgrName]
```

Required parameters

-c *ChannelName*
The name of the channel to run.

Optional parameters

-m *QMgrName*
The name of the queue manager with which this channel is associated. If you omit the name, the default queue manager is used.

& To run the channel as a background process add ampersand (&) at the end of the **runmqchl** control command line.

Return codes

0	Command completed normally
10	Command completed with unexpected results
20	An error occurred during processing

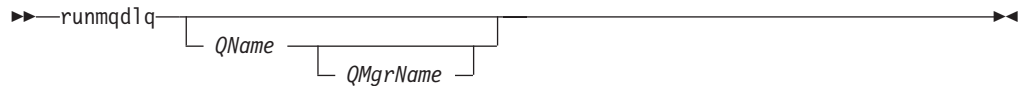
If return codes 10 or 20 are generated, review the error log of the associated queue manager for the error messages, and the system error log for records of problems that occur before the channel is associated with the queue manager.

runmqdlq (run dead-letter queue handler)

Purpose

Use the **runmqdlq** command to start the dead-letter queue (DLQ) handler, which monitors and handles messages on a dead-letter queue.

Syntax



Description

Use the dead-letter queue handler to perform various actions on selected messages by specifying a set of rules that can both select a message and define the action to be performed on that message.

The **runmqdlq** command takes its input from `stdin`. When the command is processed, the results and a summary are put into a report that is sent to `stdout`.

By taking `stdin` from the keyboard, you can enter **runmqdlq** rules interactively.

By redirecting the input from a file, you can apply a rules table to the specified queue. The rules table must contain at least one rule.

If you use the DLQ handler without redirecting `stdin` from a file (the rules table), the DLQ handler reads its input from the keyboard. In WebSphere MQ for AIX, Solaris, HP-UX, and Linux, the DLQ handler does not start to process the named queue until it receives an `end_of_file` (Ctrl+D) character. In WebSphere MQ for Windows, it does not start to process the named queue until you press the following sequence of keys: Ctrl+Z, Enter, Ctrl+Z, Enter.

For more information about rules tables and how to construct them, see “The DLQ handler rules table” on page 126.

Optional parameters

The MQSC command rules for comment lines and for joining lines also apply to the DLQ handler input parameters.

QName

The name of the queue to be processed.

If you omit the name, the dead-letter queue defined for the local queue manager is used. If you enter one or more blanks (' '), the dead-letter queue of the local queue manager is explicitly assigned.

QMgrName

The name of the queue manager that owns the queue to be processed.

If you omit the name, the default queue manager for the installation is used. If you enter one or more blanks (' '), the default queue manager for this installation is explicitly assigned.

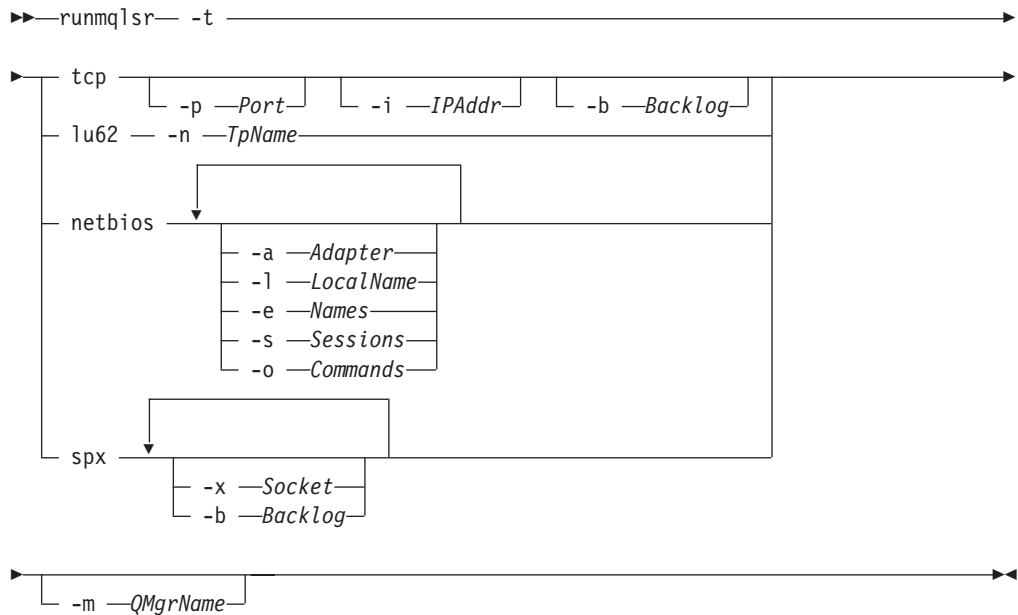
runmqlsr (run listener)

Purpose

Use the **runmqlsr** command to start a listener process.

This command is run synchronously and waits until the listener process has finished before returning to the caller.

Syntax



Required parameters

-t The transmission protocol to be used:

tcp	Transmission Control Protocol / Internet Protocol (TCP/IP)
lu62	SNA LU 6.2 (Windows only)
netbios	NetBIOS (Windows only)
spx	SPX (Windows only)

Optional parameters

-p *Port*

The port number for TCP/IP. This flag is valid for TCP only. If you omit the port number, it is taken from the queue manager configuration information, or from defaults in the program. The default value is 1414.

-i *IPAddr*

The IP address for the listener, specified in one of the following formats:

- IPv4 dotted decimal
- IPv6 hexadecimal notation
- Alphanumeric format

This flag is valid for TCP/IP only.

On systems that are both IPv4 and IPv6 capable you can split the traffic by running two separate listeners, one listening on all IPv4 addresses and one listening on all IPv6 addresses. If you omit this parameter, the listener listens on all configured IPv4 and IPv6 addresses.

-n *TpName*

The LU 6.2 transaction program name. This flag is valid only for the LU 6.2 transmission protocol. If you omit the name, it is taken from the queue manager configuration information.

-a *Adapter*

The adapter number on which NetBIOS listens. By default the listener uses adapter 0.

-l *LocalName*

The NetBIOS local name that the listener uses. The default is specified in the queue manager configuration information.

-e *Names*

The number of names that the listener can use. The default value is specified in the queue manager configuration information.

-s *Sessions*

The number of sessions that the listener can use. The default value is specified in the queue manager configuration information.

-o *Commands*

The number of commands that the listener can use. The default value is specified in the queue manager configuration information.

-x *Socket*

The SPX socket on which SPX listens. The default value is hexadecimal 5E86.

-m *QMgrName*

The name of the queue manager. By default the command operates on the default queue manager.

-b *Backlog*

The number of concurrent connection requests that the listener supports. See "The LU62 and TCP stanzas" on page 85 for a list of default values and further information.

& To run listener as a background process add ampersand (&) at the end of the **runmqtsr** control command line.

Return codes

0	Command completed normally
10	Command completed with unexpected results
20	An error occurred during processing

Examples

The following command runs a listener on the default queue manager using the NetBIOS protocol. The listener can use a maximum of five names, five commands, and five sessions. These resources must be within the limits set in the queue manager configuration information.

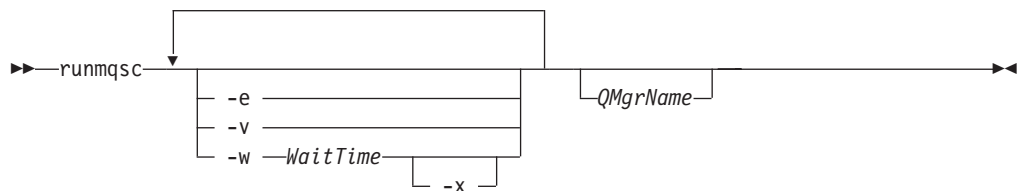
```
runmqtsr -t netbios -e 5 -s 5 -o 5
```

runmqsc (run MQSC 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 WebSphere MQ Script (MQSC) Command Reference *WebSphere MQ Script (MQSC) Command Reference*.

Syntax



Description

You can invoke the **runmqsc** command in three ways:

Verify command

Verify MQSC commands but do not run them. An output report is generated indicating the success or failure of each command. This mode is available on a local queue manager only.

Run command directly

Send MQSC commands directly to a local queue manager.

Run command indirectly

Run MQSC commands on a remote queue manager. These commands are put on the command queue on a remote queue manager and run in the order in which they were queued. Reports from the commands are returned to the local queue manager.

Indirect mode operation is performed through the default queue manager.

The **runmqsc** command takes its input from `stdin`. When the commands are processed, the results and a summary are put into a report that is sent to `stdout`.

By taking `stdin` from the keyboard, you can enter MQSC commands interactively.

By redirecting the input from a file, you can run a sequence of frequently-used commands contained in the file. You can also redirect the output report to a file.

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 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*

Run the MQSC commands 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 60 for more information.

WaitTime

The time, in seconds, that **runmqsc** waits for replies. Any replies received after this are discarded, but the MQSC commands 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 The target queue manager is running under z/OS. 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 WebSphere MQ for z/OS command queue.

QMgrName

The name of the target queue manager on which to run the MQSC commands, by default, the default queue manager.

Return codes

00	MQSC command file processed successfully
10	MQSC command file processed with errors; report contains reasons for failing commands
20	Error; MQSC command file not run

Examples

1. Enter this command at the command prompt:

```
runmqsc
```

Now you can enter MQSC commands directly at the command prompt. No queue manager name is specified, so the MQSC commands are processed on the default queue manager.

2. Use one of these commands, as appropriate in your environment, to specify that MQSC commands are to be verified only:

```
runmqsc -v BANK < "/u/users/commfile.in"
```

```
runmqsc -v BANK < "c:\users\commfile.in"
```

This command verifies the MQSC commands in file `commfile.in`. The queue manager name is `BANK`. The output is displayed in the current window.

3. These commands run the MQSC command file `mqscfile.in` against the default queue manager.

```
runmqsc < "/var/mqm/mqsc/mqscfile.in" > "/var/mqm/mqsc/mqscfile.out"

runmqsc < "c:\Program Files\IBM\WebSphere MQ\mqsc\mqscfile.in" >
"c:\Program Files\IBM\WebSphere MQ\mqsc\mqscfile.out"
```

In this example, the output is directed to file `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 WebSphere MQ Application Programming Guide *WebSphere MQ Application Programming Guide*.

Once a trigger monitor has started, it continuously monitors the specified initiation queue. The trigger monitor does not stop until the queue manager ends, see “endmqm (end queue manager)” on page 235. While the client trigger monitor is running it keeps the dead letter queue open.

Syntax

```
runmqtmc [-m QMgrName] [-q InitiationQName]
```

Optional parameters

-m *QMgrName*

The name of the queue manager on which the client trigger monitor operates, by default the default queue manager.

-q *InitiationQName*

The name of the initiation queue to be processed, by default SYSTEM.DEFAULT.INITIATION.QUEUE.

& To run listener as a background process add ampersand (&) at the end of the **runmqtmc** control command line.

Return codes

0	Not used. The client trigger monitor is designed to run continuously and therefore not to end. The value is reserved.
10	Client trigger monitor interrupted by an error.
20	Error; client trigger monitor not run.

Examples

For examples of using this command, refer to the WebSphere MQ Application Programming Guide *WebSphere MQ Application Programming Guide*.

runmqtrm (start trigger monitor)

Purpose

Use the **runmqtrm** command to invoke a trigger monitor. For further information about using trigger monitors, refer to the WebSphere MQ Application Programming Guide *WebSphere MQ Application Programming Guide*.

Once a trigger monitor has started, it continuously monitors the specified initiation queue. The trigger monitor does not stop until the queue manager ends, see “endmqm (end queue manager)” on page 235. While the trigger monitor is running it keeps the dead letter queue open.

Syntax

```
runmqtrm [-m QMgrName] [-q InitiationQName]
```

Optional parameters

-m *QMgrName*

The name of the queue manager on which the trigger monitor operates, by default the default queue manager.

-q *InitiationQName*

Specifies the name of the initiation queue to be processed, by default SYSTEM.DEFAULT.INITIATION.QUEUE.

& To run the trigger monitor as a background process add ampersand (&) at the end of the **runmqtrm** control command line.

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 (grant or revoke authority)

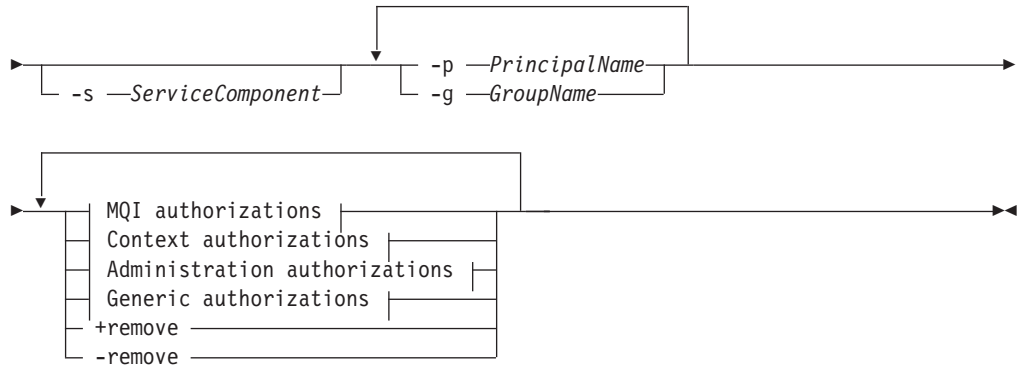
Purpose

Use the **setmqaut** command to change the authorizations to a profile, object, or class of objects. Authorizations can be granted to, or revoked from, any number of principals or groups.

For more information about authorization service components, see “The Service stanza” on page 79 and “The ServiceComponent stanza” on page 80.

Syntax

```
setmqaut [-m QMgrName] -n Profile -t ObjectType
```



MQI authorizations:



Context authorizations:



Administration authorizations:



Generic authorizations:



Description

Use **setmqaut** both to *grant* an authorization, that is, give a principal or user group permission to perform an operation, and to *revoke* an authorization, that is, remove the permission to perform an operation. You must specify the principals and user groups to which the authorizations apply, the queue manager, object type, and the profile name identifying the object or objects.

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 a plus sign (+) or a minus sign (-). For example, if you include **+put** in the authorization list, you grant authority to issue MQPUT calls against a queue. Alternatively, if you include **-put** in the authorization list, you revoke the authority to issue MQPUT calls.

You can specify any number of principals, user groups, and authorizations in a single command, but you must specify at least one principal or user group.

If a principal is a member of more than one user group, the principal effectively has the combined authorities of all those user groups. On Windows systems, the principal also has all the authorities that have been granted to it explicitly using the **setmqaut** command.

On UNIX systems, all authorities are held by user groups internally, not by principals. This has the following implications:

- If you use the **setmqaut** command to grant an authority to a principal, the authority is actually granted to the primary user group of the principal. This means that the authority is effectively granted to all members of that user group.
- If you use the **setmqaut** command to revoke an authority from a principal, the authority is actually revoked from the primary user group of the principal. This means that the authority is effectively revoked from all members of that user group.

To alter authorizations for a cluster sender channel that has been automatically generated by a repository, see *WebSphere MQ Queue Manager Clusters*. This book describes how the authority is inherited from a cluster receiver channel object.

Required parameters

-t *ObjectType*

The type of object for which to change authorizations.

Possible values are:

authinfo	An authentication information object
channel or chl	A channel
clntconn or clcn	A client connection channel
lstr or listener	A listener
namelist or nl	A namelist
process or prcs	A process
queue or q	A queue
qmgr	A queue manager
srvc or service	A service

-n *Profile*

The name of the profile for which to change authorizations. The authorizations apply to all WebSphere MQ objects with names that match the profile name specified.

If you give an explicit profile name (without any wildcard characters), the object identified must exist.

This parameter is required, unless you are changing the authorizations of a queue manager, in which case you must *not* include it. To change the authorizations of a queue manager use the queue manager name, for example

```
setmqaut -m QMGR -t qmgr -p user1 +connect
```

where *QMGR* is the name of the queue manager and *user1* is the user requesting the change.

Optional parameters

-m *QMgrName*

The name of the queue manager of the object for which to change authorizations. The name can contain up to 48 characters.

This parameter is optional if you are changing the authorizations of your default queue manager.

-p *PrincipalName*

The name of the principal for which to change authorizations.

For WebSphere MQ for Windows only, the name of the principal can optionally include a domain name, specified in the following format:

userid@domain

For more information about including domain names on the name of a principal, see Chapter 7, “WebSphere MQ security,” on page 91.

You must have at least one principal or group.

-g *GroupName*

The name of the user group for which to change authorizations. You can specify more than one group name, but each name must be prefixed by the **-g** flag. On Windows systems, you can use only local groups.

-s *ServiceComponent*

The name of the authorization service to which the authorizations apply (if your system supports installable authorization services). This parameter is optional; if you omit it, the authorization update is made to the first installable component for the service.

+remove or -remove

Remove the specified profile. The authorizations associated with the profile no longer apply to WebSphere MQ objects with names that match the profile.

This option cannot be used with the option **-t qmgr**.

Authorizations

The authorizations to be granted or revoked. Each item in the list is prefixed by a plus sign (+), indicating that authority is to be granted, or a minus sign (-), indicating that authority is to be revoked.

For example, to grant authority to issue MQPUT calls, specify **+put** in the list. To revoke the authority to issue MQPUT calls, specify **-put**.

Table 16 shows the authorities that can be given to the different object types.

Table 16. Specifying authorities for different object types

Authority	Queue	Process	Queue manager	Namelist	Auth info	Clntconn	Channel	Listener	Service
all	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
alladm	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
allmqi	Yes	Yes	Yes	Yes	Yes	No	No	No	No
none	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
altusr	No	No	Yes	No	No	No	No	No	No
browse	Yes	No	No	No	No	No	No	No	No
chg	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
clr	Yes	No	No	No	No	No	No	No	No

Table 16. Specifying authorities for different object types (continued)

Authority	Queue	Process	Queue manager	Namelist	Auth info	Clntconn	Channel	Listener	Service
connect	No	No	Yes	No	No	No	No	No	No
crt	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
ctrl	No	No	No	No	No	No	Yes	Yes	Yes
ctrlx	No	No	No	No	No	No	Yes	No	No
dlt	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
dsp	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
get	Yes	No	No	No	No	No	No	No	No
put	Yes	No	No	No	No	No	No	No	No
inq	Yes	Yes	Yes	Yes	Yes	No	No	No	No
passall	Yes	No	No	No	No	No	No	No	No
passid	Yes	No	No	No	No	No	No	No	No
set	Yes	Yes	Yes	No	No	No	No	No	No
setall	Yes	No	Yes	No	No	No	No	No	No
setid	Yes	No	Yes	No	No	No	No	No	No

Return codes

0	Successful operation
36	Invalid arguments supplied
40	Queue manager not available
49	Queue manager stopping
69	Storage not available
71	Unexpected error
72	Queue manager name error
133	Unknown object name
145	Unexpected object name
146	Object name missing
147	Object type missing
148	Invalid object type
149	Entity name missing
150	Authorization specification missing
151	Invalid authorization specification

Examples

1. This example shows a command that specifies that the object on which authorizations are being given is the queue orange.queue on queue manager saturn.queue.manager. If the queue does not exist, the command fails.

```
setmqaut -m saturn.queue.manager -n orange.queue -t queue
-g tango +inq +alladm
```

The authorizations are given to a user group called tango, and the associated authorization list specifies that the user group can:

- Issue MQINQ calls
- Perform all administration operations on that object

2. In this example, the authorization list specifies that a user group called foxy:

- Cannot issue any MQI calls to the specified queue
- Can perform all administration operations on the specified queue

If the queue does not exist, the command fails.

```
setmqaut -m saturn.queue.manager -n orange.queue -t queue
-g foxy -allmqi +alladm
```

3. This example gives user1 full access to all queues with names beginning a.b. on queue manager qmgr1. The profile is persistent and applies to any object with a name that matches the profile.

```
setmqaut -m qmgr1 -n a.b.* -t q -p user1 +all
```

4. This example deletes the specified profile.

```
setmqaut -m qmgr1 -n a.b.* -t q -p user1 -remove
```

5. This example creates a profile with no authority.

```
setmqaut -m qmgr1 -n a.b.* -t q -p user1 +none
```

Related commands

dmpmqaut	Dump authority
dspmqaut	Display authority

Authorizations for MQI calls

altusr	Use another user's authority for MQOPEN and MQPUT1 calls.
browse	Retrieve a message from a queue using an MQGET call with the BROWSE option.
connect	Connect the application to the specified queue manager using an MQCONN call.
get	Retrieve a message from a queue using an MQGET call.
inq	Make an inquiry on a specific queue using an MQINQ call.
put	Put a message on a specific queue using an MQPUT call.
set	Set attributes on a queue from the MQI using an MQSET call.

Note: If you open a queue for multiple options, you must be authorized for each option.

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.
crt	Create objects of the specified type.
dlt	Delete the specified object.
dsp	Display the attributes of the specified object.

ctrl	Start, and stop the specified channel, listener, or service. And ping the specified channel.
ctrlx	Reset or resolve the specified channel.

Authorizations for generic operations

all	Use all operations applicable to the object.
alladm	Use all administration operations applicable to the object.
allmqi	Use all MQI calls applicable to the object.
none	No authority. Use this to create profiles without authority.

setmqprd (enroll production license)

Purpose

Use the **setmqprd** command to enroll a WebSphere MQ production license.

For further information about enrolling production licenses, see the WebSphere MQ *Quick Beginnings* book for your operating system.

Syntax

►► setmqprd *LicenseFile* ◀◀

Required parameters

LicenseFile

Specifies the fully-qualified name of the production license certificate file.

This is usually **amqpcert.lic**.

strmqcsv (start command server)

Purpose

Use the **strmqcsv** command to start the command server for the specified queue manager. This enables WebSphere MQ to process commands sent to the command queue.

If the queue manager attribute, *SCMDSERV*, is specified as *QMGR* then changing the state of the command server using **strmqcsv** does not effect how the queue manager acts upon the *SCMDSERV* attribute at the next restart.

Syntax

►► strmqcsv -a QMgrName ◀◀

Required parameters

None

Optional parameters

- a Blocks the following PCF commands from modifying or displaying authority information:
 - Inquire authority records (MQCMD_INQUIRE_AUTH_RECS)
 - Inquire entity authority (MQCMD_INQUIRE_ENTITY_AUTH)
 - Set authority record (MQCMD_SET_AUTH_REC).
 - Delete authority record (MQCMD_DELETE_AUTH_REC).

QMgrName

The name of the queue manager on which to start the command server. If 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

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

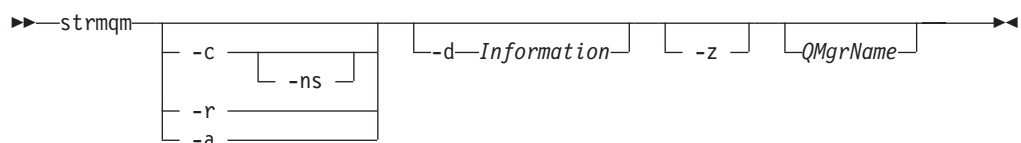
strmqm (start queue manager)

Purpose

Use the **strmqm** command to start a local queue manager.

If the queue manager start up takes more than a few seconds WebSphere MQ shows intermittent messages detailing the start up progress. For more information on these messages see WebSphere MQ Messages *WebSphere MQ Messages*.

Syntax



Optional parameters

- c Starts the queue manager, redefines the default and system objects, then stops the queue manager. (Use the **crmqm** command to create the default and system objects for a queue manager.) Any existing system and default objects belonging to the queue manager are replaced if you specify this flag.

-ns

Prevents any of the following processes from starting automatically when the queue manager starts:

- The channel initiator
- The command server
- Listeners
- Services

-r

Updates the backup queue manager. The backup queue manager is not started. WebSphere MQ updates the backup queue manager's objects by reading the queue manager log and replaying updates to the object files.

For more information on using backup queue managers, see "Backing up and restoring WebSphere MQ" on page 169.

-a

Once activated, a backup queue manager can be started using the control command `strmqm QMgrName`. The requirement to activate a backup queue manager prevents accidental startup.

Once activated, a backup queue manager can no longer be updated.

For more information on using backup queue managers, see "Backing up and restoring WebSphere MQ" on page 169.

-d *Information*

Specifies whether information messages are displayed. Possible values for *Information* follow:

all	All information messages are displayed. This is the default value.
minimal	The minimal number of information messages are displayed.
none	No information messages are displayed. This parameter is equivalent to <code>-z</code> .

The `-z` parameter takes precedence over this parameter.

-z

This flag is used within WebSphere MQ to suppress unwanted information messages. Because using this flag could result in loss of information, do not use it when entering commands on a command line.

This parameter takes precedence over the `-d` parameter.

QMgrName

The name of a local queue manager. If omitted, the default queue manager is used.

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

24	A process that was using the previous instance of the queue manager has not yet disconnected.
49	Queue manager stopping
69	Storage not available
71	Unexpected error
72	Queue manager name error
100	Log location invalid
119	User not authorized to start the queue manager

Examples

The following command starts the queue manager account:

```
strmqm account
```

Related commands

<code>crtmqm</code>	Create a queue manager
<code>dltmqm</code>	Delete a queue manager
<code>endmqm</code>	End a queue manager

Appendix A. System and default objects

When you create a queue manager using the **crtmqm** control command, the system objects and default objects are created automatically.

- The system objects are those WebSphere MQ objects required for the operation of a queue manager or channel.
- The default objects define all of the attributes of an object. When you create an object, such as a local queue, any attributes that you do not specify explicitly are inherited from the default object.

Table 17. System and default objects for queues

Object Name	Description
SYSTEM.ADMIN.ACCOUNTING.QUEUE	The queue that holds accounting monitoring data.
SYSTEM.ADMIN.ACTIVITY.QUEUE	The queue that holds returned activity reports.
SYSTEM.ADMIN.CHANNEL.EVENT	Event queue for channels.
SYSTEM.ADMIN.COMMAND.QUEUE	Administration command queue. Used for remote MQSC commands and PCF commands.
SYSTEM.ADMIN.CONFIG.EVENT	Event queue for configuration events.
SYSTEM.ADMIN.PERFM.EVENT	Event queue for performance events.
SYSTEM.ADMIN.QMGR.EVENT	Event queue for queue manager events.
SYSTEM.ADMIN.STATISTICS.QUEUE	The queue that holds statistics monitoring data.
SYSTEM.ADMIN.TRACE.ROUTE.QUEUE	The queue that holds returned trace-route reply messages.
SYSTEM.AUTH.DATA.QUEUE	The queue that holds access control lists for the queue manager.
SYSTEM.CHANNEL.INITQ	Channel initiation queue.
SYSTEM.CHANNEL.SYNCQ	The queue that holds the synchronization data for channels.
SYSTEM.CICS.INITIATION.QUEUE	Default CICS initiation queue.
SYSTEM.CLUSTER.COMMAND.QUEUE	The queue used to carry messages to the repository queue manager.
SYSTEM.CLUSTER.REPOSITORY.QUEUE	The queue used to store all repository information.
SYSTEM.CLUSTER.TRANSMIT.QUEUE	The transmission queue for all messages to all clusters.
SYSTEM.DEAD.LETTER.QUEUE	Dead-letter (undelivered-message) queue.
SYSTEM.DEFAULT.ALIAS.QUEUE	The transmission queue for all messages to all clusters.
SYSTEM.DEFAULT.INITIATION.QUEUE	Default initiation queue.
SYSTEM.DEFAULT.LOCAL.QUEUE	Default local queue.
SYSTEM.DEFAULT.MODEL.QUEUE	Default model queue.
SYSTEM.DEFAULT.REMOTE.QUEUE	Default remote queue.

Table 17. System and default objects for queues (continued)

Object Name	Description
SYSTEM.MQEXPLORER.REPLY.MODEL	The WebSphere MQ Explorer reply-to queue. This is a model queue that creates a temporary dynamic queue for replies to the WebSphere MQ Explorer.
SYSTEM.MQSC.REPLY.QUEUE	MQSC reply-to-queue. This a model queue that creates a temporary dynamic queue for replies to remote MQSC commands.
SYSTEM.PENDING.DATA.QUEUE	Support deferred messages in JMS.

Table 18. System and default objects for channels

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

Table 19. System and default objects for namelists

Object Name	Description
SYSTEM.DEFAULT.NAMELIST	Default namelist.

Table 20. System and default objects for processes

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

Appendix B. Directory structure

Figure 22 on page 276 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 are modified accordingly.

```

MQS_ROOT:[MQM]
+--> [.CONV]
|
+--> [.TABLE] --> CCSID.TBL
+--> [.ERRORS]
+--> [.EXITS]
+--> [.LOG] --> [.Qmname] --> amqh1ct1.lfh
|
+--> [.ACTIVE] --> S0000000*.LOG
+--> MQS.INI
+--> [.QMGRS] +--> [.SYSTEM] -->> amqcap.inf
|
| +--> [.ERRORS]
| +--> [.ESEM]
| +--> [.ISEM]
| +--> [.MSEM]
| +--> [.SHMEM]
| +--> [.SSEM]
|
| [.Qmname] --> amqalchk.fil
| +--> qm.ini
| +--> qmstatus.ini
| +--> [,$APP]
| | +--> [.ESEM]
| | +--> [.ISEM]
| | +--> [.MSEM]
| | +--> [.SHMEM]
| | +--> [,$PIPE]
| +--> [,$IPCC]
| | +--> AMQCLCHL.TAB
| | +--> AMQRSYNA.DAT
| | +--> [.ESEM]
| | +--> [.ISEM]
| | +--> [.MSEM]
| | +--> [.SHMEM]
| | +--> [,$PIPE]
| +--> [,$QMPERSIST]
| | +--> [.ESEM]
| | +--> [.ISEM]
| | +--> [.MSEM]
| | +--> [.SHMEM]
| | +--> [,$PIPE]
| +--> [.AUTHINFO]
| | +--> SYSTEM$DEFAULT$AUTHINFO$CRLLDAP.
| +--> [.CHANNEL]
| | +--> SYSTEM$AUTO$RECEIVER.
| | +--> SYSTEM$AUTO$SVRCONN.
| | +--> SYSTEM$DEF$CLUSRCVR.
| | +--> SYSTEM$DEF$CLUSDR.
| | +--> SYSTEM$DEF$RECEIVER.
| | +--> SYSTEM$DEF$REQUESTER.
| | +--> SYSTEM$DEF$SENDER.
| | +--> SYSTEM$DEF$SERVER.
| | +--> SYSTEM$DEF$SVRCONN.
| +--> [.CLNTCONN]
| | +--> SYSTEM$DEF$CLNTCONN.
| +--> [.ERRORS] --> AMQERR0*.LOG
| +--> [.ESEM]
| +--> [.ISEM]
| +--> [.LISTENER]
| | +--> SYSTEM$DEFAULT$LISTENER$TCP.
| +--> [.MSEM]
| +--> [.NAMELIST]
| | +--> SYSTEM$DEFAULT$NAMELIST.
| +--> [.PLUGCOMP]
| +--> [.PROCDEF]
| | +--> SYSTEM$DEFAULT$PROCESS.
|
| +--> [.QMANAGER]
| | +--> QMANAGER.
| | +--> QMQOBJCAT.
|
| +--> [.QUEUES]
| +--> [.SERVICES]
| | +--> SYSTEM$BROKER.
| | +--> SYSTEM$DEFAULT$SERVICE.
|
| +--> [.SHMEM]
| +--> [,$PIPE]
| +--> [,$SSL]
| +--> [,$STARTPRM]
| +--> [,$ZSOCKETAPP]
| +--> [,$ZSOCKETEC]
|
+--> [.TRACE]

```

Figure 22. Default directory structure after a queue manager has been started

In Figure 22, the layout is representative of WebSphere MQ 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.

Directories and files in MQS_ROOT:[MQM]

By default, the following directories and files are located in the directory MQS_ROOT:[MQM]:

.conv This directory contains all files used for data conversion.

.table This directory contains the ccsid.tbl. file.

.errors This directory contains the operator message files, from newest to oldest:

- AMQERR01.LOG
- AMQERR02.LOG
- AMQERR03.LOG

.exits An empty directory to contain user-written exits.

.lib This directory contains the subdirectory .iconv. The subdirectory contains all the codeset conversion tables.

.iconv A directory containing codeset conversion tables (such as 002501B5.TBL to 44B031A8.TBL).

.log This directory contains the following subdirectory and files after you have installed WebSphere MQ, 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.

mqs.ini

WebSphere MQ configuration file.

.qmgrs

This directory contains a subdirectory .*\$system* and a subdirectory .*qmname* for each queue manager. The .*\$system* directory contains directories and files used internally by WebSphere MQ. For more information about the .*qmname* subdirectory, see “Directories and files in the MQS_ROOT:[MQM.QMGRS.QMNAME] subdirectory.”

.trace This directory contains the trace files created from the **strmqtrc** command.

Directories and files in the MQS_ROOT:[MQM.QMGRS.QMNAME] subdirectory

By default, the following directories and files are located in the directory MQS_ROOT:[MQM.QMGRS.QMNAME]. The .QMNAME is created for every queue manager created and running on the system.

amqalchk.fil

Checkpoint file containing information about last checkpoint.

.auth This directory contains subdirectories and files associated with authority.

\$aclass;

This file contains the authority stanzas for all classes.

.namelist

This directory contains a file for each namelist. Each file contains the authority stanzas for the associated namelist.

\$class;

This file contains the authority stanzas for the namelist class.

.\$mangled

When namelist names contain invalid OpenVMS characters, they are automatically converted to valid OpenVMS names. The valid OpenVMS names are held in this file. See “Directories and files in the MQS_ROOT:[MQM.QMGRS.QMNAME] subdirectory” on page 277.

system\$default\$namelist

This file contains authority stanzas for the system default namelist.

.procdef

Each WebSphere MQ process definition is associated with a file in this directory.

\$class;

This file contains the authority stanzas for the process definition class.

.\$mangled

When process definition names contain invalid OpenVMS characters, they are automatically converted to valid OpenVMS names. The valid OpenVMS names are held in this file. See “Understanding WebSphere MQ file names” on page 17.

.system\$default\$process;

This file contains authority stanzas for the system default processes.

.qmanager

This directory contains a file for each queue manager. Each file contains the authority stanzas for the associated queue manager.

\$class;

This file contains the authority stanzas for the queue manager class.

.\$mangled

When queue manager definition names contain invalid OpenVMS characters, they are automatically converted to valid OpenVMS names. The valid OpenVMS names are held in this file. See “Understanding WebSphere MQ file names” on page 17.

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.

.\$mangled

When queue names contain invalid OpenVMS characters, they are automatically converted to valid OpenVMS names. The valid OpenVMS names are held in this file. See "Understanding WebSphere MQ file names" on page 17.

Definition files for the queue

Each file corresponds to an object predefined for the queue manager.

- system\$admin\$channel\$event.;
- system\$admin\$command\$queue.;
- system\$admin\$perfm\$event.;
- system\$admin\$qmgr\$event.;
- system\$channel\$initq.;
- system\$channel\$syncq.;
- system\$cics\$initiation\$queue.;
- system\$cluster\$command\$queue.;
- system\$cluster\$repository\$queue.;
- system\$cluster\$transmit\$queue.;
- system\$dead\$letter\$queue.;
- system\$default\$alias\$queue.;
- system\$default\$initiation\$queue.;
- system\$default\$local\$queue.;
- system\$default\$model\$queue.;
- system\$default\$remote\$queue.;
- system\$mqsc\$reply\$queue.;

.qaadmin.;

File used internally for controlling authorizations.

.dce Empty directory reserved for use by DCE support.

.errors This directory contains the operator message files, from newest to oldest:

- amqerr01.log
- amqerr02.log
- amqerr03.log

.esem Directory containing files used internally.

.isem Directory containing files used internally.

.msem Directory containing files used internally.

.namelist

This directory contains namelists for each queue manager.

.plugcomp

This empty directory is reserved for use by installable services.

.procdef

Each WebSphere MQ process definition is associated with a file in this directory. The filename matches the process definition name.

qm.ini Queue manager configuration file.

.qmanager

The queue manager object.

qmstatus.ini

This file contains text describing the status of the queue manager.

.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 WebSphere MQ file names” on page 17.

.shmem

.perQueue

Directory containing files used internally.

.ssem Directory containing files used internally.

.startprm

Directory containing temporary files used internally.

.\$ipcc

amqclchl.tab

Client channel table file.

amqrfcda.dat

Channel table file.

.esem Directory containing files used internally.

.isem Directory containing files used internally.

.msem Directory containing files used internally.

.shmem

.perQueue

Directory containing files used internally.

.ssem Directory containing files used internally.

Appendix C. Comparing command sets

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

- “Commands for queue manager administration”
- “Commands for command server administration”
- “Commands for queue administration” on page 282
- “Commands for process administration” on page 282
- “Commands for channel administration” on page 283
- “Other control commands” on page 284

Note: Only MQSC commands that apply to WebSphere MQ for HP OpenVMS are shown.

Commands for queue manager administration

Table 21. Commands for queue manager administration

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

Note: * Not available as PCF commands.

Commands for command server administration

Table 22. Commands for command server administration

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

Table 22. Commands for command server administration (continued)

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

Commands for queue administration

Table 23. Commands for queue administration

PCF	MQSC
Change Queue	ALTER QLOCAL ALTER QALIAS ALTER QMODEL ALTER QREMOTE
Clear Queue	CLEAR QUEUE
Copy Queue	DEFINE QLOCAL(x) LIKE(y) DEFINE QALIAS(x) LIKE(y) DEFINE QMODEL(x) LIKE(y) DEFINE QREMOTE(x) LIKE(y)
Create Queue	DEFINE QLOCAL DEFINE QALIAS DEFINE QMODEL DEFINE QREMOTE
Delete Queue	DELETE QLOCAL DELETE QALIAS DELETE QMODEL DELETE QREMOTE
Inquire Queue	DISPLAY QUEUE
Inquire Queue Names	DISPLAY QUEUE
Note: There are no equivalent control commands in this group.	

Commands for process administration

Table 24. Commands for process administration

PCF	MQSC
Change Process	ALTER PROCESS
Copy Process	DEFINE PROCESS(x) LIKE(y)

Table 24. Commands for process administration (continued)

PCF	MQSC
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.	

Commands for channel administration

Table 25. Commands for channel administration

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

Other control commands

Table 26. Other control commands

Description	Control
Create WebSphere MQ conversion exit	crtmqcvx
Display authority	dspmqaut
Display files used by objects	dspmqfls
Display WebSphere MQ formatted trace output	dspmqtrc
End WebSphere MQ trace	endmqtrc
Manage a failover set	failover
Record media image	rcdmqing
Recreate media object	rcrmqobj
Resolve WebSphere MQ transactions	rsvmqtrn
Run MQSC commands	runmqsc
Run trigger monitor	runmqtrm
Run client trigger monitor	runmqtmc
Set or reset authority	setmqaut
Start a failover monitor	runmqfm
Start WebSphere MQ trace	strmqtrc
Note: Functions in this group are available only as control commands. There are no direct PCF or MQSC equivalents.	

Appendix D. Stopping and removing queue managers manually

If the standard methods for stopping and removing queue managers fail, try the methods described here.

Stopping a queue manager manually

The standard way of stopping queue managers, using the **endmqm** command, should work even in the event of failures within the queue manager. In exceptional circumstances, if this method of stopping a queue manager fails, you can use one of the procedures described here to stop it manually.

Stopping queue managers in WebSphere MQ for OpenVMS systems

To stop a queue manager running under WebSphere MQ for OpenVMS systems:

1. Find the process IDs of the queue manager programs that are still running using the **MQPROC** command:
\$ MQPROC
2. End any queue manager processes that are still running. Use the **DCL** command **STOP/ID**, specifying the process IDs discovered using the **MQPROC** command.

End the processes in the following order:

AMQZMUC0	Critical process manager
AMQZXMA0	Execution controller
AMQZFUMA	OAM process
AMQZLAA0	LQM agents
AMQZMGR0	Process controller
AMQZMUR0	Restartable process manager
AMQRRMFA	The repository process (for clusters)
AMQZDMAA	Deferred message processor
AMQPCSEA	The command server
RUNMQCHI	The channel initiator

If you stop the queue manager manually, FFSTs might be taken, and FDC files placed in `MQS_ROOT:[MQM.ERRORS]`. Do not regard this as a defect in the queue manager.

The queue manager should restart normally, even after you have stopped it using this method.

Removing queue managers manually

If you want to delete the queue manager after stopping it manually, use the **dltmqm** command.

Removing queue managers in WebSphere MQ for Windows

If you encounter problems with the `dltmqm` command in WebSphere MQ for Windows, use the following procedure to delete a queue manager:

1. Type REGEDIT from the command prompt to start the Registry Editor.
2. Select the HKEY_LOCAL_MACHINE window.
3. Navigate the tree structure in the left-hand pane of the Registry Editor to the following key:

```
HKEY_LOCAL_MACHINE\SOFTWARE\IBM\MQSeries\CurrentVersion
```

Make a note of the values within this key called WorkPath and LogPath. Within each of the directories named by these values, you are going to delete a subdirectory containing the data for the queue manager that you are trying to delete. You now need to find out the name of the subdirectory which corresponds to your queue manager.

4. Navigate the tree structure to the following key:

```
HKEY_LOCAL_MACHINE\SOFTWARE\IBM\MQSeries\CurrentVersion\  
Configuration\QueueManager
```

Within this key there is a key for each of the queue managers on this computer containing the configuration information for the queue manager. The name of this queue manager key is the name of the subdirectory in which the queue manager's data is stored in the file system. By default, this name is the same as the queue manager name, but the name might be a transformation of the queue manager name.

5. Examine the keys within the current key. Look for the key that contains a value called Name. Name contains the name of the queue manager you are trying to delete. Make a note of the name of the key containing the name of the queue manager you are trying to delete. This is the subdirectory name.
6. Locate the queue manager data directory. The name of this directory is the WorkPath followed by the subdirectory name. Delete this directory, and all subdirectories and files.
7. Locate the queue manager's log directory. The name of this directory is the LogPath followed by the subdirectory name. Delete this directory, and all subdirectories and files.
8. Remove the registry entries that refer to the deleted queue manager. First, navigate the tree structure in the Registry Editor to the following key:

```
HKEY_LOCAL_MACHINE\SOFTWARE\IBM\MQSeries\CurrentVersion\  
Configuration\DefaultQueueManager
```
9. If the value called Name within this key matches the name of the queue manager you are deleting, delete the DefaultQueueManager key.
10. Navigate the tree to the following key:

```
HKEY_LOCAL_MACHINE\SOFTWARE\IBM\MQSeries\CurrentVersion\  
Configuration\Services
```
11. Within this key, delete the key whose name matches the subdirectory name of the queue manager which you are deleting.
12. Navigate the tree to the following key:

```
HKEY_LOCAL_MACHINE\SOFTWARE\IBM\MQSeries\CurrentVersion\  
Configuration\QueueManager
```
13. Within this key, delete the key whose name matches the subdirectory name of the queue manager which you are deleting.

Removing queue managers from the automatic startup list

If for any reason the WebSphere MQ Explorer cannot be used to change the startup state of a particular queue manager, use the following routine to carry out the same procedure manually:

1. Stop the WebSphere MQ Explorer either from the task bar icon or from the control panel.
2. Type REGEDIT on the command line.
3. Select the HKEY_LOCAL_MACHINE window.
4. Navigate the tree structure to find the following key:
LOCAL_MACHINE\Software\IBM\MQSeries\CurrentVersion\Configuration\Services\\QueueManager
5. Change the startup value to 0. (1 means automatic and 0 means manual.)
6. Close the Registry Editor.
7. Run `amqmdain regsec`.

Removing queue managers in WebSphere MQ for UNIX systems

The 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, to completely remove a queue manager, you must delete files, shared memory, and semaphores.

If you need to delete a queue manager manually, use the following procedure:

1. Stop the queue manager running, and execute the following command, as user `mqm`:

```
amqiclen -x -m QMGR
```

This ensures that all IPC resources that are specifically reserved for queue manager `QMGR` are removed.

2. Locate the queue manager directory from the configuration file `/var/mqm/mqs.ini`. To do this, look for the QueueManager stanza naming the queue manager to be deleted.
Its Prefix and Directory attributes identify the queue manager directory. For a Prefix attribute of `<Prefix>` and a Directory attribute of `<Directory>`, the full path to the queue manager directory is: `<Prefix>/qmgrs/<Directory>`
3. Locate the queue manager log directory from the `qm.ini` configuration file in the queue manager directory. The LogPath attribute of the Log stanza identifies this directory.
4. Delete the queue manager directory, all subdirectories and files.
5. Delete the queue manager log directory, all subdirectories and files.
6. Remove the queue manager's QueueManager stanza from the `/var/mqm/mqs.ini` configuration file.
7. If the queue manager being deleted is also the default queue manager, remove the DefaultQueueManager stanza from the `/var/mqm/mqs.ini` configuration file.

Appendix E. Sample MQI programs and MQSC files

WebSphere MQ for HP OpenVMS provides a set of short sample MQI programs and MQSC command files that you can use and experiment with. These are described in the following sections:

- “MQSC command file samples”
- “C and COBOL program samples”
- “Miscellaneous tools” on page 290

MQSC command file samples

Table 27 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 34.

By default, these files are located in directory MQS_EXAMPLES:

Table 27. MQSC command files

File name	Purpose
AMQSCOS0.TST	Creates a set of MQI objects for use with the C and COBOL program samples.

C and COBOL program samples

Table 28 lists the sample MQI source files. By default, the source files are located in directory MQS_EXAMPLES: and the compiled versions in [.BIN] directory under MQS_EXAMPLES:. To find out more about what the programs do and how to use them, see the *WebSphere MQ Application Programming Guide*.

Table 28. Sample programs - source files

C	COBOL	Purpose
AMQSBCG0.C	–	Reads and then outputs both the message descriptor and message context fields of all the messages on a specified queue.
AMQSECHA.C	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.

Table 28. Sample programs - source files (continued)

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.

Miscellaneous tools

These tool files are provided to support the formatter and code conversion.

Table 29. Miscellaneous files

File name	Location	Purpose
AMQTRC.FMT	SYS\$LIBRARY	Defines WebSphere MQ trace formats.
CCSID.TBL	MQS_ROOT:[MQM.CONV.TABLE]	Edit this file to add any newly supported CCSID values to your WebSphere MQ system. For more information about CCSID, see the CDRA (Character Data Representation Architecture) documentation.

Command file for application triggering

The command file MQTRIGGER.COM is supplied as an example of a command file designed to take the parameters supplied by the WebSphere MQ trigger monitor (RUNMQTRM) and separate the fields in the MQTMC2 structure.

The command file expects the first parameter to be the image, or command file, to invoke with selected fields from the MQTMC2 structure.

MQTRIGGER passes the following fields from the MQTMC2 structure to the invoked image or command file:

Parameter	MQTMC2 Field
1	QName
2	ProcessName
3	TriggerData
4	ApplType
5	UserData
6	QMgrName

Examples

- To trigger the amqsech image:
The ApplId field of the trigger process definition is specified as follows:


```
APPLICID('@mqs_examples:mqtrigger $mqbin:amqsech')
```

This example assumes that the MQBIN logical directory has been defined as:

```
SYS$SYSROOT:[SYSHLP.EXAMPLES.MQSERIES.BIN]
```

2. To invoke a command file, dka200:[user]cmd.com:

The ApplicId field of the trigger process definition is specified as follows:

```
APPLICID('@mqs_examples:mqtrigger @dka200:[user]cmd')
```

Appendix F. OpenVMS cluster failover set templates

This appendix contains the following failover set templates:

- “Template Configuration File FAILOVER.TEMPLATE”
- “Template StartCommand procedure START_QM.TEMPLATE” on page 294
- “Template EndCommand procedure END_QM.TEMPLATE” on page 295
- “Template TidyCommand procedure TIDY_QM.TEMPLATE” on page 297

Template Configuration File FAILOVER.TEMPLATE

```
#####  
#*                                     *#  
#* Statement:      Licensed Materials - Property of IBM          *#  
#*                                     *#  
#*                (C) Copyright IBM Corp. 2000, 2004           *#  
#*                                     *#  
#####  
#  
# FAILOVER.TEMPLATE  
# Template for creating a FAILOVER.INI configuration file  
# All lines beginning with a '#' are treated as comments  
#  
# OpenVMS Cluster Failover Set Configuration information  
# -----  
#  
# The TCP/IP address used by the OpenVMS Cluster Failover Set  
#  
IpAddress=n.n.n.n  
#  
# The TCP/IP port number used by the WebSphere MQ Queue Manager  
#  
PortNumber=1414  
#  
# The timeout used by the EndCommand command procedure  
#  
TimeOut=30  
#  
# The command procedure used to start the Queue Manager  
#  
StartCommand=@sys$manager:start_qm  
#  
# The command procedure used to end the Queue Manager  
#  
EndCommand=@sys$manager:end_qm  
#  
# The command procedure used to tidy up on a node after a  
# Queue Manager failure but the OpenVMS node did not fail  
#  
TidyCommand=@sys$manager:tidy_qm  
#  
# The directory in which the log files for the start, end and  
# tidy commands are written  
#  
LogDirectory=mqs_root:[mqm.errors]  
#  
# The number of nodes in the OpenVMS Cluster Failover Set. The  
# number of nodes defined below must agree with this number  
#  
NodeCount=2  
#  
# The Name of the OpenVMS node
```

```

#
NodeName=BATMAN
#
# The TCP/IP interface name for the node
#
Interface=we0
#
# The priority of the node
#
Priority=1
#
# The Name of the OpenVMS node
#
NodeName=ROBIN
#
# The TCP/IP interface name for the node
#
Interface=we0
#
# The priority of the node
#
Priority=2

```

Figure 23. Template configuration file: failover.template

Template StartCommand procedure START_QM.TEMPLATE

```

$ on error then exit
$!*****
$!* Statement:      Licensed Materials - Property of IBM      *
$!*                (C) Copyright IBM Corp. 2000, 2004      *
$!*****
$! Template command procedure used by Failover Sets to start the
$! queue manager
$! Parameters :
$! P1 = Queue Manager Name
$! P2 = Queue Manager Directory Name
$! P3 = TCP/IP address
$! P4 = TCP/IP interface name
$! P5 = Listener port number
$!
$ @sys$startup:mqs_symbols
$ set def mqs_root:[mqm.qmgrs.'p2'.errors]
$ define sys$scratch mqs_root:[mqm.qmgrs.'p2'.errors]
$!
$! Digital TCP/IP Services for OpenVMS commands
$!
$ @sys$startup:tcpip$define_commands
$!
$! Configure the IP address
$!
$ ifconfig 'p4' alias 'p3'
$!
$! TCPware for OpenVMS commands
$!
$! @tcpware:tcpware_commands
$!
$! Configure the IP address
$!
$! netcu add secondary 'p3'
$!
$! MultiNet for OpenVMS commands
$!
$! Configure the IP address

```

```

$!
$! define/sys/exec multinet_ip_cluster_aliases "'p3'"
$!
$! Restart the Multinet server
$!
$! @multinet:start_server
$!$! Start the queue manager
$!
$ strmqm 'p1'
$!
$! Start the listener
$!
$! runmqlsr -t tcp -p 'p5' -m 'p1'
$!
$! Insert commands to start any applications
$!
$exit

```

Figure 24. Template StartCommand procedure: Start_QM.template

Template EndCommand procedure END_QM.TEMPLATE

```

$ on error then exit
$!
$!*****
$!*
$!* Statement:      Licensed Materials - Property of IBM
$!*
$!*                (C) Copyright IBM Corp. 2000, 2004
$!*
$!******
$!
$! Template Command procedure used by Failover Sets to end the
$! queue manager
$!
$! Parameters :
$!
$! P1 = Queue Manager Name
$! P2 = Queue Manager Directory Name
$! P3 = TCP/IP address
$! P4 = TCP/IP interface name
$! P5 = Listener port number
$! P6 = End Queue Manager Timeout
$!
$ @sys$startup:mqs_symbols
$ check_qm==$sys$system:mqcheckqm
$ set def mqs_root:[mqm.qmgrs.'p2'.errors]
$ define sys$scratch mqs_root:[mqm.qmgrs.'p2'.errors]
$ SS$ _NORMAL=1
$ SS$ _ABORT=44
$ SS$ _TIMEOUT=556
$!
$! Insert commands to shutdown any applications prior to ending WebSphere MQ
$!
$! Get the timeout period for each operation seconds
$!
$ timeout = 'p6'
$!
$! Initialise the outer loop
$!
$ out_count = 0
$!
$! Initialise the complete flag
$!

```

```

$ complete = 0
$!
$ out_next:
$ if (out_count .gt. 2) .or. (complete .eq. 1) then goto out_finish
$!
$ if out_count .eq. 0
$ then
$!
$! End the queue manager gracefully first
$!
$ spawn/nosplit $endmqm -i 'p1'
$ else
$ if out_count .eq. 1
$ then
$!
$! End the queue manager abruptly
$!
$ spawn/nosplit $endmqm -p 'p1'
$ else
$!
$! Stop/id the execution controller
$!
$ check_qm -m 'p1'
$ if ( mqs$ec_pid .nes. "" ) then $stop/id='mqs$ec_pid'
$ endif
$ endif
$!
$ in_start:
$!
$! Initialise the outer loop
$!
$ in_count = 0
$!
$ in_next:
$!
$! Inner loop
$!
$ if ( ( in_count .ge. timeout) .and. ( timeout .ne. 0 ) ) -
    .or. (complete .eq. 1) then goto in_finish
$!
$! Check if the execution controller is still running
$!
$ check_qm -m 'p1'
$ if mqs$ec_pid .eqs. ""
$ then
$!
$! The Execution controller is no longer running so we are finished
$!
$ complete = 1
$ goto in_finish
$ endif
$!
$! Wait a second and go round again
$!
$ wait 00:00:01
$ in_count = in_count + 1
$ goto in_next
$ in_finish:
$!
$! End of the inner loop
$!
$ out_count = out_count + 1
$ goto out_next
$ out_finish:
$!
$! End of the outer loop
$!

```

```

$! Digital TCP/IP Services for OpenVMS commands
$!
$ @sys$startup:tcpip$define_commands
$!
$! De-configure the IP address
$!
$ ifconfig 'p4' -alias 'p3'
$!
$! TCPware for OpenVMS commands
$!
$! @tcpware:tcpware_commands
$!
$! De-configure the IP address
$!
$! netcu remove secondary 'p3'
$!
$! MultiNet for OpenVMS commands
$!
$! De-configure the IP address
$!
$! deass/sys/exec multinet_ip_cluster_aliases
$!
$! Restart the Multinet server
$!
$! @multinet:start_server
$!
$!
$! If the Queue Manager was shutdown successfully set the status
$! to SS$_NORMAL. If it was necessary to STOP/ID the Execution
$! controller set the status to SS$_ABORT and if the Execution
$! controller is still running set the status to SS$_TIMEOUT to
$! indicate an error
$!
$ if ( complete .eq. 1 )
$then
$!
$! End the listener process
$!
$! endmqlsr -m 'p1'
$!
$ if ( out_count .eq. 3 )
$ then
$ exit SS$_ABORT
$ else
$ exit SS$_NORMAL
$ endif
$else
$ exit SS$_TIMEOUT
$endif

```

Figure 25. Template EndCommand procedure: END_QM.template

Template TidyCommand procedure TIDY_QM.TEMPLATE

```

$ on error then exit
$!*****
$!* Statement:      Licensed Materials - Property of IBM      *
$!*
$!*                (C) Copyright IBM Corp. 2000, 2004      *
$!*****
$! Template Command procedure used by Failover Sets to tidy up after
$! a queue manager failure
$!
$! Parameters :

```

```

$! P1 = Queue Manager Name
$! P2 = Queue Manager Directory Name
$! P3 = TCP/IP address
$! P4 = TCP/IP interface name
$! P5 = Listener port number
$!
$ @sys$startup:mqs_symbols
$ set def mqs_root:[mqm.qmgrs.'p2'.errors]
$ define sys$scratch mqs_root:[mqm.qmgrs.'p2'.errors]
$!
$! Insert commands to do any tidying up after a queue manager has failed
$!
$! Digital TCP/IP Services for OpenVMS commands
$!
$ @sys$startup:tcpip$define_commands
$!
$! De-configure the IP address
$!
$ ifconfig 'p4' -alias 'p3'
$!
$! TCPware for OpenVMS commands
$!
$! @tcpware:tcpware_commands
$!
$! De-configure the IP address
$!
$! netcu remove secondary 'p3'
$!
$! MultiNet for OpenVMS commands
$!
$! De-configure the IP address
$!
$! deass/sys/exec multinet_ip_cluster_aliases
$!
$! Restart the Multinet server
$!
$! @multinet:start_server
$!
$exit

```

Figure 26. Template TidyCommand procedure: TIDY_QM.template

Appendix G. MONMQ diagnostic utility

The MONMQ utility is a tool to assist in the diagnosis and resolution of problems with WebSphere MQ for HP OpenVMS. The MONMQ utility can be used interactively, from the command line, or from within a DCL script.

The MONMQ utility is most commonly used to:

- Manage shared memory
- Help gather OpenVMS resource usage information
- Obtain trace output from a running queue manager.

MONMQ has a help system to assist with parameters and can also run a script of MONMQ commands. When MONMQ starts, a default script `sys$manager:mqs_trace_startup.mqt` is run to provide an initial configuration.

```
$monmq
ok - LU:0 opened

MQT> help
Help [ <verb> | <parameter/variable name> | commands | parameters | examples ]
Help [ expressions ]

Valid trace commands are in the format:
Verb [<parameters>] [<variable = expression>] [; [optional second command]]

A list of valid command verbs is displayed by entering
MQT> help commands
A list of valid parameters or variables can also be displayed by entering
MQT> help parameters
The expression can be numeric, a range of values, a string, or even an
arithmetic expression. The following command displays more information on
expressions
MQT> help expression
To obtain more help on a specific verb, parameter or variable enter the command
MQT> help <parameter or verb name>

Verb, parameter, or variable names can be shortened to any unique abbreviation

Commands that make use of a variable parameter can often use a default value
if it is not specified on the command line. The current default values can be
displayed with the "variables" command. The default can be changed with the
"default" command
```

Overview

Tracing WebSphere MQ on OpenVMS is implemented using global sections and mailboxes. Up to ten trace sections (LUs) can coexist on any one node where WebSphere MQ is installed. However, you should ensure that a trace session employs only one LU at any time. It is also not advisable for more than one user to have the same LU open at any one time. The results of either of these conditions are unpredictable.

Each shared section (LU) contains the channel definitions and the LU definition itself. Each channel definition contains the connected thread details, the threads

private stack and the threads circular buffer. Furthermore the shared section contains a set of flags used for interprocess communication between MONMQ and the connected threads.

For each LU there is an associated mailbox used for receiving realtime trace messages. To perform realtime tracing, a client process must be initiated using the **TRACE START** command. This dedicated detached process reads, formats and displays each message as it arrives in the LUs mailbox. Each connected thread writes to the same mailbox and thus provides you with the ability to physically view the intercommunication between WebSphere MQ processes/threads.

MONMQ, if driven correctly, can provide a comprehensive method for diagnosing problems such as, interprocess timing problems, exhausted operating system resources or even coding problems.

The MONMQ commands are described in this appendix.

Variables within MONMQ

Many commands within MONMQ make use of variables. A variable uses a default value, defined by the **set** command, if one is not specified within the command. When a variable is used with a command other than set, the default value for that variable is not changed.

Variables can contain:

- Integer variables (either decimal or hexadecimal).
Hexadecimal values can be entered with a leading 0x, or by entering a value with letters a-f where a hexadecimal value is expected.
- Text, which must be quoted.
- A range, which is entered by putting minimum:maximum.
A range is used so that, for example, a command can apply to a range of channels.

For example:

```
MQT> set lu=2
MQT> set pid=0x223
MQT> set pid=2fa
MQT> set buffile="filename.buf"
MQT> set chl=0:20
```

The current default value for the variables can be displayed by using the variables command.

```

MQT> variables
defined variables
lu=0:0          nochls=20          buffer=1000
chl=0:20       component=0(HEX)         line=0
mask=0(HEX)    pid=0(HEX)              node=(null)
function=0(HEX) div=0          depth=32
resource=0     wait=1(BOOL)           timestamp=0(BOOL)
listfile=(null) buffile=(null) step=0(BOOL)
active=0(BOOL) fname=(null)  delay=100
post=0(BOOL)

defined constants
fent=1(HEX)    fout=2(HEX)          ferr=4(HEX)
fxxx=8(HEX)   dgn=10(HEX)         shm=20(HEX)
spl=40(HEX)   evt=80(HEX)        mtx=100(HEX)
prc=200(HEX)  msc=400(HEX)       inf=800(HEX)
log=2000(HEX) shl=4000(HEX) memory=3
mutex=4        mailbox=5      nanoseconds=1
microseconds=2 milliseconds=3 seconds=4

```

Set default values to simplify the commands. For example, the following command sequences are functionally identical:

```

MQT> open lu=0 buffer=1000 nochls=20
MQT> open lu=1 buffer=1000 nochls=20
MQT> show channels lu=0 chl=1:10
MQT> show channels lu=1 chl=1:10

```

or

```

MQT> set nochls=20 chl=0:10 buffer=1000 lu=0
MQT> open
MQT> open lu=1
MQT> show channels
MQT> show channels lu=1

```

MONMQ commands can be abbreviated to the minimum number of characters required to ensure a unique command. This series of commands could be shortened still further to:

```

MQT> se noc=20 ch=0:10 buffe=1000 lu=0
MQT> op
MQT> op lu=1
MQT> sh ch
MQT> sh ch lu=1

```

MONMQ can also perform simple arithmetic operations with variables so that commands such as set lu=lu+1 is possible.

New variables can be declared with the **declare** command. The parameters are:

- Variable name
- Variable type
- Help text. The help command can then retrieve the help text.

```

MQT> declare ec int "channel number for execution controller"
MQT> set ec = 4
MQT> show channel chl=ec
Chl   Pid   Mailbox   Stack   Active   Post   Time   Mask   Process Name
 4    2c1f   7ee70290   4       0       0     0     ffffffff   AMQZMA0.EXE
MQT> help ec

VARIABLE ec:
channel number for execution controller

MQT>

```

Assigning default values

DEFAULT variable=<expression> [variable=expression] ...

This command allows default values to be assigned to all variables defined within MONMQ. Once set, the default variable name can then be omitted from the command line. For example:

```
MQT>default lu=2 chl=3:6
```

This command sets the default value 2 to the lu variable and the values 3 to 6 inclusive to the channel variable. From now on, when using a typical command such as **show channels**, channels 3 to 6 inclusive on lu 2 is displayed.

Default values are used only where the variable is omitted from the command line. All default values are set in the startup script file MQS_TRACE_STARTUP.MQT. This file can be edited to suit your needs.

Opening or creating a trace section and associated mailbox

OPEN [lu=number] [nochls=number] [buffer=number]

This command opens or creates a trace section and associated mailbox. The **open** command creates the basic resource required for tracing WebSphere MQ processes. Each LU has an associated shared section and mailbox used to communicate with WebSphere MQ processes.

This command takes three optional parameters. The first parameter [LU] is the number assigned to the trace section/mailbox and is used as a reference by most other MONMQ commands. A maximum of ten LUs may be created on a single node. The default value is zero. If the specified LU already exists then MONMQ connects to the existing trace section. If no section exists then a new section is created.

The second parameter [nochls] specifies the number of channels that this LU has. Each channel represents a single WebSphere MQ process/thread connection. The default value is 20.

The third parameter [buffer] specifies the maximum size of the trace history buffer for each channel. The default is 1000.

You must have at least one LU open before being able to perform other MONMQ commands.

Displaying the logical unit definition

SHOW SEGMENT [lu=*range*]

This command displays the Logical Unit definition. An example of the output is shown below with a brief description along side each field when you type the command **show segment lu=0**.

```
Trace LU           : 0           /* The LU number as specified in the OPEN [lu] parameter.
Mailbox name       : MQS_TRC_MBX_0 /* The permanent mailbox name assigned to this LU
Device name        : MBA1065:      /* The device name of the mailbox
Status             : Disabled      /* The current status of the mailbox ie.
Mailbox channel    : 352           /* The mailbox channel number assigned to the MONMQ process
History buffer size: 1000          /* The maximum number of message entries in the history
/* circular buffer (as specified by the OPEN [buffer] parameter)
Threads mapped #   : 1             /* The number of processes/threads mapped to this LUs global
/* section (MONMQ always attached)
Time stamping      : Enabled       /* Global timestamp flag (not yet implemented)
Max channels #     : 20            /* Number of channels defined for this LU as specified by the
/* OPEN [nochls] parameter.
Display depth      : 0             /* The stack display depth. Default (0) is to display all stack entries.
Text filename      :               /* The client text trace file
Binary filename    :               /* The client binary trace file
Last status        : 1             /* Last status of mailbox Qio activity (useful if VMS low on
/* resources and MONMQ fails)
Connection map[0] : 0             /* A bit map of all connected channels (maximum no. of channels is 128)
```

Closing and deleting an LU

CLOSE [lu=*number*]

This command performs the opposite to **OPEN** and closes and deletes the specified LU. The LU is closed in a controlled sequence by first signalling each connected process to disconnect, then resetting each channel and then finally deassigning the trace mailbox and deleting the shared section. This command should only be performed when a trace session has been completed.

Display channel details

SHOW CHANNELS [full] [connect] [chl=*range*]

This command displays the details of the specified channels. The [connected] parameter causes only channels that have a thread connected to be displayed. For example, the show channels connected command displays the following:

Ch1	Pid/Tid	Mailbox	Stack	History	RTime	Time	Mask	Process Name
0	00000245/1	800b0330	4	0	0	0	fffffff	AMQZLAA0.EXE
1	00000244/1	800b0200	8	0	0	0	fffffff	RUNMQCHI.EXE
2	00000243/1	800b01e0	10	0	0	0	fffffff	AMQRRMFA.EXE
3	00000242/1	800b01c0	4	0	0	0	fffffff	AMQZLLP0.EXE
4	00000241/1	800b0220	5	0	0	0	fffffff	AMQHASM0.EXE
5	00000240/1	800b03f0	4	0	0	0	fffffff	AMQZXMA0.EXE

The [full] parameter displays the complete definition of the specified channels. For example, the **show channels full connected chl=0:3** command displays:

```

Pid/Tid          : 0000024b/1      /* Connected threads process id and thread sequence number
Status           : ** Connected ** /* Current status of channel (thread is connected)
Process name     : AMQRRMFA.EXE   /* Process name of connected thread
Assigned LU      : 0              /* This channels associated LU
Channel no.      : 2              /* Allocated channel number within the LU
Mailbox channel  : 800b01e0       /* Connected threads mailbox channel number for the trace mailbox
Current stack depth : 10          /* Threads current stack depth
Circular logging : Disabled        /* History enabled flag
Next log entry   : 0              /* Next history buffer slot number
Realtime tracing : Disabled        /* Real time enable flag (needs client to read messages)
Time stamping    : Disabled        /* Enables timestamping for this threads messages
Trace mask       : ffffffff        /* Hexadecimal format of trace mask for this thread (see show mask command)
Step mode        : Off             /* Not yet implemented
No Wait          : On              /* Forces threads qio activity to wait for a resource if not available
Last QIO status  : 0              /* Threads last qio call status
Mapped address   : 9a2000-c9ffff   /* The virtual mapped address range of the LU global section for this thread

```

Display the current trace mask for a channel

SHOW MASK [chl=range]

This command displays the current trace mask for a channel. A highlighted line indicates that the btrace mask bit is enabled. For example, the command, **show mask chl=1** displays:

Trace Mask for Channel 1

Bit 00 - (fent) function entry	Function entry messages
Bit 01 - (fout) function exit	Function exit messages
Bit 02 - (ferr) function exit with error	Function exit with error return status
Bit 03 - (fxx) missing function exit	Unbalanced function entry/exit message (see note below)
Bit 04 - (dgn) diagnostic messages	Diagnostic messages
Bit 05 - (shm) shared memory	OVMS shared memory messages
Bit 06 - (spl) spinlocks	OVMS spinlock messages
Bit 07 - (evt) events	OVMS event messages
Bit 08 - (mtx) mutexes	OVMS mutex messages
Bit 09 - (prc) process msgs	OVMS thread messages
Bit 10 - (msc) miscellaneous	OVMS kernel miscellaneous messages
Bit 11 - (inf) informational	Internal data messages as requested by show command
Bit 12 -	Reserved for user defined messages

This output shows that function entry, function exit, spinlocks and event messages are traced for this thread. All other types of messages are blocked.

Display the contents of the target threads stack

SHOW STACK [chl=range]

This command displays the contents of the target threads stack. For example, the command **show stack chl=0:1** displays:

```
0001- 00:00:00.00 03 - 01 -->| ExecCtrlrMain
0002 - 12:36:20.18 03 - 02 ---->| zcpReceiveOnLink
0003 - 12:36:20.81 03 - 03 ---->| xcsWaitEventSem
0004 - 12:36:20.83 03 - 04 ----->| vms_evt

0001- 00:00:00.00 03 - 01 -->| ExecCtrlrMain
0002 - 12:36:20.18 03 - 02 ---->| zcpSendOnLink
0003 - 12:36:20.81 03 - 03 ---->| xcsPostEventSem
0004 - 12:36:20.83 03 - 04 ----->| vms_evt
```

Display active WebSphere MQ related processes and memory usage

SHOW PROCESSES

This command displays all active WebSphere MQ related processes on the current node along with their memory usage. For example, the command **show process** displays:

PID	Proc_Name	Image	Process	WS_Size	WS_Peak	Virt_Peak	Gbl_Pg_Cnt	Prc_Pg_Cnt	Total_Mem
0000023D	BKM3_AG	AMQZLAA0	Agent	23152	16576	203776	3616	12960	16576
0000023C	BKM3_CI	RUNMQCHI	Run Chan Init	8752	6208	180832	1840	4368	6208
0000023B	BKM3_RM	AMQRRMFA	Repository Mgr	11152	8144	185360	2224	5920	8144
0000023A	BKM3_CP	AMQZLLP0	Checkpoint	8752	6384	185952	1920	4464	6384
00000239	BKM3_LG	AMQHASM	Logger	8752	6288	182016	2080	4208	6288
00000238	BKM3_EC	AMQZXMA0	EC	20752	15232	203792	3536	11680	15216
00000128	_FTA4:	MONMQ	MONMQ Utility	8400	8528	198736	2224	3584	5808

Displays all messages held in a channel

SHOW HISTORY [chl=*range*]

This command displays all messages held in the channel circular history buffer. Each message is formatted and the output is indented according to the stack depth at which it was generated. For example, the command **show history chl=3** displays:

```

0215 - 12:35:44.52 03 - 02 ----<| zxcProcessChildren
0216 - 12:35:44.55 03 - 02 ---->| zxcStartWLMServer
0217 - 12:35:44.57 03 - 02 ----<| zxcStartWLMServer
0218 - 12:35:44.59 03 - 02 ---->| zcpReceiveOnLink
0219 - 12:35:44.61 03 - 03 ---->| xcsRequestMutexSem
0220 - 12:35:44.63 03 - 04 ---->| xllSemReq
0221 - 12:35:44.66 03 - 05 ---->| vms_mtx
0222 - 12:35:44.66 03 - 05 .....| vms_mtx :- Locking BKM3/@ipcc_m_1_10 - timeout: -1
0223 - 12:35:44.70 03 - 06 ---->| vms_get_lock
0224 - 12:35:44.72 03 - 06 ----<| vms_get_lock
0225 - 12:35:44.74 03 - 05 ----<| vms_mtx
0226 - 12:35:44.76 03 - 04 ----<| xllSemReq
0227 - 12:35:44.79 03 - 03 ----<| xcsRequestMutexSem
0228 - 12:35:44.81 03 - 03 ---->| xcsResetEventSem
0229 - 12:35:44.83 03 - 04 ---->| vms_evt
0230 - 12:35:44.83 03 - 04 .....| vms_evt Reset on mailbox BKM3/@ipcc_e_1_2 : tout = -1
0231 - 12:35:44.87 03 - 05 ---->| vms_get_mbx_chan
0232 - 12:35:44.87 03 - 05 .....| vms_get_mbx_chan Getting mbx BKM3/@ipcc_e_1_2
0233 - 12:35:44.87 03 - 05 .....| vms_get_mbx_chan Returning key 1a0
0234 - 12:35:44.94 03 - 05 ----<| vms_get_mbx_chan
0235 - 12:35:44.83 03 - 04 .....| vms_evt rc = 0
0236 - 12:35:44.98 03 - 04 ----<| vms_evt
0237 - 12:35:45.00 03 - 03 ----<| xcsResetEventSem
0238 - 12:35:45.03 03 - 03 ---->| xcsReleaseMutexSem
0239 - 12:35:45.05 03 - 04 ---->| xllSemRel
0240 - 12:35:45.07 03 - 05 ---->| vms_mtx
0241 - 12:35:45.07 03 - 05 .....| vms_mtx :- Unlocking BKM3/@ipcc_m_1_10 - timeout: -1
0242 - 12:35:45.11 03 - 06 ---->| vms_get_lock
0243 - 12:35:45.13 03 - 06 ----<| vms_get_lock
0244 - 12:35:45.16 03 - 05 ----<| vms_mtx

```

This sample output shows the line number within the history buffer, the time the message was generated, the channel number, the stack depth when the message was generated and the name of the function. When the LU was opened, the maximum number of history messages entries was defined. When this buffer is full, MONMQ wraps back to the first entry and overwrites the first and subsequent messages. While tracing, if an FFST is generated, then at the point of failure tracing is disabled for the failing thread. This is to prevent trace messages generated by error routines from filling the buffer. Therefore the last message displayed in the history buffer is the point at which the FFST was generated.

Display all WebSphere MQ related global sections on the current node

SHOW GLOBALS

This command displays all WebSphere MQ related global sections on the current node.

MQS1_shm_00000000 (00000000)	WRT	DZRO PRM SYS	Pgl tcnt/Refcnt=6128/383
MQS1_shm_01300010 (00000000)	WRT	DZRO PRM SYS	Pgl tcnt/Refcnt=1904/119
MQS1_shm_012c000f (00000000)	WRT	DZRO PRM SYS	Pgl tcnt/Refcnt=464/87
MQS1_shm_012c000e (00000000)	WRT	DZRO PRM SYS	Pgl tcnt/Refcnt=4112/514
MQS1_shm_012c000d (00000000)	WRT	DZRO PRM SYS	Pgl tcnt/Refcnt=240/30
MQS1_shm_012c000c (00000000)	WRT	DZRO PRM SYS	Pgl tcnt/Refcnt=528/132
MQS1_shm_012c000b (00000000)	WRT	DZRO PRM SYS	Pgl tcnt/Refcnt=272/51
MQS1_shm_012c000a (00000000)	WRT	DZRO PRM SYS	Pgl tcnt/Refcnt=4112/771
MQS1_shm_012c0009 (00000000)	WRT	DZRO PRM SYS	Pgl tcnt/Refcnt=272/51
MQS1_shm_012c0008 (00000000)	WRT	DZRO PRM SYS	Pgl tcnt/Refcnt=144/27
MQS1_shm_012c0007 (00000000)	WRT	DZRO PRM SYS	Pgl tcnt/Refcnt=528/99
MQS1_shm_012c0006 (00000000)	WRT	DZRO PRM SYS	Pgl tcnt/Refcnt=16/2
MQS1_shm_012c0005 (00000000)	WRT	DZRO PRM SYS	Pgl tcnt/Refcnt=128/24
MQS1_shm_012c0004 (00000000)	WRT	DZRO PRM SYS	Pgl tcnt/Refcnt=1968/492
MQS1_shm_012c0003 (00000000)	WRT	DZRO PRM SYS	Pgl tcnt/Refcnt=1904/476
MQS1_shm_012c0002 (00000000)	WRT	DZRO PRM SYS	Pgl tcnt/Refcnt=304/76
MQS1_shm_012c0001 (00000000)	WRT	DZRO PRM SYS	Pgl tcnt/Refcnt=1904/595
MQS1_shm_01280000 (00000000)	WRT	DZRO PRM SYS	Pgl tcnt/Refcnt=16/6
MQS1_shm_fffffffe (00000000)	WRT	DZRO PRM SYS	Pgl tcnt/Refcnt=16/0
MQS1_shm_fffffff (00000000)	WRT	DZRO PRM SYS	Pgl tcnt/Refcnt=144/63

Signals target thread to send mutex table to client trace process

SHOW MUTEX [chl=*range*]

This command signals the target thread to send the contents of its internal mutex table to the client trace process. Note that it is important that the correct trace mask bits are set to enable this type of informational data to be displayed by the client. Bits INF and DGN must be enabled in the trace mask for this channel. (See “Operator messages” on page 193.) For example, the command **show mutex chl=2** displays:

Mutex Utilisation for Process AMQZMA0.EXE - Pid 248 ***

```
0960 - Lock ID: 0100013c - Name: BKM3/@ipcc_m_1_24
0961 - Lock ID: 020006ca - Name: BKM3/@ipcc_m_1_23
0962 - Lock ID: 0b00061e - Name: BKM3/@ipcc_m_1_22
0963 - Lock ID: 0900068e - Name: BKM3/@ipcc_m_1_21
0964 - Lock ID: 0b00032f - Name: BKM3/@ipcc_m_1_20
0965 - Lock ID: 210006eb - Name: BKM3/@ipcc_m_1_19
0966 - Lock ID: 07000742 - Name: BKM3/@ipcc_m_1_18
0967 - Lock ID: 1e000075 - Name: BKM3/@ipcc_m_1_17
0968 - Lock ID: 0c0004dd - Name: BKM3_m_1_45
0969 - Lock ID: 0d00035a - Name: BKM3/@ipcc_m_1_16
0970 - Lock ID: 190000a1 - Name: BKM3/@ipcc_m_1_15
0971 - Lock ID: 1a0005a3 - Name: BKM3/@ipcc_m_1_14
0972 - Lock ID: 14000628 - Name: BKM3/@ipcc_m_1_13
0973 - Lock ID: 130005f3 - Name: BKM3/@ipcc_m_1_12
0974 - Lock ID: 0f0000dc - Name: BKM3_m_1_43
0975 - Lock ID: 02000095 - Name: BKM3_m_1_42
0976 - Lock ID: 2200053e - Name: BKM3_m_1_41
0977 - Lock ID: 020000fc - Name: BKM3_m_1_40
0978 - Lock ID: 31000113 - Name: BKM3_m_1_39
0979 - Lock ID: 02000555 - Name: BKM3_m_1_38
0980 - Lock ID: 2e000389 - Name: BKM3_m_1_37
0981 - Lock ID: 2300011f - Name: BKM3_m_1_36
0982 - Lock ID: 02000109 - Name: BKM3_m_1_35
0983 - Lock ID: 02000327 - Name: BKM3_m_1_34
0984 - Lock ID: 020004a8 - Name: BKM3_m_1_33
0985 - Lock ID: 02000453 - Name: BKM3_m_1_32
0986 - Lock ID: 260007ad - Name: BKM3_m_1_31
0987 - Lock ID: 0200060c - Name: BKM3_m_1_30
```

The data shows the line number in the history file, the mutex name and the system lock id.

Signals target thread to send internal events table to client trace process

SHOW EVENTS [chl=*range*]

This command signals the target thread to send the contents of its internal events table to the client trace process. Note that it is important that the correct trace mask bits are set to enable this type of informational data to be displayed by the client. Bits INF and DGN must be enabled in the trace mask for this channel. (See "Operator messages" on page 193.) For example, the command **show events chl=2** displays:

Event Utilisation for Process AMQZXMA0.EXE - Pid 248 ***

```
1037 - Channel: 000003e0 - Name: BKM3/@ipcc_e_1_19
1038 - Channel: 000003d0 - Name: BKM3/@ipcc_e_1_18
1039 - Channel: 000003c0 - Name: BKM3/@ipcc_e_1_17
1040 - Channel: 000003b0 - Name: BKM3/@ipcc_e_1_14
1041 - Channel: 000003a0 - Name: BKM3/@ipcc_e_1_13
1042 - Channel: 00000390 - Name: BKM3/@ipcc_e_1_12
1043 - Channel: 00000380 - Name: BKM3/@ipcc_e_1_10
1044 - Channel: 00000370 - Name: BKM3/@ipcc_e_1_9
1045 - Channel: 00000360 - Name: BKM3/@ipcc_e_1_8
1046 - Channel: 00000330 - Name: BKM3/@ipcc_e_1_7
1047 - Channel: 00000300 - Name: BKM3/@ipcc_e_1_6
1048 - Channel: 000002f0 - Name: BKM3/@ipcc_e_1_5
1049 - Channel: 000002b0 - Name: BKM3_e_1_11
1050 - Channel: 000002a0 - Name: BKM3_e_1_10
1051 - Channel: 00000290 - Name: BKM3_e_1_9
1052 - Channel: 00000280 - Name: BKM3_e_1_8
1053 - Channel: 00000270 - Name: BKM3_e_1_7
1054 - Channel: 00000260 - Name: BKM3_e_1_6
1055 - Channel: 00000250 - Name: BKM3_e_1_5
1056 - Channel: 00000240 - Name: BKM3_e_1_4
1057 - Channel: 00000230 - Name: BKM3_e_1_3
1058 - Channel: 00000220 - Name: BKM3_e_1_2
1059 - Channel: 00000210 - Name: BKM3_e_1_1
1060 - Channel: 00000200 - Name: BKM3_e_1_0
1061 - Channel: 000001c0 - Name: BKM3/@ipcc_e_1_4
1062 - Channel: 000001b0 - Name: BKM3/@ipcc_e_1_3
1063 - Channel: 000001a0 - Name: BKM3/@ipcc_e_1_2
1064 - Channel: 00000190 - Name: BKM3/@ipcc_e_1_1
1065 - Channel: 00000180 - Name: BKM3/@ipcc_e_1_0
1066 - *** End of data ***
```

The data shows the line number in the history file, the mailbox channel number and the event name.

Signals target thread to send internal mapped shared memory table to the client trace process

SHOW MEMORY [chl=*range*]

This command signals the target thread to send the contents of its internal mapped shared memory table to the client trace process. Please note that it is important that the correct trace mask bits are set to enable this type of informational data to be received by the client. Bits INF and DGN must be enabled in the trace mask for this channel. (See “Operator messages” on page 193.) For example, the command **show memory chl=2** displays:

*** Shared Memory Utilisation for Process AMQZXMA0.EXE - pid/tid 248-1 ***

```
0942 - ShmId: 0248000f - Addr: 011d8000/01211fff - Perm: 950 - Size: 00038530 Name: /mqs_root/mqm/qmgrs/BKM3/@ipcc/shmem/AMQ
0943 - ShmId: 0248000e - Addr: 00fbc000/011bdfff - Perm: 950 - Size: 002005f8 Name: /mqs_root/mqm/qmgrs/BKM3/@ipcc/shmem/AMQ
0944 - ShmId: 0248000d - Addr: 00f1c000/00f39fff - Perm: 950 - Size: 0001d478 Name: /mqs_root/mqm/qmgrs/BKM3/@ipcc/shmem/WLM
0945 - ShmId: 0248000c - Addr: 00cba000/00cfbfff - Perm: 944 - Size: 000405f0 Name: /mqs_root/mqm/qmgrs/BKM3/shmem/HMEMSET.0
0946 - ShmId: 0248000b - Addr: 00c98000/00cb9fff - Perm: 944 - Size: 000205f0 Name: /mqs_root/mqm/qmgrs/BKM3/shmem/Anon005.0
0947 - ShmId: 0248000a - Addr: 00a96000/00c97fff - Perm: 944 - Size: 00200584 Name: /mqs_root/mqm/qmgrs/BKM3/shmem/Anon004.0
0948 - ShmId: 02480009 - Addr: 00a74000/00a95fff - Perm: 944 - Size: 000205f0 Name: /mqs_root/mqm/qmgrs/BKM3/shmem/Anon003.0
0949 - ShmId: 02480008 - Addr: 00a62000/00a73fff - Perm: 944 - Size: 000105f0 Name: /mqs_root/mqm/qmgrs/BKM3/shmem/Anon002.0
0950 - ShmId: 02480007 - Addr: 00a20000/00a61fff - Perm: 944 - Size: 000405f0 Name: /mqs_root/mqm/qmgrs/BKM3/shmem/Anon001.0
0951 - ShmId: 02480006 - Addr: 00908000/00909fff - Perm: 950 - Size: 00001664 Name: /mqs_root/mqm/qmgrs/BKM3/@ipcc/shmem/PLU
0952 - ShmId: 02480005 - Addr: 008f8000/00907fff - Perm: 950 - Size: 0000fff8 Name: /mqs_root/mqm/qmgrs/BKM3/@ipcc/shmem/IPC
0953 - ShmId: 02480004 - Addr: 00802000/008f7fff - Perm: 950 - Size: 000f4838 Name: /mqs_root/mqm/qmgrs/BKM3/@ipcc/shmem/IPC
0954 - ShmId: 02480003 - Addr: 00714000/00801fff - Perm: 950 - Size: 000ec718 Name: /mqs_root/mqm/qmgrs/BKM3/@ipcc/shmem/SUB
0955 - ShmId: 02480002 - Addr: 006ee000/00713fff - Perm: 944 - Size: 000253a8 Name: /mqs_root/mqm/qmgrs/BKM3/shmem/zutSESSAN
0956 - ShmId: 02480001 - Addr: 00600000/006edfff - Perm: 944 - Size: 000ec710 Name: /mqs_root/mqm/qmgrs/BKM3/shmem/SUBPOOL.0
0957 - ShmId: 01280000 - Addr: 005f8000/005f9fff - Perm: 950 - Size: 00000454 Name: /var/mqm/errors
0958 - *** End of data ***
```

The data shows the line number in the history file, shared memory id, the virtual mapped address range, the flags used in creating/mapping to the section and the internal WebSphere MQ name given to the section.

Displays active WebSphere MQ components by name and hexadecimal ids

SHOW COMPONENTS

This command displays all active WebSphere MQ components by name and their associated hexadecimal ids. Use these hex ids in other MONMQ show commands such as show functions and select component. For example, the command **show components** displays:

00000001 - Data hardening
00000002 - Log management
00000003 - Object Catalogue
00000004 - Queue management
00000005 - Transaction Management
00000006 - Mobile Component
00000007 - Mobile Component
00000008 - Communications
0000000a - Object Authority Manager
0000000b - Logger
0000000d - LQM Kernal
0000000f - Administration App
00000010 - Administration App
00000013 - Command Server
00000014 - Remote queue processor
00000015 - XA Transaction Manager
00000016 - Data Conversion
00000017 - Common Services
00000018 - Common Services (overflow)
00000019 - Application Interface
0000001a - IPCC
0000001b - DCE Support
0000001c - Pluggable Services
0000001d - Agent
0000001e - XA Transaction Manager
0000001f - C++ Layer
00000020 - CLI
00000021 - Z Utilities
00000022 - Execution Controller
00000023 - App. Bindings
00000024 - Service Component
00000025 - Publish/Subscribe
00000026 - MMC Snap-in for Admin
00000027 - Web Administration
00000028 - KYG Services
00000029 - Thread Manager/MTS Support
0000002a - MS Cluster Server
0000002b - Active Directory support
0000002d - OVMS MQ kernel

Display functions within specified component

SHOW FUNCTIONS [comp=*hex*]

This command displays all functions within the specified component. The component must be entered in hex. Use SHOW COMPONENT to display all active WebSphere MQ components. For example, the command **show functions component=0x1f** displays:

```
00000000 - ImqBinary::copyOut
00000001 - ImqBinary::pasteIn
00000002 - ImqCache::operator =
00000003 - ImqCache::moreBytes
00000004 - ImqCache::read
00000005 - ImqCache::resizeBuffer
00000006 - ImqCache::setDataOffset
00000007 - ImqCache::setMessageLength
00000008 - ImqCache::useEmptyBuffer
00000009 - ImqCache::write
0000000a - ImqDeadLetterHeader::pasteIn
0000000b - ImqDistributionList::openInfoPrepare
0000000c - ImqItem::structureIds
0000000d - ImqQueueManager::backout
0000000e - ImqQueueManager::begin
0000000f - ImqQueueManager::commit
00000010 - ImqQueueManager::connect
00000011 - ImqQueueManager::disconnect
00000012 - ImqMessageTracker::setAccountingToken
00000013 - ImqMessageTracker::setCorrelationId
00000014 - ImqMessageTracker::setGroupId
00000015 - ImqMessageTracker::setMessageId
00000016 - ImqObject::close
00000017 - ImqObject::closeTemporarily
00000018 - ImqObject::inquire
00000019 - ImqObject::open
0000001a - ImqObject::openFor
.....
```

Activate tracing from the point a process starts

ONSTARTUP [ON] [lu=*number*] [chl=*range*]

This command allows tracing to be activated from the point a process starts. When executed, a logical name MQS_DEF_TRACE is defined in the system logical name table and has an equivalent name of the following format: *lu channel*. When any WebSphere MQ process starts, this logical name is checked inside the processes initialization routine and, if present, connects to the specified LU and channel number. If the channel id is already allocated then the next available channel is used. This command is useful when tracing is required during the early phases of WebSphere MQ process/thread creation.

Prevent WebSphere MQ process from tracing immediately from startup

ONSTARTUP [OFF]

This command deassigns the MQS_DEF_TRACE logical from the system logical name table and thus prevents WebSphere MQ processes from tracing immediately from startup.

Connect target thread to specified channel

CONNECT pid *number* [tid=*number*] [chl=*range*]

This command signals the target thread to connect to the specified channel. If no channel is specified then the first available channel is used.

Disconnect target thread to specified channel

DISCONNECT [chl=*range*]

This command signals the target thread to disconnect from the specified channel.

Display real-time trace message written to the LUs trace mailbox

TRACE START [node=*string*]

This command launches a client trace process to display the real-time trace message written to the LUs trace mailbox. The optional node parameter creates a window on the specified node and directs output to that window.

Detach and end current client process

TRACE STOP

This command causes the current client process to detach from the trace mailbox and end. All threads currently writing to this mailbox are disabled from writing messages.

```
MQT> trace stop
```

```
Circular buffering has been disabled for process 24d thread 1
Circular buffering has been disabled for process 24c thread 1
Circular buffering has been disabled for process 24b thread 1
Circular buffering has been disabled for process 248 thread 1
Disconnecting thread pid : 24d, tid : 1 from channel 0 ..... OK
Disconnecting thread pid : 24c, tid : 1 from channel 1 ..... OK
Disconnecting thread pid : 24b, tid : 1 from channel 2 ..... OK
Disconnecting thread pid : 248, tid : 1 from channel 3 .....OK
```

```
*** Trace ended - no processes connected ***
```

The above output appears on the trace client window.

Specify trace data

SELECT [component] AND/OR [function] OR [*fname*]

This command allows you to specify up to eight combinations of component/functions to be traced. All other trace data are filtered out. Either a function name can specified or a component or a component/function. The selected component/function if valid is written to the filter table. If no entries exist then ALL function component/functions are traced as the default.

Entering the **SELECT** command with no parameters causes the contents of the filter table to be displayed. Against each line of output is the table index entry and the component and function in hex and the text name of the function. If only a component is entered then all functions within this component are traced. This is shown as 0xffff against the function value.

For example, the command **SELECT** on its own displays:

```
Ch1:0 - Cmp/fnc selection criteria
ALL component/functions
```

The following set of commands,

```
MQT>select fname="kill"
MQT>select comp=0x1f
MQT>select comp=0x20 func=0x3
MQT>select
```

displays:

```
Ch1:0 - Cmp/fnc selection criteria
Idx: 0 - Cmp: 00000029 - Fnc: 00000005 - Name - kill
Idx: 1 - Cmp: 00000019 - Fnc: 0000ffff - Name -
Idx: 2 - Cmp: 00000016 - Fnc: 00000003 - Name - vqiAddCacheEntry
```

Remove single entry from the trace filter table

DESELECT INDEX=<0:7>

This command removes a single entry from the trace filter table as specified by the table index parameter. All components/functions are traced when all eight entries are empty. For example, a **select** command displays the following entries with their indexes:

```
Ch1:0 - Cmp/fnc selection criteria
Idx: 0 - Cmp: 00000029 - Fnc: 00000005 - Name - kill
Idx: 1 - Cmp: 00000019 - Fnc: 0000ffff - Name -
Idx: 2 - Cmp: 00000016 - Fnc: 00000003 - Name - vqiAddCacheEntry
```

The following deselect commands remove the specified processes or functions:

```
MQT> dese1 index=0
MQT> dese1 index=2

MQT>select
Ch1:0 - Cmp/fnc selection criteria
Idx: 1 - Cmp: 00000019 - Fnc: 0000ffff - Name -
-----
MQT> deselect index=1
MQT> select

Ch1:0 - Cmp/fnc selection criteria
ALL component/functions
-----
```

Client process writes trace messages to a binary file

OPEN BINARY [filename=*string*]

This command opens a trace message binary file and causes the client process to write realtime trace messages to this file. This file can be later used for analyzing performance of WebSphere MQ applications. The default filename is `mqs_root:[mqm.errors]mqs_buffer_xx.bin` (where *xx* is the LU number).

Close binary trace messages file

CLOSE BINARY

This command closes the specified LU binary trace file.

Client process writes trace messages to a text file

OPEN TEXT [filename=*string*]

This command opens a readable text file and causes the client process to write formatted binary trace messages to this file. This file can be viewed later by simply using the DCL type command or edit. The default filename is `mqs_root:[mqm.errors]mqs_buffer_xx.lis` (where *xx* is the LU number). The advantages of using this type of output file is that it requires no preprocessing for it to be read. However the disadvantage is that it consumes more disk space than a binary file.

Close text trace messages file

CLOSE TEXT

This command closes the specified LU text trace file.

Timestamp messages

ENABLE TIMESTAMP [chl=*range*]

This command sets the Timestamp flag in the channel definition table. Use this command to force WebSphere MQ processes to stamp each message with the current time. This flag has to be set when using a binary trace file for performance analysis.

Stop timestamping messages

DISABLE TIMESTAMP [chl=*range*]

This command unsets the Timestamp flag in the channel definition table. (See "Timestamp messages.")

Enable tracing

ENABLE TRACE [chl=*range*]

This command sets the RTime trace flag in the channel definition table. Use this flag to enable and disable the sending of trace messages to a trace client. When a thread is connected and a trace client is present, for example, TRACE START can be used to switch the channel in or out rather than disconnecting this thread.

Disable tracing

DISABLE TRACE [chl=*range*]

This command unsets the RTime trace flag in the channel definition table. (See “Enable tracing” on page 315.)

Save message history

ENABLE HISTORY [chl=*range*]

This command sets the History flag in the channel definition table for the specified channels. This command signals the connected thread to write trace messages to the LUs circular buffer. As the writing of the message is performed by the traced process then it is not necessary for a client process to exist. The size of the trace circular buffer is defined during LU creation by the **open** command. This buffer wraps back to the beginning when the last entry is written. The Next Log record field in the LU definition table specifies where the next record in the buffer is to be written.

Disable message history

DISABLE HISTORY [chl=*range*]

This command unsets the History flag in the channel definition table for the specified channels. See “Save message history.”

Delete message history

DELETE HISTORY [chl=*range*]

This command deletes all messages in the circular history buffer. This command can be performed even when there are processes writing to the buffer so it is not necessary to disable history before deleting.

Set history depth

SET [depth]

This command controls the maximum stack depth to be output to the trace client window. Messages deeper than this value are not output: however, they do appear in the binary trace file and history buffer if enabled. The default value of zero allows all messages at whatever stack depth to be output. Users should set this to a very low value (for example, 1) when writing analysis data to the binary file. Full stack display adversely affects the performance of a client process.

Reset stack and history data for a channel

```
SET [free] [chl=range]
```

This command resets the specified channel. All existing stack and history data is deleted and the channel is unallocated and available for reuse.

Enable or disable mask bit

```
SET [mask=var] [chl=range]
```

This command either enables or disables a mask bit within the connected threads bit mask field. Each bit represents a message type that is generated by a WebSphere MQ process. You can use this command to filter the type of messages that need to be traced. The message types are as follows:

```
MQT>set mask = 0xffffffff chl=1
MQT>show mask chl=1
```

```
Trace Mask for Channel 1
Bit 00 - (fent) function entry
Bit 01 - (fout) function exit
Bit 02 - (ferr) function exit with error
Bit 03 - (fxx) missing function exit
Bit 04 - (dgn) diagnostic messages
Bit 05 - (shm) shared memory
Bit 06 - (spl) spinlocks
Bit 07 - (evt) events
Bit 08 - (mtx) mutexes
Bit 09 - (prc) process msgs
Bit 10 - (msc) miscellaneous
Bit 11 - (inf) informational
Bit 12 -
```

To specify a combination of these message types delimit each mask type with an OR symbol for example:

```
MQT>set mask =0x0 chl=1 MQT>show mask chl=1
```

```
Trace Mask for Channel 1
Bit 00 - (fent) function entry
Bit 01 - (fout) function exit
Bit 02 - (ferr) function exit with error
Bit 03 - (fxx) missing function exit
Bit 04 - (dgn) diagnostic messages
Bit 05 - (shm) shared memory
Bit 06 - (spl) spinlocks
Bit 07 - (evt) events
Bit 08 - (mtx) mutexes
Bit 09 - (prc) process msgs
Bit 10 - (msc) miscellaneous
Bit 11 - (inf) informational
Bit 12 -
```

```
MQT>set mask = mtx | evt | fent chl=1
MQT>show mask chl=1
```

```
Trace Mask for Channel 1
Bit 00 - (fent) function entry
Bit 01 - (fout) function exit
Bit 02 - (ferr) function exit with error
Bit 03 - (fxx) missing function exit
Bit 04 - (dgn) diagnostic messages
Bit 05 - (shm) shared memory
Bit 06 - (spl) spinlocks
Bit 07 - (evt) events
Bit 08 - (mtx) mutexes
Bit 09 - (prc) process msgs
Bit 10 - (msc) miscellaneous
Bit 11 - (inf) informational
Bit 12 -
```

Each mask comprises of eight mask types which you can toggle to either enable or disable a particular message type. For example if you were interested only in function entry points then enter the command **set mask = fent**.

Set a color for a channel

SET COLOR [chl=*range*]

This command associates a color with the specified channel. All output related to this channel is displayed in this color until either the color is changed or the channel is reset. This command is useful for highlighting or distinguishing between different threads' messages within a single output stream. For example, the commands:

```
MQT> set color=yellow chl=2
MQT> set color=blue chl=0
MQT> sho chan chl=0:3 connected
```

displays:

Ch1	Pid/Tid	Mailbox	Stack	History	RTime	Time	Mask	Process Name
0	00000245/1	800b0330	4	0	0	0	ffffffff	AMQZLAA0.EXE
1	00000244/1	800b0200	8	0	0	0	ffffffff	RUNMQCHI.EXE
2	00000243/1	800b01e0	10	0	0	0	ffffffff	AMQRRMFA.EXE
3	00000242/1	800b01c0	4	0	0	0	ffffffff	AMQZLLP0.EXE

where Channel 0 is blue and Channel 2 is yellow.

Redirect output to file

SET OUTPUT [filename=*string*]

This command directs all output to the specified file and disables output to the display. The **output** command, when used as a parameter with other commands, is effective for that command only. Note that errors continue to be reported to the display device and not to file. Only valid trace data is written to the specified file.

Analyze trace binary file

ANALYSE [component] [function] [unit=*xx*]

This command analyzes the contents of trace binary file previously used in a trace session. Although you can specify the component or function to be analyzed, this is effective only if the file contains such data relating to this component or function. For example, if when the file was generated, you specified a trace mask or even selected a specific component, then only these selected items are found in the binary file and hence components or functions outside this criteria cannot be used in the analysis.

Note: If you are going to use the full command, spell it ANALYSE with a S. Do not confuse this MONMQ command with the OpenVMS command ANALYZE. The two commands are different from each other.

The unit parameter is used to specify the unit of time for the analysis and can have one of the following values (*xx*) - seconds, milliseconds, microseconds, nanoseconds. The default is milliseconds.

For example, to display output in microseconds, using the command **analyze unit=micro**: The following is a sample output for this command:

```

=====
                        COMPONENT :- Common Services
=====
Calls  Minimum      Average      Maximum      Total      Function
42     0.00         66.41       311.78      2789.15    xcsRequestMutexSem
42     0.00         96.98       222.64      4072.98    xcsReleaseMutexSem
6      0.00         138.50      286.11      831.00     xcsResetEventSem
5      0.00        10112.60    10159.51    50563.01   xcsWaitEventSem
6      0.00         564.74     1029.23     3388.45    xcsCheckExtendMemory
42     0.00         51.32       266.86     2155.41    xllSemReq
42     0.00         73.05       159.17     3068.16    xllSemRel
=====
                        COMPONENT :- Common Services (overflow)
=====
Calls  Minimum      Average      Maximum      Total      Function
36     0.00         41.15       122.06     1481.35    xcsCheckProcess
6      0.00         37.92       68.35      227.52     xihGetConnSPDetailsFromList
6      0.00         65.26       114.25     391.58     xihHANDLEtoSUBPOOLFn
6      0.00         12.69       23.44     1267.49    xihGetNextSetConnDetailsFromList
6      0.00         12.53       22.46       75.19     xcsRequestThreadMutexSem
6      0.00         12.37       22.46       74.21     xcsReleaseThreadMutexSem
=====
                        COMPONENT :- IPCC
=====
Calls  Minimum      Average      Maximum      Total      Function
5      0.00        10589.97    11029.84    52949.85   xcpReceiveOnLink
=====
                        COMPONENT :- CLI
=====
Calls  Minimum      Average      Maximum      Total      Function
6      0.00         1.95        1.95        11.72      zapInquireStatus
=====
                        COMPONENT :- Execution Controller
=====
Calls  Minimum      Average      Maximum      Total      Function
6      0.00         954.46     3275.18    11726.79   zxcProcessChildren
6      0.00         12.69       22.46       76.17     zxcStartWLMServer
=====
                        COMPONENT :- OVMS MQ kernel
=====
Calls  Minimum      Average      Maximum      Total      Function
36     0.00         15.49       99.60      557.58     kill
6      0.00         0.65        3.91        3.91       vms_mapgbl
84     0.00         11.17       88.16     938.68     vms_get_lock
84     0.00         42.41      221.94    3562.54    vms_mtx
12     0.00         44.51      149.40     534.14    vms_get_mbx_chan
11     0.00        4651.77    10137.05   51169.42   vms_evt
6      0.00         1.63        4.88        9.76       vms_check_health
=====

```

Columns are:

Calls The number of entries into the function during the trace session.

Minimum

This is the fastest time spent inside the function.

Average

This is the total time spent inside all calls to the function divided by the number of calls.

Maximum

The longest time spent inside the function.

Total The total time spent in this function for all calls.

Function

The function name.

Note: The scope of the analysis is the content of the binary trace file. It is up to the user to define the boundaries of the analysis by opening a trace binary file and enable/disable the trace at the desired time.

Display current state of WebSphere MQ threads

FFST [chl=*range*]

This command forces the thread connected to the target channel to force an FFST. This command does NOT effect the target threads path of execution. This command allows you to take a snapshot of any WebSphere MQ threads current state. The FFST cut contains the threads resource usage, privileges and other useful system information. The FFST is clearly marked in the header as having been created by MONMQ (see below) and is NOT a result of a failure.

The following is some sample output:

```
WebSphere MQ First Failure Symptom Report
=====
Date/Time      :- Friday Sep 17 10:59:38 GMT 2004
Host Name     :- CATWMN (Unknown)
PIDS          :- 5697175
LVLS          :- 530
Product Long Name :- WebSphere MQ for OpenVMS Alpha
Vendor        :- IBM
Probe Id      :- VM026000
Application Name :- MQM
Component     :- vms_evt
Build Date    :- July 22 2004 (Collector)
Userid        :- [400,400] (SYSTEM)
Program Name  :- AMQZXMA0.EXE
Process       :- 00000248
Thread        :- 00000001
QueueManager  :- BKM3
Major Errorcode :- xecF_E_UNEXPECTED_SYSTEM_RC
Minor Errorcode :- OK
Probe Type    :- MSGAMQ6119
Probe Severity :- 2
Probe Description :- AMQ6119: An internal WebSphere MQ error has occurred
                  (** FORCED FFST BY USER **)
Comment1      :- *** FORCED FFST BY USER ***
Comment2      :- -SYSTEM-S-NORMAL, normal successful completion
```

etc.....

Close trace and exit MONMQ

EXIT

This command performs a CLOSE command and exits MONMQ.

Quit MONMQ without closing trace

QUIT

This command does not perform a CLOSE command but exits MONMQ. This command is useful if you want to leave Trace running but want to shutdown MONMQ. The next time MONMQ is activated the previous trace session is resumed.

Managing shared memory with MONMQ

In unusual circumstances, for example, a queue manager failure or forced shutdown with the OpenVMS stop /id command, it is possible that WebSphere MQ shared memory segments are not automatically deleted by the queue manager. If this occurs it is not possible to restart the queue manager, because **strmqm** reports that the queue manager is already running.

MONMQ can list MQ shared memory (global sections) that currently exist, and can delete these shared memory sections.

Note: Ensure that all queue managers are shutdown before using the MONMQ utility to delete shared memory segments. Deleting the shared memory of a running queue manager causes the queue manager to fail, possibly corrupting the queue files.

The MONMQ SHOW PROCESS command can be used to ensure that there are no MQ processes running. If there are processes running on a failed queue manager that can not be stopped by the endmqm command then the OpenVMS DCL command stop /id=<pid> can be used.

Check there are no WebSphere MQ processes running by using the following command:

```
MQT> show process
```

```
MQ Processes
PID      Proc Name      Image      Process      WS Size  WS Peak  Virt Peak  Gbl Pg Cnt Prc Pg Cnt Total Mem
-----
List the shared memory global sections that currently exist

MQT> show globals
MQS1_shm_2695000a
(00000000) WRT   DZRO PRM SYS Pgltcnt/Refcnt=4112/514
MQS1_shm_26950009
(00000000) WRT   DZRO PRM SYS Pgltcnt/Refcnt=272/34
MQS1_shm_ffffffff
(00000000) WRT   DZRO TMP SYS Pgltcnt/Refcnt=144/63
MQS1_shm_00000000
(00000000) WRT   DZRO TMP SYS Pgltcnt/Refcnt=6304/394
```

List the shared memory global sections that currently exist by using the command:

```
MQT> show globals
```

Delete these global sections:

```
MQT> delete
Deleted global section: MQS1_shm_2695000a
Deleted global section: MQS1_shm_26950009
Deleted global section: MQS1_shm_ffffffff
sys$delgbl - unable to delete section MQS1_shm_00000000
```


The error deleting section MQS_shm_00000000 is expected since this section is used by MONMQ. You can now exit MONMQ by issuing the command:

```
MQT> exit
```

You can use the delete command from within a script if you are certain that all queue manager processes are stopped:

```
$ monmq delete
Deleted global section: MQS1_shm_2695000a
Deleted global section: MQS1_shm_26950009
Deleted global section: MQS1_shm_ffffffff
sys$delgbl - unable to delete section MQS1_shm_00000000
```

Scripts and macros in MONMQ

It is possible to run a script of MONMQ commands either from within MONMQ or from the command prompt. Scripts can be useful to collect a set of data, or to configure the MONMQ environment. When MONMQ starts, a script is run from SYS\$MANAGER:MQS_TRACE_STARTUP.MQT to configure the trace variables in MONMQ.

Note: If the script is not in the current directory, the full path name to the script must be quoted. For example:

```
MQT> ! "sys$manager:test.mqt"
```

It is also possible to define a macro to shorten common or repetitive tasks. A macro declaration consists of three parts:

1. The first part is the macro name, which must be a unique command name.
2. The second part is the macro body, which can span multiple lines and consists of a list of WebSphere MQ commands. The macro body is delimited by { and }. The MONMQ prompt changes to ****MACRO>** when a multiple line macro body is being declared. Any \$n, where n is a single digit number, is replaced with parameter n on the macro command line.
3. The third part of a macro definition is a short help text description that is displayed when help <macroname> is used. The help text must be quoted.

You must consider timing issues when declaring a macro. A macro processes very quickly, but some MONMQ commands signal a remote process to perform a task, and this task must be finished before the next macro command is started.

For this reason a short delay is sometimes required. You do this by using the **sleep** command, which has a delay parameter that is specified in tenths of a second.

The following commands can be entered to create a macro that disconnects a channel, resets the trace mask, and frees the channel.

Note: More than one MONMQ command can be placed on a line by using the ";" as a separator.

```

MQT> declare tmpchl intrange "variable to hold a chl range temporarily"
MQT> macro remove { set tmpchl = chl ; dis chl= $1 ; sleep delay=5
**MACRO> set mask=0xffffffff chl= $1 ; set free chl = $1
**MACRO> set chl=tmpchl
**MACRO> } "A macro to disconnect and free channels Param: chl number"
MQT> help remove

VERB remove:
A macro to disconnect and free channels Param: chl number

Macro text:
set tmpchl = chl ; dis chl= $1 ; sleep delay=5 ; set mask=0xffffffff chl=$1
; set free chl = $1 ; set chl=tmpchl
MQT>remove 4
ok - process disconnected process 282 from channel 4

```

Sample trace session

This section describes a typical trace session showing each MONMQ command in sequence. This sample is tracing a running queue manager's execution controller and its related agents main thread.

Before you begin the trace, the following conditions must be met:

- Start queue manager to be traced - STRMQM BKM3
- Check that sys\$manager:mqs_trace_startup.mqt has no additional commands apart from the preinstalled defaults.
- Check that the logical MQS_DEF_TRACE is NOT defined. If it is then perform an **ONSTARTUP END** in MONMQ.

Start momq.

```
>monmq
```

The MONMQ prompt is displayed:

```
MQT>
```

Open a single LU with an ID of zero with ten channels and a history buffer of 100 messages. (Note: 100 messages would be too small for normal tracing purposes. 1000 is normally adequate.)

```
MQT> open lu=0 nochls=10 buffer=100
ok - LU:0 opened
```

Display LU1 definition:

```
MQT> show seg lu=1
```

```
Trace LU           : 1
Mailbox name       : MQS_TRC_MBX_1
Device name        : MBA431:
Status             : Disabled
Mailbox channel    : 384
History buffer size : 100
Threads mapped #   : 1
Time stamping      : Enabled
Max channels #     : 10
Display depth      : 0
Text filename      :
Binary filename    :
Last status        : 1
Connection map[0]  : 0
=====
```

Display WebSphere MQ processes:

```
MQT> show process
```

PID	Proc_Name	Image	Process	WS_Size	WS_Peak	Virt_Peak	Gbl_Pg_Cnt	Prc_Pg_Cnt	Total_Mem
2A00023D	BKM1_AG	AMQZLAA0	Agent	23152	16576	203776	3616	12960	16576
2A00023C	BKM1_CI	RUNMQCHI	Run Chan Init	8752	6208	180832	1840	4368	6208
2A00023B	BKM1_RM	AMQRRMFA	Repository Mgr	11152	8144	185360	2224	5920	8144
2A00023A	BKM1_CP	AMQZLLP0	Checkpoint	8752	6384	185952	1920	4464	6384
2A000239	BKM1_LG	AMQHASMXX	Logger	8752	6288	182016	2080	4208	6288
2A000238	BKM1_EC	AMQZXMA0	EC	20752	15232	203792	3536	11680	15216
2A000128	_FTA4:	MONMQ	MONMQ Utility	8400	8528	198736	2224	3584	5808

									52112

Identify the execution controller and agent process and connect them to channel one and two respectively.

```
MQT>connect pid=0x238 tid=1 lu=1 chl=1
MQT>connect pid=0x23D tid=1 lu=1 chl=2
```

Check connection details.

```
MQT>show channel full connected lu=1
```

```

Pid/Tid           : 2a000bc/1
Status           : *** Connected ***
Process name     : AMQZXMA0.EXE
Assigned LU      : 1
Channel no.      : 1
Mailbox channel  : 800c03f0
Current stack depth : 4
Circular logging  : Disabled
Next log entry   : 0
Realtime tracing : Disabled
Time stamping    : Disabled
Trace mask       : ffffffff
Step mode        : Off
No Wait          : On
Last QIO status  : 0
Mapped address   : 1242000-126bfff
=====

```

```

Pid/Tid           : 2a000c1/1
Status           : *** Connected ***
Process name     : AMQZLAA0.EXE
Assigned LU      : 1
Channel no.      : 2
Mailbox channel  : 800c03f0
Current stack depth : 4
Circular logging  : Disabled
Next log entry   : 0
Realtime tracing : Disabled
Time stamping    : Disabled
Trace mask       : ffffffff
Step mode        : Off
No Wait          : On
Last QIO status  : 0
Mapped address   : 1376000-139ffff
=====

```

Set defaults for chl, lu and tid to save entering these each time for subsequent commands.

```
MQT>default chl=1:2 lu=1 tid=1
```

Set channel colors so as to distinguish between different trace message. Note that the chl parameter is specified in these two commands because, had the default (1:2) been used then both channels would have been set to yellow and then cyan.

```
MQT>set chl=1 color=yellow
MQT>set chl=2 color=cyan
```

Now show channels.

```
MQT>show channels
```

Ch1	Pid/Tid	Mailbox	Stack	History	RTime	Time	Mask	Process Name
1	2a0000bc/1	800c03f0	4	0	0	0	fffffff	AMQZXMA.EXE
2	2a0000c1/1	800c0360	4	0	0	0	fffffff	AMQZLAA0.EXE

Now that both processes have been connected to channels, we can now examine their stacks.

```
MQT>show stacks
```

```
0001- 00:00:00.00 03 - 01 -->| ExecCtrlrMain
0002 - 00:00:00.00 03 - 02 ---->| zcpReceiveOnLink
0003 - 00:00:00.00 03 - 03 ---->| xcsWaitEventSem
0004 - 00:00:00.00 03 - 04 ---->| vms_evt

0001- 00:00:00.00 03 - 01 -->| zlaMain
0002 - 00:00:00.00 03 - 02 ---->| zcpReceiveOnLink
0003 - 00:00:00.00 03 - 03 ---->| xcsWaitEventSem
0004 - 00:00:00.00 03 - 04 ---->| vms_evt
```

By enabling timestamping for these two channels we are able to see whether either process has hung or not.

```
MQT>enable timestamp
MQT>show stack
```

```
0001- 00:00:00.00 03 - 01 -->| ExecCtrlrMain
0002 - 12:36:20.18 03 - 02 ---->| zcpReceiveOnLink
0003 - 12:36:20.81 03 - 03 ---->| xcsWaitEventSem
0004 - 12:36:20.83 03 - 04 ---->| vms_evt

0001- 00:00:00.00 03 - 01 -->| zlaMain
0002 - 12:36:20.18 03 - 02 ---->| zcpReceiveOnLink
0003 - 12:36:20.81 03 - 03 ---->| xcsWaitEventSem
0004 - 12:36:20.83 03 - 04 ---->| vms_evt
```

It can now be seen that there are some messages with a valid timestamp. This shows that both processes are active. In this case both processes are in an event loop with a 10 second timeout period. This timeout can be checked against the message timestamp by continuously performing a **show stack** command until there is a change in the timestamp data.

By enabling history we can now force each process to write their trace messages to the circular buffer.

```
MQT>enable history
MQT> show history
```

At this point we are writing all trace messages to the buffer. You can check this by showing the trace mask and the component/function table.

```
MQT>show mask
```

```
Trace Mask for Channel 1
Bit 00 - (fent) function entry
Bit 01 - (fout) function exit
Bit 02 - (ferr) function exit with error
Bit 03 - (fxx) missing function exit
Bit 04 - (dgn) diagnostic messages
Bit 05 - (shm) shared memory
Bit 06 - (spl) spinlocks
Bit 07 - (evt) events
Bit 08 - (mtx) mutexes
Bit 09 - (prc) process msgs
Bit 10 - (msc) miscellaneous
Bit 11 - (inf) informational
Bit 12 -
```

```
Trace Mask for Channel 2
Bit 00 - (fent) function entry
Bit 01 - (fout) function exit
Bit 02 - (ferr) function exit with error
Bit 03 - (fxx) missing function exit
Bit 04 - (dgn) diagnostic messages
Bit 05 - (shm) shared memory
Bit 06 - (spl) spinlocks
Bit 07 - (evt) events
Bit 08 - (mtx) mutexes
Bit 09 - (prc) process msgs
Bit 10 - (msc) miscellaneous
Bit 11 - (inf) informational
Bit 12 -
```

```
MQT> select
```

```
Ch1:1 - Cmp/fnc selection criteria
ALL component/functions
```

```
-----
Ch1:2 - Cmp/fnc selection criteria
ALL component/functions
-----
```

Trace Mask for Channel 1
Bit 00 - (fent) function entry
Bit 01 - (fout) function exit
Bit 02 - (ferr) function exit with error
Bit 03 - (fxx) missing function exit
Bit 04 - (dgn) diagnostic messages
Bit 05 - (shm) shared memory
Bit 06 - (spl) spinlocks
Bit 07 - (evt) events
Bit 08 - (mtx) mutexes
Bit 09 - (prc) process msgs
Bit 10 - (msc) miscellaneous
Bit 11 - (inf) informational
Bit 12 -

Trace Mask for Channel 2
Bit 00 - (fent) function entry
Bit 01 - (fout) function exit
Bit 02 - (ferr) function exit with error
Bit 03 - (fxx) missing function exit
Bit 04 - (dgn) diagnostic messages
Bit 05 - (shm) shared memory
Bit 06 - (spl) spinlocks
Bit 07 - (evt) events
Bit 08 - (mtx) mutexes
Bit 09 - (prc) process msgs
Bit 10 - (msc) miscellaneous
Bit 11 - (inf) informational
Bit 12 -

Let's now focus on a particular type of message. Say, for example, that we are interested only in shared memory diagnostic messages.

```
MQT>set mask=shm  
MQT>show mask
```

Both processes write only diagnostic memory type messages to the buffer. Let's delete the buffer, wait a few seconds and re-examine the contents of the buffer.

```
MQT>clear history  
  
(wait a few seconds)  
  
MQT> show history
```

*** Trace History Ch1:1 ***

```
0990 - 00:00:00.00 00 - 04 ..... | vms_mapgbl key : ffffffff - addr : 0/0
0991 - 00:00:00.00 00 - 04 ..... | vms_mapgbl Section MQS1_shm_fffffffe mapped at 1510000-1511fff
0992 - 00:00:00.00 00 - 04 ..... | vms_mapgbl key : ffffffff - addr : 0/0
0993 - 00:00:00.00 00 - 04 ..... | vms_mapgbl Section MQS1_shm_fffffffe mapped at 1510000-1511fff
0994 - 00:00:00.00 00 - 04 ..... | vms_mapgbl key : ffffffff - addr : 0/0
0995 - 00:00:00.00 00 - 04 ..... | vms_mapgbl Section MQS1_shm_fffffffe mapped at 1510000-1511fff
0996 - 00:00:00.00 00 - 04 ..... | vms_mapgbl key : ffffffff - addr : 0/0
0997 - 00:00:00.00 00 - 04 ..... | vms_mapgbl Section MQS1_shm_fffffffe mapped at 1510000-1511fff
0998 - 00:00:00.00 00 - 04 ..... | vms_mapgbl key : ffffffff - addr : 0/0
0999 - 00:00:00.00 00 - 04 ..... | vms_mapgbl Section MQS1_shm_fffffffe mapped at 1510000-1511fff
```

*** End of buffer ***

*** Trace History Ch1: ***

*** End of buffer ***

Now let's also display event type diagnostic messages in the trace output. We must wait for a few seconds after setting the mask.

```
MQT>set mask=evt | shm
```

(wait a few seconds)

```
MQT>show history
```


*** Trace History Ch1:1 ***

```
0977 - 00:00:00.00 00 - 04 ..... | vms_mapgbl Section MQS1_shm_ffffffff mapped at 1510000-1511fff
0978 - 00:00:00.00 00 - 04 ..... | vms_evt Event BKM3/@ipcc_e_1_2 TIMEOUT
0979 - 00:00:00.00 00 - 04 ..... | vms_evt rc = 1
0980 - 00:00:00.00 00 - 04 ..... | vms_mapgbl key : ffffffff - addr : 0/0
0981 - 00:00:00.00 00 - 04 ..... | vms_mapgbl Section MQS1_shm_ffffffff mapped at 1510000-1511fff
0982 - 00:00:00.00 00 - 04 ..... | vms_evt Reset on mailbox BKM3/@ipcc_e_1_2 : tout = -1
0983 - 00:00:00.00 00 - 05 ..... | vms_get_mbx_chan Getting mbx BKM3/@ipcc_e_1_2
0984 - 00:00:00.00 00 - 05 ..... | vms_get_mbx_chan Returning key 1a0
0985 - 00:00:00.00 00 - 04 ..... | vms_evt rc = 0
0986 - 00:00:00.00 00 - 04 ..... | vms_evt Wait on mailbox BKM3/@ipcc_e_1_2 : tout = 10000
0987 - 00:00:00.00 00 - 05 ..... | vms_get_mbx_chan Getting mbx BKM3/@ipcc_e_1_2
0988 - 00:00:00.00 00 - 05 ..... | vms_get_mbx_chan Returning key 1a0
0989 - 00:00:00.00 00 - 04 ..... | vms_evt Event BKM3/@ipcc_e_1_2 TIMEOUT
0990 - 00:00:00.00 00 - 04 ..... | vms_evt rc = 1
0991 - 00:00:00.00 00 - 04 ..... | vms_mapgbl key : ffffffff - addr : 0/0
0992 - 00:00:00.00 00 - 04 ..... | vms_mapgbl Section MQS1_shm_ffffffff mapped at 1510000-1511fff
0993 - 00:00:00.00 00 - 04 ..... | vms_evt Reset on mailbox BKM3/@ipcc_e_1_2 : tout = -1
0994 - 00:00:00.00 00 - 05 ..... | vms_get_mbx_chan Getting mbx BKM3/@ipcc_e_1_2
0995 - 00:00:00.00 00 - 05 ..... | vms_get_mbx_chan Returning key 1a0
0996 - 00:00:00.00 00 - 04 ..... | vms_evt rc = 0
0997 - 00:00:00.00 00 - 04 ..... | vms_evt Wait on mailbox BKM3/@ipcc_e_1_2 : tout = 10000
0998 - 00:00:00.00 00 - 05 ..... | vms_get_mbx_chan Getting mbx BKM3/@ipcc_e_1_2
0999 - 00:00:00.00 00 - 05 ..... | vms_get_mbx_chan Returning key 1a0
```

*** End of buffer ***

*** Trace History Ch1:2 ***

```
0982 - 00:00:00.00 01 - 04 ..... | vms_evt Event BKM3/@ipcc_e_1_7 TIMEOUT
0983 - 00:00:00.00 01 - 04 ..... | vms_evt rc = 1
0984 - 00:00:00.00 01 - 04 ..... | vms_evt Reset on mailbox BKM3/@ipcc_e_1_7 : tout = -1
0985 - 00:00:00.00 01 - 05 ..... | vms_get_mbx_chan Getting mbx BKM3/@ipcc_e_1_7
0986 - 00:00:00.00 01 - 05 ..... | vms_get_mbx_chan Returning key 1c0
0987 - 00:00:00.00 01 - 04 ..... | vms_evt rc = 0
0988 - 00:00:00.00 01 - 04 ..... | vms_evt Wait on mailbox BKM3/@ipcc_e_1_7 : tout = 10000
0989 - 00:00:00.00 01 - 05 ..... | vms_get_mbx_chan Getting mbx BKM3/@ipcc_e_1_7
0990 - 00:00:00.00 01 - 05 ..... | vms_get_mbx_chan Returning key 1c0
0991 - 00:00:00.00 01 - 04 ..... | vms_evt Event BKM3/@ipcc_e_1_7 TIMEOUT
0992 - 00:00:00.00 01 - 04 ..... | vms_evt rc = 1
0993 - 00:00:00.00 01 - 04 ..... | vms_evt Reset on mailbox BKM3/@ipcc_e_1_7 : tout = -1
0994 - 00:00:00.00 01 - 05 ..... | vms_get_mbx_chan Getting mbx BKM3/@ipcc_e_1_7
0995 - 00:00:00.00 01 - 05 ..... | vms_get_mbx_chan Returning key 1c0
0996 - 00:00:00.00 01 - 04 ..... | vms_evt rc = 0
0997 - 00:00:00.00 01 - 04 ..... | vms_evt Wait on mailbox BKM3/@ipcc_e_1_7 : tout = 10000
0998 - 00:00:00.00 01 - 05 ..... | vms_get_mbx_chan Getting mbx BKM3/@ipcc_e_1_7
0999 - 00:00:00.00 01 - 05 ..... | vms_get_mbx_chan Returning key 1c0
```

*** End of buffer ***

```
MQT>disable history
MQT>clear history
```

Now we focus on tracing a specific function. Using **SHOW COMPONENT** and **SHOW FUNCTION** we can identify the particular area we want to trace. In this example we are going to trace the common services function 'xcsRequestMutexSem'. The component is 0x17 and the function code is 0x1b. We can set this one of two ways:

```
MQT>select comp=0x17 func=0x1b
```

or

```
MQT>select fname="xcsRequestMutexSem"
```

If we now enable history, we find that no output appears in the buffer. This is because we need to reset the trace mask bits to all.

```
MQT>enable history  
MQT>show history
```

```
*** Trace History Ch1:1 ***
```

```
*** End of buffer ***
```

```
*** Trace History Ch:2 ***
```

```
*** End of buffer ***
```

```
MQT>set mask=0xffffffff  
MQT>show history
```

*** Trace History Ch1:1 ***

```
0973 - 00:00:00.00 01 - 02 --->| xcsRequestMutexSem
0974 - 00:00:00.00 01 - 03 ---->| x11SemReq
0975 - 00:00:00.00 01 - 04 ---->| vms_mtx
0976 - 00:00:00.00 01 - 04 .....| vms_mtx :- Locking BKM3_m_1_45 - timeout: -1
0977 - 00:00:00.00 01 - 05 ---->| vms_get_lock
0978 - 00:00:00.00 01 - 05 ----<| vms_get_lock
0979 - 00:00:00.00 01 - 04 ----<| vms_mtx
0980 - 00:00:00.00 01 - 03 ----<| x11SemReq
0981 - 00:00:00.00 01 - 02 ---<| xcsRequestMutexSem
0982 - 00:00:00.00 01 - 03 ---->| xcsRequestMutexSem
0983 - 00:00:00.00 01 - 04 ---->| x11SemReq
0984 - 00:00:00.00 01 - 05 ---->| vms_mtx
0985 - 00:00:00.00 01 - 05 .....| vms_mtx :- Locking BKM3_m_1_6 - timeout: -1
0986 - 00:00:00.00 01 - 06 ---->| vms_get_lock
0987 - 00:00:00.00 01 - 06 ----<| vms_get_lock
0988 - 00:00:00.00 01 - 05 ----<| vms_mtx
0989 - 00:00:00.00 01 - 04 ----<| x11SemReq
0990 - 00:00:00.00 01 - 03 ----<| xcsRequestMutexSem
0991 - 00:00:00.00 01 - 03 ---->| xcsRequestMutexSem
0992 - 00:00:00.00 01 - 04 ---->| x11SemReq
0993 - 00:00:00.00 01 - 05 ---->| vms_mtx
0994 - 00:00:00.00 01 - 05 .....| vms_mtx :- Locking BKM3/@ipcc_m_1_18 - timeout: -1
0995 - 00:00:00.00 01 - 06 ---->| vms_get_lock
0996 - 00:00:00.00 01 - 06 ----<| vms_get_lock
0997 - 00:00:00.00 01 - 05 ----<| vms_mtx
0998 - 00:00:00.00 01 - 04 ----<| x11SemReq
0999 - 00:00:00.00 01 - 03 ----<| xcsRequestMutexSem
```

*** End of buffer ***

We can now see that only the specified function and child functions are traced for both processes. Up to eight components and functions can be traced simultaneously using the **select** command. To enable trace in real time (that is, as it happens) we need to create a client process to display the messages for a specific LU. We do this by performing the **TRACE** command.

This launches a client process on the specified node and waits for incoming trace messages. Trace sessions on client windows can still be controlled using **MONMQ**.

```
MQT>trace start node="mihell"
```

Now enable the client process and display the messages as and when they arrive.

```
MQT>enable trace
```

WebSphere MQ threads can be added or removed from the trace output at will. Threads can remain connected but their trace data can be disabled so that tracing has no adverse effects on performance.

Tracing can be initiated the moment a process or thread starts. The **ONSTARTUP** command is used to do this and results in all new WebSphere MQ processes to be traced from startup.

To shutdown a trace session, all active channels should be disabled and client process ended. The **close** command does all this for you.

If you want to leave tracing running then use **quit** from MONMQ and resume tracing at a later date.

Note that trace mask bits and component/function selection are very different. Trace mask bits control the output of trace message types. For example trace entry and trace output are message types. If you disable these then whatever you set using the **select** command has no effect because component/function selection relies on these mask bits being set.

Appendix H. User exits

WebSphere MQ for HP OpenVMS supports both channel exit programs and data-conversion exit programs. For information about channel exits, see the *WebSphere MQ Intercommunication* book. For information about data-conversion exits, see the *WebSphere MQ Application Programming Guide* and the *WebSphere MQ Application Programming Reference* book.

This appendix provides information specific to the use of exit programs in WebSphere MQ for HP OpenVMS.

Channel and Workload Exits

The requirement to link a separate threaded version of an Exit is not applicable in WebSphere MQ for HP OpenVMS.

WebSphere MQ Cluster Workload Exits

When linking a workload exit on OpenVMS, the following should be specified in the linker options file:

```
sys$share:mqm/share  
sys$share:mqt1/share  
SYMBOL_VECTOR=(c1wlFunction=PROCEDURE,MQStart=PROCEDURE)
```

A system wide executive logical name is required to reference the exit image. For example if the exit name is SYS\$SHARE:AMQSWLM.EXE the following logical name should be defined:

```
$DEFINE/SYSTEM/EXEC AMQSWLM SYS$SHARE:AMQSWLM
```

The .EXE file extension must not be specified in the logical name definition.

For this logical name to be defined during system startup, define it in SYS\$MANAGER:MQS_SYSTARTUP.COM.

Appendix I. Trusted applications

If performance is an important consideration in your environment and your environment is stable, then user applications, channels, and listeners may be defined to be "trusted" that is, they use fastpath binding. (The time taken to process MQPUT and MQGET calls of nonpersistent messages can be reduced by up to 400% on OpenVMS systems.)

In a trusted application, the WebSphere MQ application and the local queue manager agent become the same process. The application connects directly to queue manager resources and effectively becomes an extension of the queue manager. This option can compromise the integrity of a queue manager as there is no protection from overwriting its storage.

Also, trusted applications may need to create certain resources like shared memory. These resources may need to be accessed by another queue manager process and, therefore, must be owned by the same UIC. The queue manager processes all run under the MQM account and thus trusted applications must also run under this account.

The issues detailed above should be considered before using trusted applications.

User applications

It is not necessary to run your application directly from the MQM account. Following a successful connection to a queue manager, WebSphere MQ automatically modifies the security profile of the active thread such that the thread assumes the identity of the MQM account. The natural identity of the thread is resumed following a call to disconnect from the queue manager.

It is important to note that while a trusted application is connected to a queue manager the application is effectively running under the MQM account. If it is necessary to change the identity of the thread to another UIC while connected to a queue manager, you must ensure that you change it back to MQM before making the next MQI call.

Setting up trusted applications

To run a trusted application on WebSphere MQ for OpenVMS you should specify the type of binding in the Options field of the MQCONN call to be MQCNO_FASTPATH_BINDING. (For standard binding use the MQCNO_STANDARD_BINDING option.) If no options are specified (MQCNO_NONE) the default is to use STANDARD_BINDING.

In addition, the logical name MQ_CONNECT_TYPE may be used to override the binding type specified on the MQCONN call. If the logical name is defined, it should have the value FASTPATH or STANDARD to select the type of binding required. However, FASTPATH binding is used only if the connect option is appropriately specified on the MQCONN call. This logical name enables you to execute an application with the STANDARD_BINDING if any problems occur with the FASTPATH_BINDING, without the need to rebuild the application.

In summary, to run a trusted application, either:

- Specify the MQCNO_FASTPATH_BINDING option on the MQCONN call and define the MQ_CONNECT_TYPE logical name as FASTPATH

or

- Specify the MQCNO_FASTPATH_BINDING option on the MQCONN call and leave the MQ_CONNECT_TYPE logical name undefined.

For further information on the use of trusted applications see the *WebSphere MQ Intercommunication*.

Running channels and listeners as trusted applications

Channel programs started using the **runmqsc start channel** command run under the MQM account. Channel receiver programs started by incoming TCP (or DECnet connect) requests run under the MQM account also.

The **runmqchl** and **runmqslr** commands create a detached process that runs under the MQM account. A combination of the MQ_CONNECT_TYPE logical name and MQIBindType in the channels stanza of a queue manager's *qm.ini* file define whether a channel or listener is to be run as trusted.

To set up a trusted channel or listener, either:

- Specify MQIBindType=FASTPATH in the *qm.ini* file and set the logical name to FASTPATH

or

- Specify MQIBindType=FASTPATH in the *qm.ini* file and leave the logical name undefined.

Fast, nonpersistent messages

The nonpersistent message speed (NPMSPEED) channel attribute can be used to specify the speed at which nonpersistent messages are to be sent. You can specify either normal or fast. The default is fast, which means that nonpersistent messages on a channel need not wait for syncpoint before being made available for retrieval. Such messages become available for retrieval far more quickly but may be lost if there is a transmission failure or if the channel stops while the messages are in transit. For further information on running channels and listeners as trusted applications and fast, nonpersistent messages see the *WebSphere MQ Intercommunication* book.

Appendix J. SSL CipherSpecs

The following table maps the SSL cipherspecs defined for WebSphere MQ on other platforms, and their equivalents for HP SSL

WebSphere MQ	OpenSSL
DES_SHA_EXPORT	EXP-DES-CBC-SHA
DES_SHA_EXPORT1024	EXP1024-DES-CBC-SHA
NULL_MD5	NULL-MD5
NULL_SHA	NULL-SHA
RC2_MD5_EXPORT	EXP-RC2-MD5
RC4_56_SHA_EXPORT1024	EXP1024-RC4-SHA
RC4_MD5_EXPORT	EXP-RC4-MD5
RC4_MD5_US	RC4-MD5
RC4_SHA_US	RC5-SHA
TLS_RSA_WITH_3DES_EDE_CBC_SHA	DES-CBC3-SHA
TLS_RSA_WITH_AES_128_CBC_SHA	AES128-SHA
TLS_RSA_WITH_AES_256_CBC_SHA	AES256-SHA
TLS_RSA_WITH_DES_CBC_SHA	DES-CBC-SHA
TRIPLE_DES_SHA_US	DES-CBC3-SHA

The following table lists the CipherSpecs supported on WebSphere MQ.

WebSphere MQ	OpenSSL
DES_SHA_EXPORT	ADH-AES256-SHA
DES_SHA_EXPORT1024	DHE-RSA-AES256-SHA
NULL_MD5	DHE-DSS-AES256-SHA
NULL_SHA	AES256-SHA
RC2_MD5_EXPORT	ADH-AES128-SHA
RC4_56_SHA_EXPORT1024	DHE-RSA-AES128-SHA
RC4_MD5_EXPORT	DHE-DSS-AES128-SHA
RC4_MD5_US	AES128-SHA
RC4_SHA_US	DHE-DSS-RC4-SHA
TLS_RSA_WITH_3DES_EDE_CBC_SHA	EXP1024-DHE-DSS-RC4-SHA
TLS_RSA_WITH_AES_128_CBC_SHA	EXP1024-RC4-SHA
TLS_RSA_WITH_AES_256_CBC_SHA	EXP1024-DHE-DSS-DES-CBC-SHA
TLS_RSA_WITH_DES_CBC_SHA	EXP1024-DES-CBC-SHA
TRIPLE_DES_SHA_US	EXP1024-RC2-CBC-MD5
	EXP1024-RC4-MD5
	EDH-RSA-DES-CBC-SHA
	EXP-EDH-RSA-DES-CBC-SHA
	EDH-DSS-DES-CBC3-SHA
	EDH-DSS-DES-CBC-SHA
	EXP-EDH-DSS-DES-CBC-SHA
	DES-CBC3-SHA
	DES-CBC-SHA
	EXP-DES-CBC-SHA
	IDEA-CBC-SHA
	EXP-RC2-CBC-MD5
	RC4-SHA
	RC4-MD5
	EXP-RC4-MD5
	ADH-DES-CBC3-SHA
	ADH-DES-CBC-SHA
	EXP-ADH-DES-CBC-SHA
	ADH-RC4-MD5
	EXP-ADH-RC4-MD5
	RC4-64-MD5
	DES-CBC3-MD5
	DES-CBC-MD5
	IDEA-CBC-MD5
	RC2-CBC-MD5
	EXP-RC2-CBC-MD5
	RC4-MD5
	EXP-RC4-MD5
	NULL-SHA
	NULL-MD5

Notices

This information was developed for products and services offered in the United States. IBM may not offer the products, services, or features discussed in this information in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this information. The furnishing of this information does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive Armonk, NY
10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

:IBM World Trade Asia Corporation Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the information. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this information at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM United Kingdom Laboratories,
Mail Point 151,
Hursley Park,
Winchester,
Hampshire,
England
SO21 2JN.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Programming License Agreement, or any equivalent agreement between us.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This product includes the compression library zlib, which is available from:
<http://www.gzip.org/zlib>.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at www.ibm.com/legal/copytrade.shtml.

Index

A

- accidental deletion of default queue manager 207
- ACTION keyword, rules table 129
- administration
 - authorizations 105
 - control commands 21
 - for database managers 118
 - introduction to 15
 - local, definition of 15
 - MQAI, using 56
 - MQSC commands 16, 30
 - object name transformation 18
 - PCF commands 55
 - queue manager name transformation 17
 - remote administration, definition of 15
 - remote objects 57
 - understanding WebSphere MQ file names 17
 - using control commands 16
 - using PCF commands 16
- AdoptNewMCA attribute 84
- AIX operating system
 - MQAI support 56
 - start client trigger monitor (runmqtm) command 260
- alias queues
 - authorizations to 98
 - DEFINE QALIAS command 43
 - defining alias queues 43
 - remote queues as queue manager aliases 68
 - reply-to queues 68
 - working with alias queues 43
- aliases
 - queue manager aliases 68
 - working with alias queues 43
- AllQueueManagers stanza, mqs.ini 73
- alternate user authority 98
- amqsdlq, the sample DLQ handler 126
- analyse trace command (MONMQ) 319
- API exits 12
- ApiExitCommon stanza, mqs.ini 78
- ApiExitLocal stanza, qm.ini 86
- ApiExitTemplate stanza, mqs.ini 78
- application
 - trusted 337
- application programs
 - design considerations 188
 - message length, effects on performance 188
 - MQI local administration, support for 29
 - persistent messages, effect on performance 188
 - programming errors, examples of 181
 - receiving messages 2
 - retrieving messages from queues 3

- application programs (*continued*)
 - searching for messages, effect on performance 188
 - sending messages 2
 - time-independent applications 1
- application queues
 - defining application queues for triggering 51
 - description of 6
- APPLIDAT keyword, rules table 128
- APPLNAME keyword, rules table 128
- APPLTYPE keyword, rules table 128
- attributes
 - changing local queue attributes 40
 - LIKE attribute, DEFINE command 40
 - queue manager 37, 38
 - queues 5
 - WebSphere MQ and PCF commands, a comparison 56
- authentication information objects
 - description of 9
- authority
 - alternate user 98
 - commands 97
 - context 99
 - grant or revoke command 261
 - installable services 97
- authorization
 - administration 105
 - dspmqaout command 97
 - lists 95
 - MQI 102
 - rights identifiers 94
 - setmqaut command 97
- authorization service 12
- automatic definition of channels 62

B

- backing up queue manager data 169
- binary
 - close trace binary file 315
 - open a trace binary file 314
- bindings
 - for trusted applications 337
- browsing queues 41
- built-in formats, data conversion 69

C

- calculating the size of logs 162
- case sensitivity 18
 - control commands 18
 - MQSC commands 19
- ccsid.tbl, data conversion 69
- changing
 - CCSID 70
 - local queue attributes 40
 - queue manager attributes 38
 - the default queue manager 25

- channel
 - Channels stanza, qm.ini 83
 - command security requirements 100
 - commands 100
 - connect target thread to 312
 - disconnect target thread from 313
 - escape command authorizations 105
 - fastpath 84
 - security 100
 - show channel details (MONMQ) 303
 - show history of messages 305
 - show mask (MONMQ) 304
 - trusted 338
- channels
 - administering a remote queue manager from a local one 59
 - auto-definition of 62
 - defining channels for remote administration 61
 - description of 9, 57
 - exits 12
 - preparing channels for remote administration 60
 - remote queuing 57
 - starting 62
 - using the run channel (runmqchl) command 254
 - using the run initiator (runmqchi) command 253
- Channels stanza, qm.ini 83
- character code sets, updating 69
- circular logging 158
- clearing a local queue 41
- client connection channels
 - description of 9
- ClientExitPath stanza 75
- clients and servers
 - definitions 11
 - problem determination 198
 - start client trigger monitor (runmqtm) command 260
- close binary command (MONMQ) 315
- close LU command (MONMQ) 303
- close text command (MONMQ) 315
- cluster
 - OpenVMS
 - difference from queue manager cluster 137
 - failover sets 138
 - installing WebSphere MQ 137
 - queue manager workload exit 335
 - cluster alias service 150
 - cluster workload exit 335
- clusters
 - cluster transmission queues 7
 - description of 8, 58
 - ExitProperties stanza attributes 75
 - remote queuing 57
- coded character sets, specifying 69
- command files 33

- command procedures 142
 - examples 152
 - modifying 152
- command queues
 - command server status 63
 - description of 7
 - mandatory for remote administration 60
- command server
 - display command server (dspmqcsv)
 - command 220
 - displaying status 63
 - end command server (endmqcsv)
 - command 233
 - remote administration 57, 63
 - starting a command server 63
 - starting the command server (strmqcsv) command 268
 - stopping a command server 63
- command set
 - comparison 281
- command sets
 - control commands 21
 - MQSC commands 30
 - PCF commands 55
- commands
 - comparison of sets 281
 - control commands 21
 - create queue manager (crtmqm)
 - command 205
 - data conversion (crtmqcvx)
 - command 204
 - delete queue manager (dlmqm)
 - command 209
 - display authority (dspmqaut)
 - command 216
 - display command server (dspmqcsv)
 - command 220
 - display version information (dspmqver) 232
 - display WebSphere MQ files (dspmqfls) command 221
 - display WebSphere MQ formatted trace (dspmqtrc) 230
 - display WebSphere MQ queue managers (dspmq) command 215
 - display WebSphere MQ transactions (dspmqtrn) command 231
 - dump authority (dmpmqaut)
 - command 210
 - dump log (dmpmqlog)
 - command 214
 - end command server (endmqcsv)
 - command 233
 - end listener (endmqslr)
 - command 234
 - end queue manager (endmqm)
 - command 235
 - end WebSphere MQ trace (endmqtrc) 237
 - enroll production license (setmqprd) 268
 - grant or revoke authority (setmqaut) 261
 - help with syntax 201
 - issuing MQSC commands using an ASCII file 30
- commands (*continued*)
 - MQSC command files
 - input 33
 - output reports 34
 - PCF commands 55
 - receive file on client (mqftrcvc) 241
 - receive file on server (mqftrcv) 239
 - record media image (rcdmqimg)
 - command 248
 - recreate object (rcrmqobj)
 - command 250
 - resolve WebSphere MQ transactions (rsvmqtrn) command 252
 - run channel (runmqchl)
 - command 254
 - run channel initiator (runmqchi) 253
 - run dead-letter queue handler 255
 - run DLQ handler (runmqdlq)
 - command 125
 - run File Transfer Application (mqftapp) 238
 - run listener (runmqslr)
 - command 256
 - run MQSC commands (runmqsc) 258
 - runmqsc command, to issue MQSC commands 30
 - security commands
 - dspmqaut 97
 - setmqaut 95
 - send file from client (mqftsndc) 246
 - send file from server (mqftsnd) 244
 - set/reset authority (setmqaut) 96
 - start client trigger monitor (runmqmtc) command 260
 - start command server (strmqcsv) 268
 - start queue manager (strmqm) 269
 - start trigger monitor (runmqtrm) 261
 - WebSphere MQ (MQSC)
 - verifying 35
 - WebSphere MQ display route application (dspmqrte) 222
- component
 - show functions (MONMQ) 311
- configuration files
 - AllQueueManagers stanza, mqs.ini 73
 - ApiExitCommon, mqs.ini 78
 - ApiExitLocal, qm.ini 86
 - ApiExitTemplate, mqs.ini 78
 - backing up of 26
 - ClientExitPath stanza 75
 - DefaultQueueManager stanza 75
 - editing 71
 - ExitProperties stanza 75
 - failover.template 293
 - OpenVMS cluster failover set 141
 - queue manager (qm.ini)
 - Channels stanza 83
 - contents 73, 88
 - disabling the object authority manager 95
 - ExitPath stanza 86
 - Log stanza 80
 - LU62 and TCP stanzas 85
 - Service Component stanza 80
 - Service stanza 79
 - XARsourceManager stanza 82
- configuration files (*continued*)
 - QueueManager stanza, mqs.ini 78
 - WebSphere MQ (mqs.ini)
 - contents 72
 - LogDefaults stanza 76
 - path 35
- configuring
 - Channels stanza, qm.ini 83
 - database managers 113
 - databases, qm.ini 82
 - editing 71
 - example qm.ini file 88
 - Exitpath stanza, qm.ini 86
 - failover.ini 141
 - implementing changes 72
 - Log stanza, qm.ini 80
 - LogDefaults stanza, mqs.ini 76
 - logs 80
 - LU62 stanza, qm.ini 85
 - mqs.ini, description of 72
 - OpenVMS cluster failover set 141
 - Oracle 115
 - priorities 72
 - queue manager configuration file, qm.ini 73
 - Service stanza, qm.ini 79
 - ServiceComponent stanza, qm.ini 80
 - TCP stanza, qm.ini 85
 - XAResourceManager stanza, qm.ini 82
- connect command (MONMQ) 312
- context authority 99
- control commands
 - case sensitivity 18
 - changing the default queue manager 25
 - creating a default queue manager 25
 - creating a queue manager 22
 - crtmqm, creating a default queue manager 25
 - deleting a queue manager, dlmqm 28
 - dlmqm, deleting a queue manager 28
 - immediate shutdown 27
 - preemptive shutdown 27
 - quiesced shutdown 26
 - runmqsc, using interactively 31
 - starting a queue manager 26
 - strmqm, starting a queue manager, 26
 - using 21
- controlled shutdown of a queue manager 27
- CorrelId, performance considerations 188
- creating
 - a default queue manager 25
 - a dynamic (temporary) queue 3
 - a model queue 3
 - a predefined (permanent) queue 3
 - a process definition 53
 - a queue manager 22, 205
 - a transmission queue 67
- crtmqcvx (data conversion) command
 - examples 204
 - format 204

crtmqcvx (data conversion) command
 (*continued*)
 parameters 204
 purpose 204
 return codes 204
 crtmqm (create queue manager)
 command
 examples 209
 format 205
 parameters 206
 purpose 205
 related commands 209
 return codes 208
 CURDEPTH, current queue depth 40
 current queue depth, CURDEPTH 40

D

data conversion
 built-in formats 69
 ccsid.tbl, uses for 69
 ConvEBCDICNewline attribute,
 AllQueueManagers stanza 74
 converting user-defined message
 formats 70
 data conversion (crtmqcvx)
 command 204
 default data conversion 70
 EBCDIC NL character conversion to
 ASCII 74
 introduction 69
 updating coded character sets 69
 database managers
 changing the configuration
 information 122
 configuring 113
 connections to 113
 coordination 112
 database manager instances,
 removing 122
 defining database managers in
 qm.ini 113
 dspmqtrn command, checking
 outstanding UOWs 119
 in-doubt units of work 119
 restrictions, database coordination
 support 112
 rsvmqtrn command, explicit
 resynchronization of UOWs 120
 switch load files, creating 113
 DCE Generic Security Service (GSS)
 name service, installable service 12
 dead-letter header, MQDLH 125
 dead-letter queue handler
 ACTION keyword, rules table 129
 action keywords, rules table 129
 APPLIDAT keyword, rules table 128
 APPLNAME keyword, rules
 table 128
 APPLTYPE keyword, rules table 128
 control data 126
 DESTQ keyword, rules table 128
 DESTQM keyword, rules table 128
 example of a rules table 133
 FEEDBACK keyword, rules table 128
 FORMAT keyword, rules table 128
 FWDQ keyword, rules table 129

dead-letter queue handler (*continued*)
 FWDQM keyword, rules table 130
 HEADER keyword, rules table 130
 INPUTQ, rules table 126
 INPUTQM keyword, rules table 127
 invoking the DLQ handler 125
 MSGTYPE keyword, rules table 128
 pattern-matching keywords, rules
 table 128
 patterns and actions (rules) 127
 PERSIST keyword, rules table 129
 processing all DLQ messages 133
 processing rules, rules table 132
 PUTAUT keyword, rules table 130
 REASON keyword, rules table 129
 REPLYQ keyword, rules table 129
 REPLYQM keyword, rules table 129
 RETRY keyword, rules table 130
 RETRYINT, rules table 127
 rule table conventions 130
 rules table, description of 126
 sample, amqsdllq 126
 syntax rules, rules table 131
 USERID keyword, rules table 129
 WAIT keyword, rules table 127
 dead-letter queues
 defining a dead-letter queue 39
 description of 7
 DLQ handler 255
 MQDLH, dead-letter header 125
 specifying 23
 debugging
 command syntax errors 182
 common command errors 182
 common programming errors 181
 further checks 183
 preliminary checks 179
 default
 rights identifier for authority 94
 system objects 273
 default data conversion 70
 default transmission queues 68
 default variable command
 (MONMQ) 302
 DefaultQueueManager stanza 75
 defaults
 changing the default queue
 manager 25
 creating a default queue manager 25
 objects 10
 queue manager 23
 reverting to the original default queue
 manager 25
 transmission queue 24
 defining
 a model queue 45
 an alias queue 43
 an initiation queue 52
 WebSphere MQ queues 5
 delete history command (MONMQ) 316
 deleting
 a local queue 41
 a queue manager 28
 a queue manager using the dltnmqm
 command 209
 queue managers, WebSphere MQ for
 UNIX systems 287

deleting (*continued*)
 Windows queue managers 286
 Windows queue managers, automatic
 startup list 287
 deselect index command (MONMQ) 314
 DESTQ keyword, rules table 128
 DESTQM keyword, rules table 128
 determining current queue depth 40
 Digital TCP/IP Services for
 OpenVMS 143
 directories
 queue manager 98
 directory structure 275
 disable history command
 (MONMQ) 316
 disable timestamp command
 (MONMQ) 315
 disable tracing command
 (MONMQ) 316
 disabling the object authority
 manager 95
 disconnect command (MONMQ) 313
 display
 active WebSphere MQ processes 305
 current authorizations (dmpmqaut)
 command 210
 current authorizations (dspmqaut)
 command 216
 default object attributes 39
 file system name (dspmqfls)
 command 221
 hexidecimal ids for components 310
 memory table 309
 process definitions 53
 queue manager attributes 37
 queue managers (dspmq)
 command 215
 status of command server 63
 status of command server (dspmqcsv)
 command 220
 target threads stack 304
 WebSphere MQ formatted trace
 output command 230
 WebSphere MQ transactions
 (dspmqtrn) command 231
 display version information, dspmqver
 command 232
 distributed queuing, incorrect
 output 185
 dltnmqm (delete queue manager)
 command
 examples 210
 format 209
 parameters 210
 purpose 209
 related commands 210
 return codes 210
 dltnmqm control command 28
 dmpmqlog (dump log) command
 format 214
 parameters 214
 purpose 214
 dspmq (display WebSphere MQ queue
 managers) command
 format 215
 parameters 215
 purpose 215

- dspmqr (display WebSphere MQ queue managers) command *(continued)*
 - Queue Manager States 216
 - return codes 216
 - dspmqaout (display authority) command
 - dspmqaout command 219
 - examples 212, 219
 - format 217
 - parameters 217
 - purpose 210, 216
 - related commands 220
 - results 218
 - return codes 219
 - dspmqaout command
 - using 95, 97
 - dspmqrscsv (display command server)
 - command
 - examples 221
 - format 220
 - parameters 220
 - purpose 220
 - related commands 221
 - return codes 220
 - dspmqrfls (display WebSphere MQ files)
 - command
 - examples 222
 - format 221
 - parameters 221
 - purpose 221
 - return codes 222
 - dspmqrte
 - format 223
 - parameters 223
 - dspmqrtrc command 230
 - examples 231
 - parameters 230
 - related commands 231
 - dspmqrtrn (display WebSphere MQ transactions) command
 - format 231
 - parameters 231
 - purpose 231
 - related commands 232
 - return codes 232
 - dspmqrver
 - examples 233
 - format 232
 - parameters 232
 - dump
 - dumping log records (dmpmqlog command) 175
 - dumping the contents of a recovery log 175
 - formatted system log (dmpmqlog) command 214
 - dynamic definition of channels 62
 - dynamic queues
 - authorizations to 98
 - description of 3
- E**
- EBCDIC NL character conversion to ASCII 74
 - enable history command (MONMQ) 316
 - enable timestamp command (MONMQ) 315
 - enable tracing command (MONMQ) 315
 - enabling
 - security 95
 - End command procedure 142
 - end WebSphere MQ trace 237
 - EndCommand procedure
 - template 295
 - ending
 - interactive MQSC commands 32
 - endmqscsv (end command server)
 - command
 - examples 234
 - format 234
 - parameters 234
 - purpose 233
 - related commands 234
 - return codes 234
 - endmqslr (end listener) command
 - format 235
 - parameters 235
 - purpose 234
 - return codes 235
 - endmqm (end queue manager) command
 - examples 237
 - format 236
 - parameters 236
 - purpose 235
 - related commands 237
 - return codes 237
 - endmqtrc command 237
 - examples 238
 - parameters 237
 - related commands 238
 - return codes 238
 - enroll production license, setmqprd command 268
 - environment variables
 - MQSPREFIX 74
 - ORACLE_HOME, Oracle 115
 - ORACLE_SID, Oracle 115
 - error log
 - error occurring before established 190
 - example 190
 - error logs
 - description of 189
 - log files 189
 - error messages, MQSC commands 32
 - escape PCF command 17
 - escape PCFs 56
 - event queues
 - description of 7
 - events
 - show events (MONMQ) 308
 - examples
 - creating a transmission queue 67
 - crtmqcvx command 204
 - crtmqm command 209
 - dltmqm command 210
 - dmpmqaut command 212
 - dspmqaout command 219
 - dspmqrscsv command 221
 - dspmqrfls command 222
 - dspmqrte command 229
 - dspmqrtrc command 231
 - dspmqrver command 233
 - endmqscsv command 234
 - examples *(continued)*
 - endmqm command 237
 - endmqtrc command 238
 - error log 190
 - mqftrcv command 241
 - mqftrcvc command 244
 - mqftsnd command 246
 - mqftsndc command 248
 - mqts.ini file, WebSphere MQ for HP OpenVMS 87
 - programming errors 181
 - qm.ini file 88
 - rcdmqimg command 250
 - rcrmqobj command 252
 - runmqslr command 257
 - runmqsc command 259
 - runmqtmc command 260
 - setmqaut command 266
 - strmqscsv command 269
 - strmqm command 271
 - exit
 - cluster workload 335
 - exit command (MONMQ) 321
 - Exitpath stanza, qm.ini 86
 - ExitProperties stanza 75
 - extending queue manager facilities 11
- F**
- failback
 - description 139
 - failover
 - description 139
 - failover command 139, 142
 - failover monitor
 - description 139
 - halting 147
 - starting 144
 - watcher 139
 - failover set
 - changing state 148
 - command procedures 142
 - configuration file 139
 - configuring 140
 - displaying state 145
 - ending queue manager within 144
 - example configuration 150
 - moving queue manager within 145
 - starting queue manager within 144
 - steps before configuring 140
 - troubleshooting 149
 - using Intra Cluster Communication (ICC) 148
 - using MultiNet for OpenVMS 150
 - failover set templates 293
 - failover sets
 - administration 143
 - description 138
 - failover.ini configuration file 150
 - editing 141
 - failover.template 150
 - fastpath binding 337
 - fastpath channel 84
 - FEEDBACK keyword, rules table 128
 - feedback, MQSC commands 32
 - FFST
 - FFST (MONMQ) 321

- FFST command (MONMQ) 321
- FFST, examining 195
- file names 17
- file sizes, for logs 162
- files
 - log control file 158
 - log files, in problem determination 189
 - logs 158
 - names 17
 - queue manager configuration 73
 - sizes, for logs 162
 - understanding names 17
 - WebSphere MQ configuration 72
- FORMAT keyword, rules table 128
- functions
 - display within component 311
- FWDQ keyword, rules table 129
- FWDQM keyword, rules table 130

G

- global sections 306
- global units of work
 - adding XAResourceManager stanza to qm.ini, Oracle 116
 - definition of 14, 111
- guidelines for creating queue managers 22

H

- HEADER keyword, rules table 130
- help with command syntax 201
- hexadecimal ids
 - display for components 310
- history
 - delete history (MONMQ) 316
 - disable history (MONMQ) 316
 - enable history (MONMQ) 316
 - of messages in channel 305
 - reset for channel (MONMQ) 317
 - set history (MONMQ) 316
- HP-UX
 - MQAI support for 56

I

- indirect mode, runmqsc command 64
- indoubt transactions
 - database managers 119
 - display WebSphere MQ transactions (dspmqtrn) command 231
 - using the resolve WebSphere MQ (rsvmqtrn) command 252
- initiation queues
 - defining 52
 - description of 6
- input, standard 31
- installable component
 - authority manager (OAM) 92
- installable services
 - authorization service 12
 - definition of 12
 - disabling object authority manager disabling 95

- installable services (*continued*)
 - installable services, list of 12
 - name service 12
 - object authority manager 92
 - service component 12
- Installing multiple queue managers 24
- Intra Cluster Communication (ICC) 148
- issuing
 - MQSC commands remotely 64
 - MQSC commands using an ASCII file 30
 - MQSC commands using runmqsc command 30

L

- LIKE attribute, DEFINE command 40
- linear logging 159
- listener
 - end listener (endmqslsr)
 - command 234
 - starting 62
 - trusted 338
 - using the run listener (runmqslsr)
 - command 256
- listener objects
 - description of 10
- ListenerBacklog attribute 86
- listeners
 - defining listeners for remote administration 61
- local administration
 - creating a queue manager 22
 - definition of 15
 - issuing MQSC commands using an ASCII file 30
 - runmqsc command, to issue MQSC commands 30
 - support for application programs 29
- local queues 38
 - changing queue attributes, commands to use 40
 - clearing 41
 - copying a local queue definition 40
 - defining 38
 - defining application queues for triggering 51
 - deleting 41
 - description of 8
 - specific queues used by WebSphere MQ 6
 - working with local queues 38
- local unit of work
 - definition of 13, 111
- log
 - configuring 80
 - error, example of 190
 - file
 - reuse 160
 - Log stanza, qm.ini 80
 - Log stanza, qm.ini 80
 - LogDefaults stanza, mqs.ini 76
 - logging
 - calculating the size of logs 162
 - checkpoints 159, 160
 - circular 158
 - contents of logs 157

- logging (*continued*)
 - linear logging 159
 - locations for log files 166
 - media recovery 168
 - parameters 24
 - types of 158
 - what happens when a disk fills up? 164
- logical name, disabling security 95
- logical unit
 - close (MONMQ) 303
 - display using show segment (MONMQ) 303
- logs
 - calculating the size of logs 162
 - checkpoints 159
 - dumping log records (dmpmqlog command) 175
 - dumping the contents of 175
 - error logs 189
 - format of a log 158
 - log control file 158
 - log files, in problem determination 189
 - logging parameters 24
 - managing 164, 165
 - media recovery, linear logs 167
 - oldest required for recovery and restart 249
 - output from the dmpmqlog
 - command 175
 - overheads 162
 - parameters 24
 - persistent messages, effect upon log sizes 163
 - primary log files 158
 - protecting 169
 - recreating objects (rcrmqobj)
 - command 250
 - secondary log files 158
 - types of logs 157
 - using logs for recovery 166
 - what happens when a disk fills up? 164
- LU62 stanza, qm.ini 85

M

- macros
 - MONMQ 323
- managing access 93
- managing objects for triggering 51
- managing shared memory 322
- manual removal of a queue manager 285
- manually stopping a queue manager 285
- mask
 - set mask (MONMQ) 317
- maximum line length for MQSC commands 34
- MCA (message channel agent) 125
- media images
 - automatic media recovery failure, scenario 175
 - description of 167
 - oldest log required for recovery 249

- media images (*continued*)
 - record media image (rcdmqimg)
 - command 248
 - recording media images 167
 - recovering damaged objects during start up 168
 - recovering media images 168
 - memory table 309
 - message
 - nonpersistent message speed 338
 - message channel agent (MCA) 125
 - AdoptNewMCA attribute 84
 - channel in RETRY state 86
 - message length, decreasing 40
 - message queuing 1
 - message-driven processing 1
 - messages
 - application data 2
 - containing unexpected information 184
 - converting user-defined message formats 70
 - definition of 2
 - message descriptor 2
 - message length, effects on performance 188
 - message lengths 2
 - message-driven processing 1
 - not appearing on queues 183
 - operator messages 193
 - persistent messages, effect on performance 188
 - persistent messages, when determining log sizes 163
 - queuing 1
 - retrieval algorithms 3
 - retrieving messages from queues 3
 - sending and receiving 2
 - undelivered 194
 - variable length 189
 - model queues
 - creating a model queue 3
 - DEFINE QMODEL command 45
 - defining 45
 - working with 44
 - monitoring
 - start client trigger monitor (runmqtm) command 260
 - starting a trigger monitor (runmqtrm command) 261
 - monmq utility
 - commands
 - analyse trace 319
 - close binary 315
 - close lu 303
 - close text 315
 - connect 312
 - default variable 302
 - delete history 316
 - deselect index 314
 - disable history 316
 - disable timestamp 315
 - disable trace 316
 - disconnect 313
 - enable history 316
 - enable timestamp 315
 - enable trace 315
 - monmq utility (*continued*)
 - commands (*continued*)
 - exit 321
 - FFST 321
 - onstartup start 312
 - onstartup stop 312
 - open 302
 - open binary 314
 - open text 315
 - select 313
 - set color 318
 - set depth 316
 - set free 317
 - set mask 317
 - set output 319
 - show channels 303
 - show components 310
 - show events 308
 - show functions 311
 - show globals 306
 - show history 305
 - show mask 304
 - show memory 309
 - show mutex 307
 - show process 305
 - show segment 303
 - show stack 304
 - trace start 300, 313
 - trace stop 313
 - managing shared memory with 322
 - overview 299
 - tracing WebSphere MQ processes
 - sample trace session 324
 - scripts and macros in MONMQ 323
 - variables within MONMQ 300
 - MQAI (WebSphere MQ administrative interface)
 - description of 56
 - MQDLH, dead-letter header 125
 - mqftapp
 - format 238
 - related commands 238
 - mqftrcv
 - examples 241
 - format 239
 - parameters 239
 - related commands 241
 - return codes 239
 - mqftrvc
 - examples 244
 - format 241
 - parameters 241
 - related commands 244
 - return codes 241
 - mqftsnd
 - examples 246
 - format 244
 - parameters 244
 - related commands 246
 - return codes 244
 - mqftsndc
 - examples 248
 - format 246
 - parameters 246
 - related commands 248
 - return codes 246
 - MQI
 - authorizations 101, 102
 - MQI (message-queuing interface)
 - definition of 1
 - local administration support 29
 - queue manager calls 8
 - receiving messages 2
 - sending messages 2
 - MQM
 - rights identifier 91
 - user ID 98
 - MQOPEN authorizations 102
 - MQPUT and MQPUT1, performance considerations 189
 - MQPUT authorizations 102
 - mqs.ini
 - definition of 71
 - editing 71
 - LogDefaults stanza 76
 - path to 35
 - priorities 72
 - mqs.ini configuration file
 - AllQueueManagers stanza 73
 - ApiExitCommon stanza 78
 - ApiExitTemplate 78
 - QueueManager stanza 78
 - MQSC
 - case sensitivity 19
 - command files
 - input 33
 - output reports 34
 - running 34
 - maximum line length 34
 - problems
 - local 35
 - sample files 289
 - security requirements on channels 100
 - verifying commands 35
 - MQSC commands
 - case sensitivity 19
 - maximum line length 34
 - MQSNOAUT logical name 95
 - MQSPREFIX, environment variable 74
 - MQZAO constants and authority 102
 - MsgId, performance considerations when using 188
 - MSGTYPE keyword, rules table 128
 - MultiNet for OpenVMS 150
 - configuring template files 143
 - multiple queue managers, installing 24
 - mutex table 307
 - MVS/ESA queue manager 64
- ## N
- name service 12
 - name transformations 17
 - namelists
 - description of 9
 - naming conventions
 - national language support 199
 - object names 4
 - queue manager name transformation 17
 - national language support
 - data conversion 69

national language support (*continued*)
 EBCDIC NL character conversion to
 ASCII 74
 naming conventions for 199
 operator messages 193
 NL character, EBCDIC conversion to
 ASCII 74
 nobody, default rights identifier 94
 nonpersistent message 338

O

OAM 92
 OAM (Object Authority Manager)
 authorization service, installable
 service 12
 overview 13
 using the grant or revoke authority
 (setmqaut) command 261
 object authority manager 92
 default rights identifier 94
 disabling 95
 dspmqaut command 97
 how it works 93
 principals 93
 sensitive operations 97
 setmqaut command 95, 96
 object name transformation 18
 objects
 access to 91
 administration of 15
 attributes of 4, 17
 automation of administration
 tasks 16
 default object attributes,
 displaying 39
 description of 8, 9, 58
 display file system name (dspmqfls)
 command 221
 local queues 8
 managing objects for triggering 51
 media images 167
 multiple queues 8
 name transformation 18
 naming conventions 4, 199
 object name transformation 18
 process definitions 8
 queue manager objects used by MQI
 calls 8
 queue managers 8
 recovering damaged objects during
 start up 168
 recovering from media images 168
 recreate (rcrmqobj) command 250
 remote administration 57
 remote queue objects 68
 remote queues 8
 system default 273
 system default objects 10
 types of 3
 using MQSC commands to
 administer 16
 onstartup start command
 (MONMQ) 312
 onstartup stop command
 (MONMQ) 312
 open binary command (MONMQ) 314

open command (MONMQ) 302
 open text command (MONMQ) 315
 OpenVMS cluster failover set 140
 OpenVMS clusters 137
 OpenVMS logged-in user ID 98
 OpenVMS rights identifier
 default, nobody 94
 MQM 91
 operating system logical name, disabling
 security 95
 operator
 commands, no response from 186
 messages 193
 oracle
 configuration parameters,
 changing 118
 configuring 115
 environment variable settings,
 checking 115
 Oracle XA support, enabling 115
 ORACLE_HOME, environment
 variable 115
 ORACLE_SID, environment
 variable 115
 switch load file, creating 115
 XAResourcemanager stanza, adding to
 qm.ini 116
 output
 redirect 319
 output, standard 31
 overheads, for logs 162

P

PCF (programmable command format)
 administration tasks 16
 attributes in MQSC commands and
 PCF 56
 automating administrative tasks using
 PCF 55
 escape PCF 17
 escape PCFs 56
 MQAI, using to simplify use of 56
 object attribute names 4, 17
 performance
 advantages of using MQPUT1 189
 application design, impact on 188
 CorrelId, effect on 188
 message length, effects on 188
 message persistence, effect on 188
 MsgId, effect on 188
 syncpoints, effects on 189
 performance considerations
 when using trace 194
 permanent (predefined) queues 3
 PERSIST keyword, rules table 129
 persistent messages, effect on
 performance 188
 predefined (permanent) queues 3
 preemptive shutdown of a queue
 manager 27
 primary group authorizations 93
 primary rights identifier, for
 authority 93
 principals
 holding more than one rights
 identifier 93
 principals (*continued*)
 managing access to 93
 problem determination
 application design
 considerations 188
 applications or systems running
 slowly 187
 clients 198
 command errors 182
 common programming errors 181
 configuration files 194
 has the application run successfully
 before? 180
 incorrect output, definition of 183
 incorrect output, distributed
 queuing 185
 intermittent problems 182
 introduction 179
 log files 189
 no response from operator
 commands 186
 preliminary checks 179
 problems affecting parts of a
 network 182
 problems caused by service
 updates 182
 problems that occur at specific times
 in the day 182
 problems with shutdown 27
 queue failures, problems caused
 by 187
 remote queues, problems
 affecting 187
 reproducing the problem 180
 return codes 180, 181
 searching for messages, performance
 effects 188
 things to check first 179
 trace 194
 undelivered messages 194
 WebSphere MQ error messages 180
 what is different since the last
 successful run? 180
 problems
 running MQSC commands 35
 using MQSC locally 35
 process
 point to start trace 312
 process definitions
 creating 53
 description of 8
 displaying 53
 process rights authorizations 93
 processing, message-driven 1
 programmable command format (PCF)
 authorizations 101
 security requirements 100
 programming errors, examples of 181
 further checks 183, 188
 secondary checks 183, 188
 programs, samples supplied 289
 protected resources 94
 PUTAUT keyword, rules table 130

Q

- qm.ini configuration file
 - ApiExitLocal stanza 86
 - Channels stanza 83
 - definition of 73
 - editing 71
 - Exitpath stanza 86
 - Log stanza 80
 - LU62 stanza 85
 - priorities 72
 - Service stanza 79
 - ServiceComponent stanza 80
 - TCP stanza 85
 - XAResourceManager stanza 82
- queue depth, current 40
- queue manager
 - authorizations 98
 - command server 57
 - directories 98
 - ending within failover set 144
 - moving within failover set 145
 - object authority manager
 - description 92
 - disabling 95
 - qm.ini files 73
 - starting within failover set 144
- queue managers
 - accidental deletion of default 207
 - activating a backup queue manager 172
 - attributes, changing 38
 - attributes, displaying 37
 - backing up queue manager data 169
 - CCSID, changing 70
 - changing the CCSID 70
 - changing the default queue manager 25
 - command server 63
 - configuration files, backing up 26
 - creating a default queue manager 25
 - creating a queue manager 22, 205
 - default for each node 23
 - deleting a queue manager 28
 - deleting a queue manager (dlmqm)
 - command 209
 - description of 8
 - display queue managers (dspmq)
 - command 215
 - dumping formatted system log (dmpmqlog) command 214
 - dumping the contents of a recovery log 175
 - end queue manager (endmqm)
 - command 235
 - extending queue manager facilities 11
 - guidelines for creating a queue manager 22
 - immediate shutdown 27
 - limiting the numbers of 23
 - linear logging 159
 - log maintenance, recovery 157
 - MVS/ESA queue manager 64
 - name transformation 17
 - objects used in MQI calls 8
 - oldest log required to restart 249
 - preemptive shutdown 27

- queue managers (*continued*)
 - preparing for remote administration 59
 - queue manager aliases 68
 - quiesced shutdown 26
 - recording media images 167
 - remote administration 57
 - removing a queue manager manually 285
 - restoring queue manager data 169, 170
 - reverting to the original default 25
 - specifying unique names for 22
 - starting a queue manager 26
 - starting a queue manager automatically 26
 - starting a queue manager, strmqm command 269
 - stopping a queue manager manually 285
- QueueManager stanza, mqs.ini 78
- queues
 - alias 43
 - application queues 51
 - attributes 5
 - authorizations to 98
 - browsing 41
 - changing queue attributes 40
 - clearing local queues 41
 - current queue depth, determining 40
 - dead-letter, defining 39
 - defaults, transmission queues 24
 - defining WebSphere MQ queues 5
 - definition of 2
 - deleting a local queue 41
 - distributed, incorrect output from 185
 - dynamic (temporary) queues 3
 - extending queue manager facilities 11
 - for MQSeries applications 29
 - initiation queues 52
 - local definition of a remote queue 65
 - local queues 8
 - local, working with 38
 - model queues 3, 44, 45
 - multiple queues 8
 - predefined (permanent) queues 3
 - preparing transmission queues for remote administration 60
 - queue manager aliases 68
 - queue managers, description of 8
 - remote queue objects 68
 - reply-to queues 68
 - retrieving messages from 3
 - specific local queues used by WebSphere MQ 6
 - specifying dead-letter queues 23
 - specifying undelivered-message 23
- quiesced shutdown of a queue manager 26
- preemptive shutdown 27
- quit command (MONMQ) 321

R

- railroad diagrams, how to read 200

- rcdmqimg (record media image) command
 - examples 250
 - format 248
 - parameters 249
 - purpose 248
 - related commands 250
 - return codes 250
- rcremqobj (recreate object) command
 - examples 252
 - format 250
 - parameters 250
 - purpose 250
 - related commands 252
 - return codes 251
- REASON keyword, rules table 129
- receiver channel, automatic definition of 62
- recovery
 - activating a backup queue manager 172
 - automatic media recovery failure, scenario 175
 - backing up queue manager data 169
 - backing up WebSphere MQ 170
 - disk drive failure, scenario 173
 - making sure messages are not lost using logs 157
 - media images, recovering 167, 168
 - recovering a damaged queue manager object, scenario 174
 - recovering a damaged single object, scenario 175
 - recovering damaged objects at other times 169
 - recovering damaged objects during start up 168
 - recovering from problems 166
 - restoring queue manager data 170
 - scenarios 173
 - using the log for recovery 166
- redirecting input and output, MQSC commands 32
- remote
 - security considerations 99
- remote administration
 - administering a remote queue manager from a local one 59
 - command server 57, 63
 - defining channels and transmission queues 61
 - definition of remote administration 15
 - initial problems 65
 - of objects 57
 - preparing channels for 60
 - preparing queue managers for 59
 - preparing transmission queues for 60
- remote issuing of MQSC commands 64
- remote queue objects 68
- remote queues
 - as reply-to queue aliases 68
 - authorizations to 98
 - defining remote queues 65
 - recommendations for remote queuing 65
- remote queuing 57

- removing a queue manager
 - manually 285
- REPLACE attribute, DEFINE
 - commands 34
- reply-to queue aliases 68
- reply-to queues
 - description of 7
 - reply-to queue aliases 68
- REPLYQ keyword, rules table 129
- REPLYQM keyword, rules table 129
- resources
 - protected 94
 - updating under syncpoint control 13
 - why protect 91
- restarting a queue manager
 - oldest logs required 249
- restoring queue manager data 169
- restrictions
 - access to MQM objects 91
 - database coordination support 112
 - on object names 199
- retrieval algorithms for messages 3
- RETRY keyword, rules table 130
- RETRYINT keyword, rules tables 127
- return codes
 - crtmqcvx command 204
 - crtmqm command 208
 - dltmqm command 210
 - dspmqr command 216
 - dspmqrsv command 220
 - dspmqrfls command 222
 - dspmqrte command 229
 - dspmqrtrn command 232
 - dspmqrver command 233
 - endmqcsv command 234
 - endmqqlsr command 235
 - endmqm command 237
 - endmqtrc command 238
 - mqftrcv command 239
 - mqftrcvc command 241
 - mqftsnd command 244
 - mqftsndc command 246
 - problem determination 181
 - rcdmqimg command 250
 - rcrmqobj command 251
 - rsvmqtrn command 253
 - runmqchi command 254
 - runmqchl command 254
 - runmqqlsr command 257
 - runmqsc command 259
 - runmqtmc command 260
 - runmqtrm command 261
 - setmqaut command 266
 - strmqcsv command 269
 - strmqm command 270
- rights identifier
 - default for authority 94
 - default, nobody 94
 - MQM 91
- rights identifiers, for authority 93
- rsvmqtrn (resolve WebSphere MQ
 - transactions) command
 - format 252
 - parameters 252
 - purpose 252
 - related commands 253
 - return codes 253

- rules table (DLQ handler)
 - ACTION keyword 129
 - action keywords 129
 - APPLIDAT keyword 128
 - APPLNAME keyword 128
 - APPLTYPE keyword 128
 - control-data entry 126
 - conventions 130
 - description of 126
 - DESTQ keyword 128
 - DESTQM keyword 128
 - example of a rules table 133
 - FEEDBACK keyword 128
 - FORMAT keyword 128
 - FWDQ keyword 129
 - FWDQM keyword 130
 - HEADER keyword 130
 - INPUTQ keyword 126
 - INPUTQM keyword 127
 - MSGTYPE keyword 128
 - pattern-matching keywords 128
 - patterns and actions 127
 - PERSIST keyword 129
 - processing rules 132
 - PUTAUT keyword 130
 - REASON keyword 129
 - REPLYQ keyword 129
 - REPLYQM keyword 129
 - RETRY keyword 130
 - RETRYINT keyword 127
 - syntax rules 131
 - USERID keyword 129
 - WAIT keyword 127
- runmqchi (run channel initiator)
 - command
 - format 253
 - parameters 253
 - purpose 253
 - return codes 254
- runmqchl (run channel) command
 - format 254
 - parameters 254
 - purpose 254
 - return codes 254
- runmqdlq (run DLQ handler) command
 - format 255
 - parameters 255
 - purpose 255
 - run DLQ handler (runmqdlq)
 - command 125
 - usage 255
- runmqfm command 139
- runmqqlsr (run listener) command
 - example 257
 - format 256
 - parameters 256
 - purpose 256
 - return codes 257
- runmqsc
 - problems 35
 - verifying 35
- runmqsc (run WebSphere MQ
 - commands) command
 - ending 32
 - examples 259
 - feedback 32
 - format 258

- runmqsc (run WebSphere MQ
 - commands) command (*continued*)
 - indirect mode 64
 - parameters 258
 - purpose 258
 - redirecting input and output 32
 - return codes 259
 - usage 258
 - using 32
 - using interactively 31
- runmqtmc (start client trigger monitor)
 - command
 - examples 260
 - format 260
 - parameters 260
 - purpose 260
 - return codes 260
- runmqtrm (start trigger monitor)
 - command
 - format 261
 - parameters 261
 - purpose 261
 - return codes 261

S

- sample
 - MQSC files 289
 - programs, using 289
 - trace data 195
- scripts
 - MONMQ 323
- secure sockets layer
 - OpenSSL setup 107
- secure sockets layer (SSL)
 - overview 13
- security 91
 - channel security 13
 - enabling 95
 - failover set monitor 148
 - name service security, overview 13
 - OAM 13
 - object authority manager (OAM) 13
 - protecting log files 169
 - remote 99
 - restoring queue manager data 169
 - ssl
 - understanding 107
 - SSL 13
 - using the commands 95, 97
 - using the grant or revoke authority
 - (setmqaut) command 261
- select command (MONMQ) 313
- server-connection channel, automatic
 - definition of 62
- servers 11
- service component 12
- service objects
 - description of 10
- Service stanza, qm.ini 79
- ServiceComponent stanza, qm.ini 80
- services 45
- set color command (MONMQ) 318
- set depth command (MONMQ) 316
- set free command (MONMQ) 317
- set mask command (MONMQ) 317
- set output command (MONMQ) 319

- setmqaut (grant or revoke authority)
 - command
 - examples 266
 - format 261
 - parameters 264
 - purpose 261
 - related commands 267
 - return codes 266
 - usage 263
- setmqaut command
 - installable services 97
 - using 95, 96
- setmqprd
 - format 268
 - parameters 268
- setmqrat (set/reset authority) command
 - examples 229
 - parameters 224
 - purpose 222
 - return codes 229
- show channels command (MONMQ) 303
- show components command (MONMQ) 310
- show events command (MONMQ) 308
- show functions command (MONMQ) 311
- show globals command (MONMQ) 306
- show history command (MONMQ) 305
- show mask command (MONMQ) 304
- show memory command (MONMQ) 309
- show mutex command (MONMQ) 307
- show processes command (MONMQ) 305
- show segment command (MONMQ) 303
- show stack command (MONMQ) 304
- shutting down a queue manager
 - a queue manager, quiesced 26
 - immediate 27
 - preemptive 27
- Solaris
 - MQAI support for 56
- specifying coded character sets 69
- SSL
 - OpenSSL setup 107
- stanzas
 - AllQueueManagers, mqs.ini 73
 - ApiExitCommon, mqs.ini 78
 - ApiExitLocal, qm.ini 86
 - ApiExitTemplate, mqs.ini 78
 - Channels, qm.ini 83
 - ClientExitPath 75
 - DefaultQueueManager 75
 - ExitPath, qm.ini 86
 - ExitProperties, 75
 - Log, qm.ini 80
 - LogDefaults, mqs.ini 76
 - LU62, qm.ini 85
 - QueueManager, mqs.ini 78
 - Service, qm.ini 79
 - ServiceComponent, qm.ini 80
 - TCP, qm.ini 85
 - XAResourceManager, qm.ini 82
- Start command procedure 142

- StartCommand procedure
 - template 294
- starting
 - a channel 62
 - a command server 63
 - a listener 62
 - a queue manager 26
 - a queue manager automatically 26
 - queue manager within failover
 - set 144
- stdin, on runmqsc 32
- stdout, on runmqsc 32
- stopping
 - a queue manager manually 285
 - command server 63
- strmqcsv (start command server)
 - command
 - examples 269
 - format 268
 - parameters 268
 - purpose 268
 - related commands 269
 - return codes 269
- strmqm (start queue manager) command
 - examples 271
 - format 269
 - parameters 269
 - purpose 232, 268, 269
 - related commands 271
 - return codes 233, 270
- strmqtrc (start WebSphere MQ trace)
 - command
 - purpose 238
- switch load files, creating 113
- syncpoint, performance
 - considerations 189
- syntax
 - diagrams, how to read 200
- syntax, help with 201
- system
 - default objects 273
- system default objects 10

T

- TCP stanza, qm.ini 85
- TCP/IP
 - configured by StartCommand
 - procedure 142
 - configuring 85
 - connection rejected 86
 - Digital TCP/IP Services for
 - OpenVMS 143
 - mandatory for OpenVMS cluster
 - operation 140
 - supports OpenVMS clusters 138
- temporary (dynamic) queues 3
- Tidy command procedure 142
- TidyCommand procedure
 - template 297
- time-independent applications 1
- timed out responses from MQSC
 - commands 64
- timestamp
 - disable timestamp (MONMQ) 315
 - enable timestamp (MONMQ) 315

- trace
 - data sample 195
 - disable trace (MONMQ) 316
 - enable trace (MONMQ) 315
 - exit trace (MONMQ) 321
 - performance considerations 194
 - quit MONMQ trace 321
 - sample MONMQ session 324
 - specify component or function 313
 - start trace (MONMQ) 313
 - start when process starts 312
 - stop trace (MONMQ) 313
 - using MONMQ 299
- trace start command (MONMQ) 313
- trace stop command (MONMQ) 313
- transactional support
 - transactional support 111
 - updating under syncpoint control 13
- transactions
 - display WebSphere MQ transactions
 - (dspmqtrn) command 231
 - using the resolve WebSphere MQ
 - (rsvmqtrn command) 252
- transmission queues
 - cluster transmission queues 7
 - creating 67
 - default 24
 - default transmission queues 68
 - defining transmission queues remote
 - administration 61
 - description of 6
 - preparing transmission queues for
 - remote administration 60
- triggering 290
 - defining an application queue for
 - triggering 51
 - managing objects for triggering 51
 - message-driven processing 1
 - start client trigger monitor
 - (runmqtrmc) command 260
 - start trigger monitor (runmqtrm)
 - command 261
- trusted application 337
 - bindings 337

U

- unauthorized access, protecting from 91
- unit of work
 - definition of 111
 - explicit resynchronization of
 - (rsvmqtrn command) 120
- units of work
 - mixed outcomes 121
- UNIX operating system
 - object authority manager (OAM) 13
 - queue managers, deleting 287
- updating coded character sets 69
- user exit 335
 - cluster workload 335
- user exits
 - channel exits 12
 - data conversion exits 12
- user ID
 - authority 91
 - authorization 98
 - belonging to group nobody 94

- user ID (*continued*)
 - for authorization 98
 - OpenVMS logged-in user 98
 - principals 93
- user-defined message formats 70
- USERID keyword, rules table 129

V

- verifying MQSC commands 35

W

- WAIT keyword, rules table 127
- watcher failover monitor
 - description 139
 - halting 147
- WebSphere MQ 232, 268
 - attributes of MQSC commands 56
 - commands 30
 - issuing MQSC commands using an ASCII file 30
 - rights identifier, MQM 91
 - runmqsc command, to issue MQSC commands 30
- WebSphere MQ commands
 - attributes of 56
 - ending interactive input 32
 - escape PCFs 56
 - issuing interactively 31
 - issuing MQSC commands
 - remotely 64
 - object attribute names 4, 17
 - overview 16, 30
 - problems using MQSC commands
 - remotely 65
 - redirecting input and output 32
 - runmqsc control command,
 - modes 16, 30
 - syntax errors 32
 - timed out command responses 64
 - using 32
- WebSphere MQ queues, defining 5
- Windows operating system
 - adding a queue manager to 26
 - deleting queue managers 286
 - deletions from automatic startup
 - list 287
 - MQAI support for 56
 - object authority manager (OAM) 13
- Windows Registry
 - deleting queue managers in
 - Windows 286
 - deletions from automatic startup
 - list 287
 - using in problem determination 189
- working with 45

X

- XAResourceManager stanza, qm.ini 82

Sending your comments to IBM

If you especially like or dislike anything about this book, please use one of the methods listed below to send your comments to IBM.

Feel free to comment on what you regard as specific errors or omissions, and on the accuracy, organization, subject matter, or completeness of this book.

Please limit your comments to the information in this book and the way in which the information is presented.

To make comments about the functions of IBM products or systems, talk to your IBM representative or to your IBM authorized remarketer.

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

You can send your comments to IBM in any of the following ways:

- By mail, to this address:

User Technologies Department (MP095)
IBM United Kingdom Laboratories
Hursley Park
WINCHESTER,
Hampshire
SO21 2JN
United Kingdom

- By fax:
 - From outside the U.K., after your international access code use 44-1962-816151
 - From within the U.K., use 01962-816151
- Electronically, use the appropriate network ID:
 - IBM Mail Exchange: GBIBM2Q9 at IBMMAIL
 - IBMLink™: HURSLEY(IDRCF)
 - Internet: idrcf@hursley.ibm.com

Whichever method you use, ensure that you include:

- The publication title and order number
- The topic to which your comment applies
- Your name and address/telephone number/fax number/network ID.



GC34-6610-02

