

WebSphere MQ



System Administration Guide

Version 6.0

WebSphere MQ



System Administration Guide

Version 6.0

Note!

Before using this information and the product it supports, be sure to read the general information under Appendix I, "Notices," on page 585.

First edition (May 2005)

This edition of the book applies to the following products:

- WebSphere MQ, Version 6.0
- WebSphere MQ for z/OS, Version 6.0

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

© Copyright International Business Machines Corporation 1994, 2005. All rights reserved.

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

Contents

Figures xi

Tables xiii

About this book xv

Who this book is for xv
What you need to know to understand this book. . . xv
 Terms used in this book xvi
 Using WebSphere MQ for Windows xvi
 Using WebSphere MQ for UNIX systems xvi
 The calls MQCONN and MQCONNX xvii

Summary of changes xix

Changes for this edition (SC34-6584-00) xix
 The WebSphere MQ Explorer xix
 The File Transfer Application xix
 Control commands. xix
 WebSphere MQ objects xx
 WebSphere MQ structures xxi
 Authorization service functions xxi
 WebSphere MQ configuration information xxii
 WebSphere MQ tracing on AIX systems xxii
 Informix XA resource manager xxii
 Backup queue managers xxii

Part 1. Introduction 1

Chapter 1. Introduction to WebSphere MQ 3

WebSphere MQ and message queuing 3
 Time-independent applications 3
 Message-driven processing 3
Messages and queues 3
 What is a message? 3
 What is a queue? 4
Objects 5
 Object names 6
 Managing objects 6
 Object attributes 6
 WebSphere MQ queues 6
 WebSphere MQ queue managers 9
 Process definitions 10
 Clusters 10
 Namelists 10
 Authentication information objects 10
 Channels 11
 Client connection channels 11
 Listeners 11
 Services. 11
 System default objects 12
Clients and servers 12
 WebSphere MQ applications in a client-server environment 12
Extending queue manager facilities 13

 User exits 13
 API exits 13
 Installable services 13
Security 14
 Object Authority Manager (OAM) facility 14
 User-written or third party channel exits 14
 Channel security using SSL 14
Transactional support 14

Chapter 2. An introduction to WebSphere MQ administration 17

Local and remote administration 17
Performing administration tasks using commands 17
 Control commands 17
 WebSphere MQ Script (MQSC) commands 18
 PCF commands 18
Further methods of administration 18
 Using the WebSphere MQ Explorer 19
 Using the Windows Default Configuration application 19
 Using the Microsoft Cluster Service (MSCS) 19
Understanding WebSphere MQ file names 20
 Queue manager name transformation. 20
 Object name transformation 21

Part 2. Administration using WebSphere MQ commands. 23

Chapter 3. Managing queue managers 25

Using control commands 25
 Using control commands on Windows systems 25
 Using control commands on UNIX systems. 26
Using the WebSphere MQ Explorer 26
Creating a queue manager 26
 Guidelines for creating queue managers 27
 Creating a default queue manager. 29
 Making an existing queue manager the default 30
 Backing up configuration files after creating a queue manager 30
Starting a queue manager 31
 Starting a queue manager automatically 31
Stopping a queue manager 31
 Quiesced shutdown 32
 Immediate shutdown 32
 Preemptive shutdown 32
 If you have problems shutting down a queue manager 32
Restarting a queue manager 33
Deleting a queue manager 33

Chapter 4. Administering local WebSphere MQ objects 35

Supporting application programs that use the MQI 35

Performing local administration tasks using MQSC commands	36
WebSphere MQ object names	37
Standard input and output	37
Using MQSC commands interactively	37
Running MQSC commands from text files	38
Running MQSC commands from batch files	41
Resolving problems with MQSC commands	42
Working with queue managers	43
Displaying queue manager attributes	43
Altering queue manager attributes	44
Working with local queues	45
Defining a local queue	45
Displaying default object attributes	46
Copying a local queue definition	46
Changing local queue attributes	47
Clearing a local queue	47
Deleting a local queue	47
Browsing queues	48
Monitoring local queues with the Windows Performance Monitor	49
Enabling large queues	50
Working with alias queues	50
Defining an alias queue	50
Using other commands with alias queues	51
Working with model queues	52
Defining a model queue	52
Using other commands with model queues	52
Working with services	53
Defining a service object	53
Managing services	54
Additional environment variables	55
Replaceable inserts on service definitions	55
Examples on using service objects	56
Managing objects for triggering	58
Defining an application queue for triggering	58
Defining an initiation queue	59
Defining a process	60
Displaying attributes of a process definition	60

Chapter 5. Automating administration tasks 61

PCF commands	61
PCF object attributes	62
Escape PCFs	62
Using the MQAI to simplify the use of PCFs	62
Active Directory Services Interfaces	63

Chapter 6. Administering remote WebSphere MQ objects 65

Channels, clusters, and remote queuing	65
Remote administration using clusters	66
Remote administration from a local queue manager	67
Preparing queue managers for remote administration	67
Preparing channels and transmission queues for remote administration	68
Managing the command server for remote administration	70

Issuing MQSC commands on a remote queue manager	71
Recommendations for issuing commands remotely	72
If you have problems using MQSC commands remotely	72
Creating a local definition of a remote queue	73
Understanding how local definitions of remote queues work	73
An alternative way of putting messages on a remote queue	74
Using other commands with remote queues	75
Defining a transmission queue	75
Using remote queue definitions as aliases	76
Queue manager aliases	76
Reply-to queue aliases	76
Data conversion	76
When a queue manager cannot convert messages in built-in formats	77
File ccsid.tbl	77
Converting messages in user-defined formats	78
Changing the queue manager CCSID	78

Part 3. Administration using the WebSphere MQ Explorer 79

Chapter 7. Administration using the WebSphere MQ Explorer 81

What you can do with the WebSphere MQ Explorer	81
Remote queue managers	82
Deciding whether to use the WebSphere MQ Explorer	82
Setting up the WebSphere MQ Explorer	83
Prerequisite software	83
Required definitions for administration	83
Cluster membership	83
Security	84
Data conversion	86
Using the WebSphere MQ Explorer	86
Showing and hiding queue managers and clusters	86
Using the WebSphere MQ Taskbar application (Windows only)	87
Security on Windows	87
Using Active directory (Windows only)	88
Controlling access (Windows only)	89
Changing the password of the AMQMSRVN user account	90

Chapter 8. Extending the WebSphere MQ Explorer 93

Who this chapter is for	93
What you need to know to understand this chapter	93
Introduction	93
Importing the simple Eclipse plug-in	94
Writing an Eclipse plug-in for the WebSphere MQ Explorer	94
Accessing Javadoc	95
Utilizing extension points	95

Part 4. Configuring WebSphere MQ 101

Chapter 9. Configuring WebSphere MQ 103

Changing configuration information on Windows systems 103
 Viewing configuration information 103
 Changing configuration information on UNIX systems 104
 Editing configuration files 104
 The WebSphere MQ configuration file, mqs.ini 105
 Queue manager configuration files, qm.ini . . . 107
 Attributes for changing WebSphere MQ configuration information 109
 All queue managers 109
 Client exit path 110
 Default queue manager 110
 Exit properties 111
 Log defaults for WebSphere MQ 111
 Advanced Configuration and Power Interface (ACPI). 114
 API exits 115
 Queue managers 115
 Changing queue manager configuration information 115
 Installable services. 116
 Queue manager logs 118
 Restricted mode 120
 XA resource managers 121
 Channels 122
 LU62, NETBIOS, TCP, and SPX 124
 Exit path 126
 Queue manager error logs 127
 Queue manager default bind type 128

Chapter 10. WebSphere MQ security 129

Authority to administer WebSphere MQ 129
 Managing the mqm group 130
 Authority to work with WebSphere MQ objects 130
 When security checks are made 131
 How access control is implemented by WebSphere MQ. 131
 Identifying the user ID 132
 Alternate-user authority 134
 Context authority 134
 Connecting to WebSphere MQ using Terminal Services 135
 Creating and managing groups 136
 Windows 2000 136
 Windows XP and Windows 2003 136
 HP-UX 138
 AIX 139
 Solaris. 139
 Linux 140
 Using the OAM to control access to objects . . . 141
 Giving access to a WebSphere MQ object . . . 141

Using OAM generic profiles 142
 Displaying access settings 145
 Changing and revoking access to a WebSphere MQ object 146
 Preventing security access checks. 146
 Channel security 146
 Protecting channel initiator definitions 147
 Transmission queues 147
 Channel exits 148
 Protecting channels with SSL 148
 How authorizations work 150
 Authorizations for MQI calls 150
 Authorizations for MQSC commands in escape PCFs 153
 Authorizations for PCF commands 156
 Guidelines for Windows 2000 and Windows 2003 161
 When you get a 'group not found' error . . . 162
 When you have problems with WebSphere MQ and domain controllers 162
 Applying security template files 163
 Nested groups 164

Chapter 11. Transactional support . . 165

Introducing units of work 165
 Scenario 1: Queue manager performs the coordination. 166
 Database coordination 166
 DB2 configuration 173
 Oracle configuration 176
 Informix configuration 178
 Sybase configuration 180
 Multiple database configurations 182
 Security considerations 183
 Administration tasks 183
 XA dynamic registration. 187
 Scenario 2: Other software provides the coordination. 190
 External syncpoint coordination 190
 Using CICS 192
 Using the Microsoft Transaction Server (COM+) 194

Chapter 12. The WebSphere MQ dead-letter queue handler 195

Invoking the DLQ handler 195
 The sample DLQ handler, amqsdlq 196
 The DLQ handler rules table 196
 Control data. 196
 Rules (patterns and actions) 198
 Rules table conventions 200
 How the rules table is processed 202
 Ensuring that all DLQ messages are processed 203
 An example DLQ handler rules table 204

Chapter 13. Supporting the Microsoft Cluster Service (MSCS) 207

Introducing MSCS clusters 207
 Setting up WebSphere MQ for MSCS clustering 209
 Setup symmetry 209
 MSCS security 209
 Using multiple queue managers with MSCS . . 210

Cluster modes	210
Creating a queue manager for use with MSCS	211
Creating a queue manager from a command prompt	211
Creating a queue manager using the WebSphere MQ Explorer	211
Moving a queue manager to MSCS storage	212
Putting a queue manager under MSCS control	213
Removing a queue manager from MSCS control	215
Taking a queue manager offline from MSCS	216
Returning a queue manager from MSCS storage	216
Hints and tips on using MSCS.	216
Verifying that MSCS is working	216
Using the IBM MQSeries Service	217
Manual startup.	217
MSCS and queue managers	217
Always use MSCS to manage clusters	218
Working in Active/Active mode	218
PostOnlineCommand and PreOfflineCommand	219
Using preferred nodes	219
Performance benchmarking.	219
WebSphere MQ MSCS support utility programs	219

Part 5. Recovery and problem determination 221

Chapter 14. Recovery and restart 223

Making sure that messages are not lost (logging)	223
What logs look like	223
Types of logging	224
Using checkpointing to ensure complete recovery	226
Checkpointing with long-running transactions	227
Calculating the size of the log	228
Managing logs	229
What happens when a disk gets full.	230
Managing log files.	230
Using the log for recovery	232
Recovering from power loss or communications failures	232
Recovering damaged objects	232
Protecting WebSphere MQ log files	234
Backing up and restoring WebSphere MQ	235
Backing up queue manager data	235
Restoring queue manager data	236
Using a backup queue manager	236
Creating a backup queue manager	237
Updating a backup queue manager	237
Starting a backup queue manager	238
Recovery scenarios	239
Disk drive failures.	239
Damaged queue manager object	240
Damaged single object	240
Automatic media recovery failure	240
Dumping the contents of the log using the dmpmqlog command.	240

Chapter 15. Problem determination 245

Preliminary checks	245
Has WebSphere MQ run successfully before?	245
Are there any error messages?.	246

Are there any return codes explaining the problem?.	246
Can you reproduce the problem?.	246
Have any changes been made since the last successful run?.	246
Has the application run successfully before?	246
Problems with commands	248
Does the problem affect specific parts of the network?.	248
Does the problem occur at specific times of the day?	248
Is the problem intermittent?	248
Have you applied any service updates?	248
Looking at problems in more detail	249
Have you obtained incorrect output?	249
Have you failed to receive a response from a PCF command?	251
Are some of your queues failing?.	252
Does the problem affect only remote queues?	253
Is your application or system running slowly?	253
Application design considerations	254
Effect of message length.	254
Effect of message persistence	255
Searching for a particular message	255
Queues that contain messages of different lengths	255
Frequency of syncpoints.	255
Use of the MQPUT1 call.	255
Number of threads in use	255
Error logs	255
Error log files	256
Error log access restrictions under UNIX systems	257
Ignoring error codes under UNIX systems.	258
Ignoring error codes under Windows systems	258
Operator messages	258
Dead-letter queues	258
Configuration files and problem determination	259
Tracing	259
Tracing WebSphere MQ for Windows	259
Tracing WebSphere MQ for UNIX systems.	260
Trace files	264
Tracing Secure Sockets Layer (SSL) on UNIX systems	265
Tracing with the AIX system trace	265
First-failure support technology (FFST)	267
FFST: WebSphere MQ for Windows	267
FFST: WebSphere MQ for UNIX systems	269
Problem determination with WebSphere MQ clients	271
Terminating clients	271
Java diagnostics	271
Using com.ibm.mq.commonservices	271
Java trace and FFDC files	273

Part 6. WebSphere MQ control commands 275

Chapter 16. How to use WebSphere MQ control commands 277

Names of WebSphere MQ objects.	277
--	-----

How to read syntax diagrams	278
Example syntax diagram	279
Syntax help	280
Examples.	280

Chapter 17. The control commands 281

amqccert (check certificate chains)	283
amqmdain (WebSphere MQ services control)	285
amqtcert (transfer certificates)	291
crtmqcvx (data conversion)	297
crtmqm (create queue manager)	299
dltmqm (delete queue manager)	303
dmpmqaut (dump authority)	305
dmpmqlog (dump log)	309
dspm (display queue managers)	311
dspmqa (display authority)	312
dspmqs (display command server)	316
dspmqls (display files)	317
dspmqrte (WebSphere MQ display route application)	319
dspmqrtr (display formatted trace output)	327
dspmqrtrn (display transactions)	328
dspmqrver (display version information)	329
endmqscsv (end command server)	331
endmqslr (end listener)	333
endmqdnm (stop .NET monitor)	334
endmqm (end queue manager)	335
endmqtrc (end trace)	337
mqftapp (run File Transfer Application GUI)	338
mqftrcv (receive file on server)	339
mqftrcvc (receive file on client)	342
mqftsnd (send file from server)	345
mqftsndc (send file from client)	347
rcdmqing (record media image)	349
rcrmqobj (recreate object)	351
rsvmqtrn (resolve transactions)	353
runmqchi (run channel initiator)	355
runmqchl (run channel)	356
runmqdlq (run dead-letter queue handler)	357
runmqdnm (run .NET monitor)	358
runmqslr (run listener)	361
runmqsc (run MQSC commands)	363
runmqtmc (start client trigger monitor)	366
runmqtrm (start trigger monitor)	367
setmqaut (set or reset authority)	368
setmqcrl (set certificate revocation list (CRL) LDAP server definitions)	375
setmqscp (set service connection points)	377
strmqcfg (start WebSphere MQ Explorer)	379
strmqscsv (start command server)	380
strmqm (start queue manager)	381
strmqtrc (Start trace)	383

Chapter 18. Managing keys and certificates 387

Preparing to use the gsk7cmd command	387
gsk7cmd and runmqckm commands	387
Commands for a CMS key database only	388
Commands for CMS or PKCS #12 key databases	388
Commands for cryptographic device operations	390

gsk7cmd and runmqckm options	392
--	-----

Part 7. WebSphere MQ installable services and the API exit 395

Chapter 19. Installable services and components 401

Why installable services?	401
Functions and components	402
Entry-points	403
Return codes	403
Component data	403
Initialization	404
Primary initialization	404
Secondary initialization	404
Primary termination	404
Secondary termination	404
Configuring services and components	404
Service stanza format	405
Service stanza format for Windows systems	405
Service component stanza format	405
Creating your own service component	406
Using multiple service components	406
Example of using multiple components	407
Omitting entry points when using multiple components	407
Example of entry points used with multiple components	407

Chapter 20. Authorization service. 409

Object authority manager (OAM)	409
Defining the service to the operating system	409
Refreshing the OAM after changing a user's authorization	409
Migrating from MQSeries Version 5.1	410
Authorization service on UNIX systems	410
Configuring authorization service stanzas: UNIX systems	410
Authorization service on Windows systems	411
Configuring authorization service stanzas: Windows systems	411
Authorization service interface	412

Chapter 21. Name service 415

How the name service works	415
Name service interface	416

Chapter 22. Installable services interface reference information. 419

How the functions are shown	420
Parameters and data types	420
MQZEP – Add component entry point	421
Syntax	421
Parameters	421
C invocation	422
MQHCONFIG – Configuration handle	422
C declaration	422
PMQFUNC – Pointer to function	422
C declaration	422

MQZ_AUTHENTICATE_USER – Authenticate user	423	Syntax	466
Syntax	423	Parameters	466
Parameters	423	C invocation.	467
C invocation.	424	MQZ_SET_AUTHORITY – Set authority	468
MQZ_CHECK_AUTHORITY – Check authority	426	Syntax	468
Syntax	426	Parameters	468
Parameters	426	C invocation.	470
C invocation.	430	MQZ_SET_AUTHORITY_2 – Set authority (extended)	471
MQZ_CHECK_AUTHORITY_2 – Check authority (extended)	431	Syntax	471
Syntax	431	Parameters	471
Parameters	431	C invocation.	473
C invocation.	435	MQZ_TERM_AUTHORITY – Terminate authorization service	474
MQZ_COPY_ALL_AUTHORITY – Copy all authority	436	Syntax	474
Syntax	436	Parameters	474
Parameters	436	C invocation.	475
C invocation.	438	MQZAC – Application context	476
MQZ_DELETE_AUTHORITY – Delete authority	439	Fields	476
Syntax	439	C declaration	478
Parameters	439	MQZAD – Authority data	478
C invocation.	441	Fields	479
MQZ_ENUMERATE_AUTHORITY_DATA – Enumerate authority data	442	C declaration	481
Syntax	442	MQZED – Entity descriptor	483
Parameters	442	Fields	483
C invocation.	444	C declaration	484
MQZ_FREE_USER – Free user.	445	MQZIC – Identity context	485
Syntax	445	Fields	485
Parameters	445	C declaration	486
C invocation.	446	MQZFP – Free parameters	487
MQZ_GET_AUTHORITY – Get authority	447	Fields	487
Syntax	447	C declaration	488
Parameters	447	MQZ_DELETE_NAME – Delete name	489
C invocation.	449	Syntax	489
MQZ_GET_AUTHORITY_2 – Get authority (extended)	450	Parameters	489
Syntax	450	C invocation.	490
Parameters	450	MQZ_INIT_NAME – Initialize name service	491
C invocation.	452	Syntax	491
MQZ_GET_EXPLICIT_AUTHORITY – Get explicit authority	453	Parameters	491
Syntax	453	C invocation.	492
Parameters	453	MQZ_INSERT_NAME – Insert name	494
C invocation.	455	Syntax	494
MQZ_GET_EXPLICIT_AUTHORITY_2 – Get explicit authority (extended)	456	Parameters	494
Syntax	456	C invocation.	495
Parameters	456	MQZ_LOOKUP_NAME – Lookup name	496
C invocation.	458	Syntax	496
MQZ_INIT_AUTHORITY – Initialize authorization service.	459	Parameters	496
Syntax	459	C invocation.	497
Parameters	459	MQZ_TERM_NAME – Terminate name service	499
C invocation.	460	Syntax	499
MQZ_INQUIRE – Inquire authorization service	462	Parameters	499
Syntax	462	C invocation.	500
Parameters	462	Chapter 23. API exits	501
C invocation.	465	Why use API exits.	501
MQZ_REFRESH_CACHE – Refresh all authorizations	466	How you use API exits	501
		How to configure WebSphere MQ for API exits	501
		How to write an API exit	502
		What happens when an API exit runs?	503
		Configuring API exits	503
		Configuring API exits on UNIX systems	503

Configuring API exits on Windows systems . . . 506

Chapter 24. API exit reference information 507

General usage notes 507

MQACH – API exit chain header 509

 Fields 509

 C declaration 511

MQAXC – API exit context 512

 Fields 512

 C declaration 515

MQAXP – API exit parameter 516

 Fields 516

 C declaration 523

MQXEP – Register entry point 524

 Syntax 524

 Parameters 524

 C invocation 526

MQ_BACK_EXIT – Back out changes 527

 Syntax 527

 Parameters 527

 C invocation 527

MQ_BEGIN_EXIT – Begin unit of work 528

 Syntax 528

 Parameters 528

 C invocation 528

MQ_CLOSE_EXIT – Close object 529

 Syntax 529

 Parameters 529

 C invocation 529

MQ_CMIT_EXIT – Commit changes 530

 Syntax 530

 Parameters 530

 C invocation 530

MQ_CONNX_EXIT – Connect queue manager (extended) 531

 Syntax 531

 Parameters 531

 Usage notes 531

 C invocation 532

MQ_DISC_EXIT – Disconnect queue manager 533

 Syntax 533

 Parameters 533

 C invocation 533

MQ_GET_EXIT – Get message 534

 Syntax 534

 Parameters 534

 Usage notes 534

 C invocation 535

MQ_INIT_EXIT – Initialize exit environment 536

 Syntax 536

 Parameters 536

 Usage notes 536

 C invocation 536

MQ_INQ_EXIT – Inquire object attributes 537

 Syntax 537

 Parameters 537

 C invocation 537

MQ_OPEN_EXIT – Open object 539

 Syntax 539

 Parameters 539

 C invocation 539

MQ_PUT_EXIT – Put message 540

 Syntax 540

 Parameters 540

 Usage notes 540

 C invocation 540

MQ_PUT1_EXIT – Put one message 542

 Syntax 542

 Parameters 542

 C invocation 542

MQ_SET_EXIT – Set object attributes 544

 Syntax 544

 Parameters 544

 C invocation 544

MQ_TERM_EXIT – Terminate exit environment 546

 Syntax 546

 Parameters 546

 Usage notes 546

 C invocation 546

Part 8. Appendixes 547

Appendix A. System and default objects 549

Windows default configuration objects 551

Appendix B. Directory structure (Windows systems). 553

Appendix C. Directory structure (UNIX systems) 555

Appendix D. Stopping and removing queue managers manually. 559

Stopping a queue manager manually 559

 Stopping queue managers in WebSphere MQ for Windows 559

 Stopping queue managers in WebSphere MQ for UNIX systems 559

Removing queue managers manually 560

 Removing queue managers in WebSphere MQ for Windows 560

 Removing queue managers in WebSphere MQ for UNIX systems 561

Appendix E. File Transfer Application 563

Introduction to the File Transfer Application . . . 563

 Advantages 563

 Components 564

Installing and configuring the File Transfer Application 564

 Installing the File Transfer Application on a WebSphere MQ server 564

 Installing the File Transfer Application on a WebSphere MQ client 565

 Setup tasks 566

 Configuring the GUI 570

 File Transfer Application channel security . . . 570

Using the File Transfer Application	570
Sending a file	571
Receiving a file	571
Listing all sent and received files	571
File status	572
Using the command line	572

Appendix F. Comparing command sets 575

Queue manager commands.	575
Command server commands	575
Authority commands.	576
Cluster commands	576
Authentication information commands	576
Channel commands	577
Listener commands	577
Namelist commands	578
Process commands	578
Queue commands	579

Service commands.	579
Other commands	580

Appendix G. WebSphere MQ and UNIX System V IPC resources. 581

Clearing WebSphere MQ shared memory resources	581
Shared memory on AIX	581

Appendix H. Common Criteria 583

Environmental Considerations.	583
Configuration Requirements	584

Appendix I. Notices. 585

Trademarks	586
----------------------	-----

Index 587

Sending your comments to IBM 603

Figures

1. Queues, messages, and applications	35	19. Commented- out XAResourceManager stanza on UNIX systems	187
2. Extract from an MQSC command file	39	20. Two-computer MSCS cluster	208
3. Extract from an MQSC command report file	40	21. Checkpointing	227
4. Example script for running MQSC commands from a batch file	42	22. Checkpointing with a long-running transaction	228
5. Typical output from a DISPLAY QMGR command	44	23. Sample WebSphere MQ error log	257
6. Typical results from queue browser	49	24. Sample WebSphere MQ for Windows trace	260
7. Remote administration using MQSC commands	68	25. Sample WebSphere MQ for HP-UX trace	261
8. Setting up channels and queues for remote administration	69	26. Sample WebSphere MQ for Solaris trace	262
9. Example of a WebSphere MQ configuration file for UNIX systems	106	27. Sample WebSphere MQ for Linux trace	263
10. Example queue manager configuration file for WebSphere MQ for UNIX systems	108	28. Sample WebSphere MQ for AIX trace	264
11. Sample XAResourceManager entry for DB2 on UNIX platforms	175	29. Sample WebSphere MQ for AIX trace	267
12. Sample XAResourceManager entry for Oracle on UNIX platforms	177	30. Sample WebSphere MQ for Windows First Failure Symptom Report	268
13. Sample XAResourceManager entry for Informix on UNIX platforms	180	31. FFST report for WebSphere MQ for UNIX systems	270
14. Example contents of \$SYBASE/\$SYBASE_OCS/xa_config	181	32. Sample com.ibm.mq.commonservices properties file	273
15. Sample XAResourceManager entry for Sybase on UNIX platforms	181	33. Understanding services, components, and entry points	403
16. Sample XAResourceManager entries for multiple DB2 databases	182	34. UNIX authorization service stanzas in qm.ini	411
17. Sample XAResourceManager entries for a DB2 and Oracle database	182	35. Name service stanzas in qm.ini (for UNIX systems)	417
18. Sample dspmqtrn output	185	36. Default directory structure (UNIX systems) after a queue manager has been started	556
		37. Using the File Transfer Application to send files between remote queue managers	566
		38. Using the File Transfer Application to send files between a queue manager and a remote client	569

Tables

1. Categories of control commands	25	27. Fields in MQZAC	476
2. User rights required to launch AMQMSRVN	88	28. Fields in MQZAD	478
3. List of possible ISO CCSIDs.	110	29. Fields in MQZED	483
4. Default outstanding connection requests (TCP)	125	30. Fields in MQZIC	485
5. Default outstanding connection requests (SPX)	126	31. Fields in MQZFP	487
6. Security authorization needed for MQCONN calls.	151	32. Fields in MQACH	509
7. Security authorization needed for MQOPEN calls.	151	33. Fields in MQAXC	512
8. Security authorization needed for MQPUT1 calls.	151	34. Fields in MQAXP	516
9. Security authorization needed for MQCLOSE calls.	152	35. System and default objects: queues	549
10. What happens when a database server crashes.	167	36. System and default objects: channels	550
11. What happens when an application program crashes.	168	37. System and default objects: authentication information objects.	550
12. XA switch function pointers.	169	38. System and default objects: listeners	550
13. Summary of XA function calls	189	39. System and default objects: namelists	551
14. XA switch load file names	191	40. System and default objects: processes	551
15. CICS task termination exits	193	41. System and default objects: services	551
16. Log overhead sizes (all values are approximate).	229	42. Objects created by the Windows default configuration application.	552
17. Queue manager error log directory	256	43. WebSphere MQ for Windows directory structure	553
18. System error log directory	256	44. Content of a \queue-manager-name\ folder for WebSphere MQ for Windows	554
19. Client error log directory.	256	45. Default content of a /var/mqm/qmgrs/qmname/ directory on UNIX systems	557
20. How to read syntax diagrams	278	46. File Transfer Application command files	573
21. Specifying authorities for different object types	313	47. Queue manager commands	575
22. Specifying authorities for different object types	371	48. Commands for command server administration	575
23. Options that can be used with gsk7cmd and runmqckm	392	49. Commands for authority administration	576
24. Installable service components summary	401	50. Cluster commands	576
25. Example of entry-points for an installable service	408	51. Authentication information commands	576
26. Installable services functions	419	52. Channel commands	577
		53. Listener commands	577
		54. Namelist commands	578
		55. Process commands	578
		56. Queue commands	579
		57. Service commands	579
		58. Other commands	580

About this book

This book applies to the following:

- WebSphere® MQ for AIX®, Version 6.0
- WebSphere MQ for HP-UX, Version 6.0
- WebSphere MQ for Linux, Version 6.0
- WebSphere MQ for Solaris, Version 6.0
- WebSphere MQ for Windows®, Version 6.0

Other supported platforms (iSeries™ and zSeries®) have their own *System Administration* Guides. See *WebSphere MQ Bibliography and Glossary* for more information.

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

This book describes the system administration aspects of WebSphere MQ Version 6, and the services it provides to support commercial messaging. This includes managing the queues that applications use to receive their messages, and ensuring that applications have access to the queues that they require.

Installation of WebSphere MQ is described in the following:

- *WebSphere MQ for AIX, V6.0 Quick Beginnings*
- *WebSphere MQ for HP-UX, V6.0 Quick Beginnings*
- *WebSphere MQ for Linux, V6.0 Quick Beginnings*
- *WebSphere MQ for Solaris, V6.0 Quick Beginnings*
- *WebSphere MQ for Windows, V6.0 Quick Beginnings*

Post-installation configuration of a distributed queuing network is described in *WebSphere MQ Intercommunication*.

Who this book is for

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

What you need to know to understand this book

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

Terms used in this book

In this book, the term **Version 6 platforms** means:

- WebSphere MQ for AIX, Version 6.0
- WebSphere MQ for HP-UX, Version 6.0
- WebSphere MQ for Linux, Version 6.0
- WebSphere MQ for Solaris, Version 6.0
- WebSphere MQ for Windows, Version 6.0

The term WebSphere MQ for UNIX systems means:

- WebSphere MQ for AIX
- WebSphere MQ for HP-UX
- WebSphere MQ for Linux
- WebSphere MQ for Solaris

We also use the term UNIX systems as a general terms for the UNIX platforms.

The term WebSphere MQ for Windows means WebSphere MQ running on the Windows platforms:

- Windows 2000
- Windows XP
- Windows 2003 Server

We also use the term **Windows systems** or just **Windows** as general terms for these Windows platforms.

The term File Transfer Application server means the File Transfer Application running on a WebSphere MQ server, and the term File Transfer Application client means the File Transfer Application running on a WebSphere MQ client.

Using WebSphere MQ for Windows

Examples in this book relevant to WebSphere MQ for Windows can use New Technology file system (NTFS), high performance file system (HPFS), or file allocation table (FAT) file names.

The examples are valid for all file-naming systems, the name being transformed if necessary when the FAT system is in use. Name transformation is described in "Understanding WebSphere MQ file names" on page 20.

Using WebSphere MQ for UNIX systems

When you use WebSphere MQ on UNIX[®] systems, **MQCONN** sets up its own signal handler for the signals SIGSEGV and SIGBUS. User handlers for these signals are restored after every MQI call.

The remaining signals are handled differently.

- SIGINT
- SIGQUIT
- SIGFPE
- SIGTERM
- SIGHUP

If any handler for this group of signals receives an interrupt within an MQI call, the application handler must exit the application.

On non-threaded applications, WebSphere MQ uses the UNIX interval timer `ITIMER_REAL` to generate `SIGALRM` signals. Any previous `SIGALRM` handler and timer interval is saved on entry to MQI and restored on exit. Any timer interval set is therefore frozen while within MQI.

The calls `MQCONN` and `MQCONNX`

References in this book to the call `MQCONN` (connect queue manager) can be replaced by references to the call `MQCONNX` (connect queue manager extended); `MQCONNX` requires an additional parameter. For more information about these calls, see the *WebSphere MQ Application Programming Reference*.

Summary of changes

This section describes changes in this edition of *WebSphere MQ System Administration*.

Changes for this edition (SC34–6584–00)

The WebSphere MQ Explorer

The WebSphere MQ Explorer has been rewritten, and extended in the following ways:

- The WebSphere MQ Explorer is now available on both WebSphere MQ for Windows, and WebSphere MQ for Linux (x86 platform) systems.
- The WebSphere MQ Explorer presents information in a style consistent with that of the WebSphere MQ Eclipse platform.
- The features, and perspectives, available with the WebSphere MQ Explorer can be extended to give further administrative control over your WebSphere MQ system.
- You can use client channel definition tables when connecting to remote queue managers to secure the MQI channels using SSL.
- You can connect to a remote queue manager via another queue manager.
- You can filter lists of objects so that only the objects of interest are displayed.
- When viewing a list of objects you can specify which attributes you want to be displayed, and in which order. You can then save your configuration as a scheme, use it again, or even specify it as the default scheme.

For more information on the WebSphere MQ Explorer, see Chapter 7, “Administration using the WebSphere MQ Explorer,” on page 81.

The File Transfer Application

The File Transfer Application is available on both WebSphere MQ for Windows, and WebSphere MQ for Linux (x86 platform) servers, and clients.

The File Transfer Application allows you to send and receive files in the form of WebSphere MQ messages. You can use the File Transfer Application to send and receive any type of file in any format, for example image files, wordprocessor files, spreadsheet files, reports, letters, memos and charts.

For more information on the File Transfer Application, see Chapter 7, “Administration using the WebSphere MQ Explorer,” on page 81.

Control commands

The following are new control commands:

amqccert (check certificate chains)

The **amqccert** command is used during SSL Certificate Migration from WebSphere MQ for Windows V5.3. For more information, see “amqccert (check certificate chains)” on page 283.

Changes

amqtcert (transfer certificates)

The **amqtcert** command is used during SSL Certificate Migration from WebSphere MQ for Windows V5.3. For more information, see “amqtcert (transfer certificates)” on page 291.

dspmqrte (WebSphere MQ display route application)

Use the **dspmqrte** to invoke the WebSphere MQ display route application. For more information, see “dspmqrte (WebSphere MQ display route application)” on page 319.

dspmqver (display version information)

Use the **dspmqver** command to display WebSphere MQ version and build information. For more information, see “dspmqver (display version information)” on page 329.

runmqdnm (run .NET monitor)

Use the **runmqdnm** control command to start processing messages on an application queue with a .NET monitor. For more information, see “runmqdnm (run .NET monitor)” on page 358.

endmqdnm (stop .NET monitor)

Use the **endmqdnm** control command to stop a .NET monitor. For more information, see “endmqdnm (stop .NET monitor)” on page 334.

mqftapp (run File Transfer Application GUI)

Use the **mqftapp** command to run the File Transfer Application GUI. For more information, see “mqftapp (run File Transfer Application GUI)” on page 338.

mqftrcv (receive file on server)

This control command is available with the File Transfer Application. Use the **mqftrcv** command to view sent files, or to retrieve, extract or delete a file. For more information, see “mqftrcv (receive file on server)” on page 339.

mqftrcvc (receive file on client)

This control command is available with the File Transfer Application. Use the **mqftrcvc** command to view sent files on a connected server, or to retrieve, extract or delete a file from a connected server. For more information, see “mqftrcvc (receive file on client)” on page 342.

mqftsnd (send file from server)

This control command is available with the File Transfer Application. Use the **mqftsnd** command to send a file from a WebSphere MQ server. For more information, see “mqftsnd (send file from server)” on page 345.

mqftsndc (send file from client)

This control command is available with the File Transfer Application. Use the **mqftsndc** command to send a file from a WebSphere MQ client. For more information, see “mqftsndc (send file from client)” on page 347.

strmqcfg (start WebSphere MQ Explorer)

Use the **strmqcfg** command to start the WebSphere MQ Explorer. For more information, see “strmqcfg (start WebSphere MQ Explorer)” on page 379.

WebSphere MQ objects

The following are now fully integrated WebSphere MQ objects:

Service and listener objects

On Windows systems, all WebSphere MQ services have been migrated to WebSphere MQ service or listener objects, with the exception of ROOT

custom services. ROOT custom services are not associated with a particular queue manager, are administered using the control command **amqmdain**.

Service objects are used to define programs that are to be executed when a queue manager starts or stops, such as trigger monitors, dead letter queue handlers, or short lived programs that perform specific tasks. You can record and recover a media image of a service, and you can set authorities to service objects. For more information, see “Services” on page 11.

Listener objects allow you to manage the starting and stopping of listener processes from within the scope of a queue manager. You can now record and recover a media image of a listener, and you can set authorities to listener objects. For more information, see “Listeners” on page 11.

Channel objects

Channels are now fully integrated WebSphere MQ objects. You can now record, and recover, a media image of a channel, and you can set authorities to channel objects. For more information, see “Channels” on page 11.

Client connection channel objects

Client connection channels are now fully integrated WebSphere MQ objects. You can now record, and recover, a media image of a client connection channel, and you can set authorities to client connection channel objects. For more information, see “Client connection channels” on page 11.

WebSphere MQ structures

There are three new WebSphere MQ structures used by the installable services, as follows:

MQZAC – Application context

The MQZAC structure is used on the MQZ_AUTHENTICIAE_USER call for the *ApplicationContext* parameter. This parameter specifies data related to the calling application. For more information, see “MQZAC – Application context” on page 476.

MQZIC – Identity context

The MQZIC structure is used on the MQZ_AUTHENTICIAE_USER call for the *IdentityContext* parameter. For more information, see “MQZIC – Identity context” on page 485.

MQZFP – Free parameters

The MQZFP structure is used on the MQZ_FREE_USER call for the *FreeParms* parameter. This parameter specifies data related to resource to be freed. For more information, see “MQZFP – Free parameters” on page 487.

Authorization service functions

There are three new authorization service functions (entry points) available, as follows:

MQZ_AUTHENTICATE_USER

Authenticates a user ID and password, and can set identity context fields. For more information, see “MQZ_AUTHENTICATE_USER – Authenticate user” on page 423.

Changes

MQZ_FREE_USER

Frees associated allocated resources. For more information, see “MQZ_FREE_USER – Free user” on page 445.

MQZ_INQUIRE

Queries the supported functionality of the authorization service. For more information, see “MQZ_INQUIRE – Inquire authorization service” on page 462.

WebSphere MQ configuration information

The following new WebSphere MQ configuration information can be configured using the WebSphere MQ Explorer, or the queue manager configuration file, qm.ini:

Queue manager error logs

You can tailor the operation and contents of queue manager error logs using the Extended queue manager properties page from the WebSphere MQ Explorer, or the QMErrorLog stanza in the configuration file, qm.ini. For more information, see “Queue manager error logs” on page 127.

Queue manager default bind type

You can specify the default bind type using the Extended queue manager properties page from the WebSphere MQ Explorer, or the Connection stanza in the configuration file, qm.ini. For more information, see “Queue manager default bind type” on page 128.

WebSphere MQ tracing on AIX systems

On WebSphere MQ for AIX systems you can now perform WebSphere MQ tracing as an alternative to tracing with the AIX system trace. You enable or modify WebSphere MQ tracing on AIX systems using the **strmqtrc** control command. To stop tracing, you use the **endmqtrc** control command, and to format trace output you use the control command **dspmqtrc** control command. For more information, see “Tracing” on page 259.

Informix XA resource manager

An Informix[®] database manager can now participate in global units of works coordinated by a queue manager. For more information, see “Database coordination” on page 166.

Backup queue managers

A backup queue manager can be used to protect an existing queue manager against possible corruption caused by hardware failures. Using a backup queue manager is an alternative to backing up queue manager data.

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.

For more information on backup queue managers, see “Backing up and restoring WebSphere MQ” on page 235.

Part 1. Introduction

Chapter 1. Introduction to WebSphere MQ	3
WebSphere MQ and message queuing	3
Time-independent applications	3
Message-driven processing	3
Messages and queues	3
What is a message?	3
Message lengths	4
How do applications send and receive messages?	4
What is a queue?	4
Predefined queues and dynamic queues	5
Retrieving messages from queues	5
Objects	5
Object names	6
Managing objects	6
Object attributes	6
WebSphere MQ queues	6
Defining queues	7
Queues used by WebSphere MQ	8
WebSphere MQ queue managers	9
Process definitions	10
Clusters	10
Namelists	10
Authentication information objects	10
Channels	11
Client connection channels	11
Listeners	11
Services	11
System default objects	12
Clients and servers	12
WebSphere MQ applications in a client-server environment	12
Extending queue manager facilities	13
User exits	13
API exits	13
Installable services	13
Security	14
Object Authority Manager (OAM) facility	14
User-written or third party channel exits	14
Channel security using SSL	14
Transactional support	14
Chapter 2. An introduction to WebSphere MQ administration	17
Local and remote administration	17
Performing administration tasks using commands	17
Control commands	17
WebSphere MQ Script (MQSC) commands	18
PCF commands	18
Further methods of administration	18
Using the WebSphere MQ Explorer	19
Using the Windows Default Configuration application	19
Using the Microsoft Cluster Service (MSCS)	19
Understanding WebSphere MQ file names	20
Queue manager name transformation	20

Object name transformation	21
----------------------------	----

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"
- "Objects" on page 5
- "Clients and servers" on page 12
- "Extending queue manager facilities" on page 13
- "Security" on page 14
- "Transactional support" on page 14

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*.

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.

Messages and queues

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*.

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; “Enabling large queues” on page 50 discusses this in more detail. For information about planning the amount of storage you need for queues, see the *Quick Beginnings* guide for your platform, or visit the WebSphere MQ Web site for platform-specific performance reports:

<http://www.ibm.com/software/integration/ts/mqseries/>

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 *WebSphere MQ Intercommunication*.
- 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 17
- Chapter 3, “Managing queue managers,” on page 25
- Chapter 4, “Administering local WebSphere MQ objects,” on page 35
- Chapter 5, “Automating administration tasks,” on page 61
- Chapter 6, “Administering remote WebSphere MQ objects,” on page 65
- Chapter 7, “Administration using the WebSphere MQ Explorer,” on page 81

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 277.

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
- The WebSphere MQ Explorer
- WebSphere MQ for Windows only:
 - MQAI Component Object Model (COM) calls in a program
 - Active Directory Service interface (ADSI) calls in a program
 - The Windows Default Configuration Application

For more information about these methods, see Chapter 2, “An introduction to WebSphere MQ administration,” on page 17.

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 66.

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)

Objects

- 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*.

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.

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 58 and “runmqtrm (start trigger monitor)” on page 367. 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 67. For information about the use of transmission queues in distributed queuing, see *WebSphere MQ Intercommunication*.

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

Instrumentation events can be used to monitor queue managers independently of MQI applications.

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

For more information about instrumentation events, see *Monitoring WebSphere MQ*.

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

Objects

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 8 for more information.

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

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 Chapter 6, “Administering remote WebSphere MQ objects,” on page 65, and *WebSphere MQ Queue Manager Clusters*.

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 **dspmqls** commands with the authentication information object. For an overview of SSL and the use of the authentication information objects, see *WebSphere MQ Security*.

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 *WebSphere MQ Intercommunication*.

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 *WebSphere MQ Intercommunication*.

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 will be 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 will be run 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

Objects

that will be 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 will not monitor the program. Typically command service objects are definitions of programs that are short lived and will 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.

See Appendix A, “System and default objects,” on page 549 for more information about system defaults.

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 *WebSphere MQ Intercommunication*.

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

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 *WebSphere MQ Clients* for further information.

Extending queue manager facilities

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

- User exits
- API exits
- Installable services

User exits

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

Channel exits

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

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 *WebSphere MQ Queue Manager Clusters*.

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 Chapter 23, "API exits," on page 501 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 10, "WebSphere MQ security," on page 129.

Extending facilities

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.

See Part 7, “WebSphere MQ installable services and the API exit,” on page 395 for more information about the installable services.

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 10, “WebSphere MQ security,” on page 129.

User-written or third party channel exits

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

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 *WebSphere MQ Security*. For an overview of SSL, including pointers to the commands described in this book, see “Protecting channels with SSL” on page 148.

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 11, “Transactional support,” on page 165.

WebSphere MQ also provides support for the Microsoft® Transaction Server (COM+). “Using the Microsoft Transaction Server (COM+)” on page 194 provides information on how to set up WebSphere MQ to take advantage of COM+ support.

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”
- “Further methods of administration” on page 18
- “Understanding WebSphere MQ file names” on page 20

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.

Chapter 6, “Administering remote WebSphere MQ objects,” on page 65 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, and PCF commands. These command sets are available on all platforms covered by this book.

Control commands

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

Using MQSC commands

They are described in Chapter 3, “Managing queue managers,” on page 25.

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 F, “Comparing command sets,” on page 575.

The *WebSphere MQ Script (MQSC) Command Reference* contains a description of each MQSC command and its syntax.

See “Performing local administration tasks using MQSC commands” on page 36 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 61 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 62.

Further methods of administration

In addition to the available command sets, there are further methods of administration:

- The WebSphere MQ Explorer
- The Windows Default Configuration application (WebSphere MQ for Windows only)

- Use of the Microsoft Cluster Service (MSCS) (WebSphere MQ for Windows only)

Using the WebSphere MQ Explorer

The WebSphere MQ Explorer is an application that runs under the Eclipse platform and is available with WebSphere MQ for Windows and WebSphere MQ for Linux. The WebSphere MQ Explorer can be used to administer local queue managers, or used to administer remote queue managers on any supported platform. It provides a graphical user interface for controlling resources in a network. Using the WebSphere MQ Explorer, you can:

- Define and control various resources including queue managers, queues, process definitions, namelists, channels, client connection channels, listeners, services, and clusters.
- Start or stop a local queue manager and its associated processes.
- View queue managers and their associated objects on your workstation or from other workstations.
- Check the status of queue managers, clusters, and channels.
- Check to see which applications, users, or channels have a particular queue open, from the queue status.

You can invoke the WebSphere MQ Explorer using the `strmqcfcg` command, and on Windows from the WebSphere MQ Taskbar application, or from the Windows Start prompt.

See Chapter 7, “Administration using the WebSphere MQ Explorer,” on page 81 for more information.

Using the Windows Default Configuration application

You can use the Windows Default Configuration program from the WebSphere MQ First Steps application to create a *starter* (or default) set of WebSphere MQ objects. A summary of the default objects created is listed in Table 42 on page 552.

Using the Microsoft Cluster Service (MSCS)

Microsoft Cluster Service (MSCS) enables you to connect servers into a *cluster*, giving higher availability of data and applications, and making it easier to manage the system. MSCS can automatically detect and recover from server or application failures.

It is important not to confuse clusters in the MSCS sense with WebSphere MQ clusters. The distinction is:

WebSphere MQ clusters

are groups of two or more queue managers on one or more computers, providing automatic interconnection, and allowing queues to be shared amongst them for load balancing and redundancy.

MSCS clusters

are groups of computers, connected together and configured in such a way that, if one fails, MSCS performs a *failover*, transferring the state data of applications from the failing computer to another computer in the cluster and reinitiating their operation there.

Chapter 13, “Supporting the Microsoft Cluster Service (MSCS),” on page 207 provides detailed information on how to configure your WebSphere MQ for Windows system to use MSCS.

Understanding WebSphere MQ file names

Each WebSphere MQ queue manager, queue, process definition, namelist, channel, client connection channel, listener, service, and authentication information object is represented by a file. Because object names are not necessarily valid file names, the queue manager converts the object name into a valid file name where necessary.

The default path to a queue manager directory is as follows:

- A prefix, which is defined in the WebSphere MQ configuration information:
 - On Windows systems the default prefix is `C:\Program Files\IBM\WebSphere MQ`. This is configured in the Windows Registry.
 - On UNIX systems the default prefix is `/var/mqm`. This is configured in the `DefaultPrefix` stanza of the `mqs.ini` configuration file.

Where available, the prefix can be changed using the WebSphere MQ properties page in the WebSphere MQ Explorer, otherwise edit the `mqs.ini` configuration file manually.

- The queue manager name is 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 of a file name, 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.

The rules governing the transformation of a queue manager name are as follows:

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

For example, assuming the default prefix and a queue manager with the name `queue.manager`:

- In WebSphere MQ for Windows with NTFS or FAT32, the queue manager name becomes:
`c:\Program Files\IBM\WebSphere MQ\qmgrs\queue!manager`
- In WebSphere MQ for Windows with FAT, the queue manager name becomes:
`c:\Program Files\IBM\WebSphere MQ\qmgrs\queue!ma`
- In WebSphere MQ for UNIX systems, the queue manager name becomes:

```
/var/mqm/qmgrs/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. You might need to transform your object names. 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. Queues, process definitions, namelists, channels, client connection channels, listeners, services, and authentication information objects are represented in the file system.

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.

Part 2. Administration using WebSphere MQ commands

Chapter 3. Managing queue managers	25	Deleting a local queue	47
Using control commands	25	Browsing queues	48
Using control commands on Windows systems	25	Monitoring local queues with the Windows	
Using control commands on UNIX systems.	26	Performance Monitor	49
Using the WebSphere MQ Explorer	26	Enabling large queues	50
Creating a queue manager	26	Working with alias queues	50
Guidelines for creating queue managers	27	Defining an alias queue	50
Creating a default queue manager	29	Using other commands with alias queues	51
Making an existing queue manager the default	30	Working with model queues.	52
Windows and WebSphere MQ for Linux (x86		Defining a model queue	52
platform) systems	30	Using other commands with model queues.	52
UNIX systems	30	Working with services	53
Backing up configuration files after creating a		Defining a service object	53
queue manager	30	Managing services	54
Windows and WebSphere MQ for Linux (x86		Additional environment variables	55
platform) systems	30	Replaceable inserts on service definitions	55
UNIX systems	30	Common tokens	55
Starting a queue manager	31	Examples on using service objects	56
Starting a queue manager automatically	31	Using a server service object.	56
Stopping a queue manager	31	Using a command service object	57
Quiesced shutdown	32	Using a command service object when a	
Immediate shutdown	32	queue manager ends only	57
Preemptive shutdown	32	More on passing arguments	58
If you have problems shutting down a queue		Managing objects for triggering.	58
manager	32	Defining an application queue for triggering	58
Restarting a queue manager	33	Defining an initiation queue	59
Deleting a queue manager	33	Defining a process	60
		Displaying attributes of a process definition	60
Chapter 4. Administering local WebSphere MQ		Chapter 5. Automating administration tasks	61
objects	35	PCF commands	61
Supporting application programs that use the MQI	35	PCF object attributes	62
Performing local administration tasks using MQSC		Escape PCFs	62
commands.	36	Using the MQAI to simplify the use of PCFs	62
WebSphere MQ object names	37	Active Directory Services Interfaces	63
Case-sensitivity in MQSC commands	37	Client connection channels in the Active	
Standard input and output	37	Directory	63
Using MQSC commands interactively	37	Chapter 6. Administering remote WebSphere MQ	
Feedback from MQSC commands	38	objects	65
Ending interactive input of MQSC commands	38	Channels, clusters, and remote queuing	65
Running MQSC commands from text files	38	Remote administration using clusters.	66
MQSC command files	39	Remote administration from a local queue manager	67
MQSC command reports	40	Preparing queue managers for remote	
Running the supplied MQSC command files	40	administration	67
Using runmqsc to verify commands	41	Preparing channels and transmission queues for	
Running MQSC commands from batch files	41	remote administration	68
Resolving problems with MQSC commands	42	Defining channels, listeners, and transmission	
Working with queue managers	43	queues	69
Displaying queue manager attributes	43	Starting the listeners and channels.	70
Altering queue manager attributes.	44	Managing the command server for remote	
Working with local queues	45	administration	70
Defining a local queue.	45	Starting the command server	71
Defining a dead-letter queue	45	Displaying the status of the command server	71
Displaying default object attributes	46	Stopping a command server	71
Copying a local queue definition	46		
Changing local queue attributes	47		
Clearing a local queue.	47		

Issuing MQSC commands on a remote queue manager	71
Working with queue managers on z/OS.	72
Recommendations for issuing commands remotely	72
If you have problems using MQSC commands remotely	72
Creating a local definition of a remote queue	73
Understanding how local definitions of remote queues work	73
Example	73
An alternative way of putting messages on a remote queue.	74
Using other commands with remote queues	75
Defining a transmission queue	75
Default transmission queues.	75
Using remote queue definitions as aliases	76
Queue manager aliases	76
Reply-to queue aliases.	76
Data conversion	76
When a queue manager cannot convert messages in built-in formats	77
File ccsid.tbl	77
Default data conversion	77
Converting messages in user-defined formats	78
Changing the queue manager CCSID.	78

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”
- “Using the WebSphere MQ Explorer” on page 26
- “Creating a queue manager” on page 26
- “Starting a queue manager” on page 31
- “Stopping a queue manager” on page 31
- “Restarting a queue manager” on page 33
- “Deleting a queue manager” on page 33

Using control commands

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• The File Transfer Application

For information about administration tasks for channels, see *WebSphere MQ Intercommunication*.

Using control commands on Windows systems

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

For example, in the command:

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

- The command name can be entered in uppercase or lowercase, or a mixture of the two. These are all valid: `crtmqm`, `CRTMQM`, and `CRTmqm`.
- The flag can be entered as `-u`, `-U`, `/u`, or `/U`.

Managing queue managers

- `SYSTEM.DEAD.LETTER.QUEUE` and `jupiter.queue.manager` must be entered exactly as shown.

For more information, see Chapter 16, “How to use WebSphere MQ control commands,” on page 277.

Using control commands on UNIX systems

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

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

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

Take care to type the commands exactly as you see them in the examples.

For more information about the `crtmqm` command, see “`crtmqm` (create queue manager)” on page 299.

Using the WebSphere MQ Explorer

The WebSphere MQ Explorer is available on Windows and WebSphere MQ for Linux (x86 platform) systems only.

You can use the WebSphere MQ Explorer to perform the operations described in this chapter, except for a preemptive shutdown.

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 299). The `crtmqm` command automatically creates the required default objects and system objects (described in “System default objects” on page 12). 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 are case-sensitive. This means that you are allowed to create two queue managers with the names `jupiter.queue.manager` and `JUPITER.queue.manager`. However, it is better to avoid such complications.

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

Creating a queue manager

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 29.

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 44 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 45 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 14, “Recovery and restart,” on page 223.

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 299.

The system and default objects are listed in Appendix A, “System and default objects,” on page 549.

For WebSphere MQ for UNIX systems only

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

Making an existing queue manager the default

You can make an existing queue manager the default queue manager. The way you do this depends on the platform you are using.

Windows and WebSphere MQ for Linux (x86 platform) systems

On Windows and WebSphere MQ for Linux (x86 platform) systems, you can make an existing queue manager the default queue manager as follows:

1. Open the WebSphere MQ Explorer.
2. Right-click **IBM WebSphere MQ**, then select **Properties...**. The Properties for WebSphere MQ panel is displayed.
3. Type the name of the default queue manager into the Default queue manager name field.
4. Click OK.

UNIX systems

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.

See Chapter 9, “Configuring WebSphere MQ,” on page 103 for information about configuration files.

Backing up configuration files after creating a queue manager

WebSphere MQ configuration information is stored in the Registry for Windows systems, and in configuration files on UNIX systems.

Windows and WebSphere MQ for Linux (x86 platform) systems

If you use WebSphere MQ for Windows, configuration information is stored in the Windows Registry. Use the WebSphere MQ Explorer (see “Changing configuration information on Windows systems” on page 103) or the `amqmdain` command (see “`amqmdain` (WebSphere MQ services control)” on page 285) to make changes to the Registry.

If you use WebSphere MQ for Linux (x86 platform), configuration information is stored in configuration files. You can use the WebSphere MQ Explorer (see “Changing configuration information on Windows systems” on page 103).

UNIX systems

There are two types of configuration file:

- When you install the product, the WebSphere MQ configuration file (`mqs.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 `mqs.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 9, “Configuring WebSphere MQ,” on page 103.

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
```

On Windows and WebSphere MQ for Linux (x86 platform) systems, you can start a queue manager as follows:

1. Open the WebSphere MQ Explorer.
2. Select the queue manager from the Navigator View.
3. Click **Start**. The queue manager starts.

If the queue manager start up takes more than a few seconds WebSphere MQ will issue information messages intermittently detailing the start up progress. For more information on these messages see *WebSphere MQ Messages*.

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

Starting a queue manager automatically

In WebSphere MQ for Windows you can start a queue manager automatically when the system starts using the WebSphere MQ Explorer. For more information, see Chapter 7, “Administration using the WebSphere MQ Explorer,” on page 81.

Stopping a queue manager

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

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

On Windows and WebSphere MQ for Linux (x86 platform) systems, you can stop a queue manager as follows:

1. Open the WebSphere MQ Explorer.
2. Select the queue manager from the Navigator View.
3. Click **Stop...** The End Queue Manager panel is displayed.
4. Select Controlled, or Immediate.
5. Click **OK**. The queue manager stops.

Stopping a queue manager

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 559 for an alternative solution.

For a detailed description of the **endmqm** command and its options, see “endmqm (end queue manager)” on page 335.

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-C.

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

Restarting a queue manager

To restart a queue manager, type:

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

On Windows and WebSphere MQ for Linux (x86 platform) systems, you can restart a queue manager in the same way as starting it, as follows:

1. Open the WebSphere MQ Explorer.
2. Select the queue manager from the Navigator View.
3. Click **Start**. The queue manager restarts.

If the queue manager restart takes more than a few seconds WebSphere MQ will issue information messages intermittently detailing the start up progress. For more information on these messages see *WebSphere MQ Messages*.

Deleting a queue manager

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

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

On Windows and WebSphere MQ for Linux (x86 platform) systems, you can delete a queue manager as follows:

1. Open the WebSphere MQ Explorer.
2. Select the queue manager from the Navigator View.
Ensure the queue manager is stopped.
3. Right-click the queue manager from the Navigation View.
4. Click **Delete....** The queue manager is deleted.

Notes:

1. 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 the Enter key all the associated resources are lost.
2. 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 31). 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 303. Ensure that only trusted administrators have the authority to use this command. (For information about security, see Chapter 10, “WebSphere MQ security,” on page 129.)

If this method for deleting a queue manager does not work, see “Removing queue managers manually” on page 560 for an alternative.

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. In addition to the approaches detailed in this chapter you can use the WebSphere MQ Explorer to administer local WebSphere MQ objects, see Chapter 7, “Administration using the WebSphere MQ Explorer,” on page 81.

This chapter contains the following sections:

- “Supporting application programs that use the MQI”
- “Performing local administration tasks using MQSC commands” on page 36
- “Working with queue managers” on page 43
- “Working with local queues” on page 45
- “Working with alias queues” on page 50
- “Working with model queues” on page 52
- “Working with services” on page 53
- “Managing objects for triggering” on page 58

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.

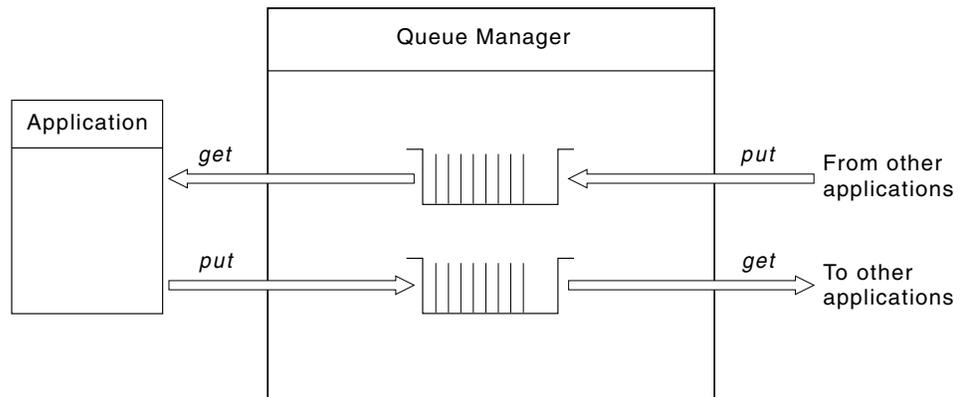


Figure 1. Queues, messages, and applications

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.

Application programs

- 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 35.

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. If you use WebSphere MQ for Windows or WebSphere MQ for Linux (x86 platform), you can also perform the operations described in this section using the WebSphere MQ Explorer. See Chapter 7, “Administration using the WebSphere MQ Explorer,” on page 81 for more information.

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 *WebSphere MQ Intercommunication*. For information about all the MQSC commands for managing queue manager objects, see *WebSphere MQ Script (MQSC) Command Reference*.

You issue MQSC commands to a queue manager using the **runmqsc** command. (For details of this command, see “runmqsc (run MQSC commands)” on page 363.) 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 38.)

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. MQSC commands are available on other platforms, including i5/OS and z/OS.

MQSC commands are summarized in Appendix F, “Comparing command sets,” on page 575. The *WebSphere MQ Script (MQSC) Command Reference* contains a description of each MQSC command and its syntax.

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* 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 25 for more information.

Standard input and output

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

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

Using MQSC commands interactively

To use MQSC commands interactively, open a command window or shell and 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.

Using MQSC commands for local administration

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* 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 stdin from a text file. (See “Standard input and output” on page 37

page 37 for information about stdin and stdout.) To do this, first create a text file containing the MQSC commands using your usual text editor. When you use the **runmqsc** command, use the redirection operators. For example, the following command runs a sequence of commands contained in the text file `myprog.in`:

```
runmqsc < myprog.in
```

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

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

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

This command invokes the MQSC commands contained in the MQSC command file `myprog.in`. Because we have not specified a queue manager name, the MQSC commands 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 40 shows the corresponding extract of the output in `myprog.out`.

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

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

MQSC command files

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

Figure 2 is an extract from an MQSC command file showing an MQSC command (`DEFINE QLOCAL`) with its attributes. The *WebSphere MQ Script (MQSC) Command Reference* contains a description of each MQSC command and its syntax.

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

Figure 2. Extract from an MQSC command file

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

Running MQSC commands

MQSC command reports

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

- A header identifying MQSC commands as the source of the report:
Starting MQSC for queue manager jupiter.queue.manager.

Where `jupiter.queue.manager` is the name of the queue manager.

- 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 MQSC for queue manager jupiter.queue.manager.
.
.
12:   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;
AMQ8006: WebSphere MQ queue created.
:
.
.
```

Figure 3. Extract from an MQSC command report file

Running the supplied MQSC command files

These MQSC command files are supplied with WebSphere MQ:

amqscos0.tst

Definitions of objects used by sample programs.

amqscic0.tst

Definitions of queues for CICS® transactions.

In WebSphere MQ for Windows, these files are located in the directory `c:\Program Files\IBM\WebSphere MQ\tools\mqsc\samples`.

On UNIX systems these files are located in the directory `opt/mqm/samp` (`usr/mqm/samp` on AIX).

The command that runs them 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. This allows you to check the syntax of the commands in your command file. This is particularly important if you are:

- Running a large number of commands from a command file.
- Using an MQSC command file many times over.

The returned report is similar to that shown in Figure 3 on page 40.

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.

Running MQSC commands from batch files

If you have very long commands, or are using a particular sequence of commands repeatedly, consider redirecting `stdin` from a batch file. To do this, first create a batch file containing the MQSC commands using your usual text editor. When you use the **runmqsc** command, use the redirection operators. The following example:

1. Creates a test queue manager, TESTQM
2. Creates a matching CLNTCONN and listener set to use TCP/IP port 1600
3. Creates a test queue, TESTQ
4. Puts a message on the queue, using the `amqsputc` sample program

Problems with MQSC commands

```
export MYTEMPQM=TESTQM
export MYPOR=1600
export MQCHLLIB=/var/mqm/qmgrs/$MQTEMPQM/@ipcc

crtmqm $MYTEMPQM
strmqm $MYTEMPQM
runmqtsr -m $MYTEMPQM -t TCP -p $MYPOR &

runmqsc $MYTEMPQM << EOF
  DEFINE CHANNEL(NTLM) CHLTYPE(SVRCONN) TRPTYPE(TCP) SCYEXIT('amqrspin(SCY_NTLM)')
  DEFINE CHANNEL(NTLM) CHLTYPE(CLNTCONN) QMNAME('$MYTEMPQM') CONNAME('127.0.0.1($MYPOR)')
  ALTER CHANNEL(NTLM) CHLTYPE(CLNTCONN) SCYEXIT('amqrspin(SCY_NTLM)')
  DEFINE QLOCAL(TESTQ)
EOF

amqspc TESTQ $MYTEMPQM << EOF
hello world
EOF

endmqm -i $MYTEMPQM
```

Figure 4. Example script for running MQSC commands from a batch file

Resolving problems with MQSC commands

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

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

- Use the < operator to redirect input from a file. If you omit this 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.
- If you redirect output to a file, use the > redirection operator. By default, the file is put in the current working directory at the time **runmqsc** is invoked. Specify a fully-qualified file name to send your output to a specific file and directory.
- Check that you have created the queue manager that is going to run the commands, by using the following command to display all queue managers:
dspmq
- The queue manager must be running. If it is not, start it; (see “Starting a queue manager” on page 31). You get an error message if you try to start a queue manager that is already running.
- Specify a queue manager name on the **runmqsc** command if you have not defined a default queue manager, or you get this error:
AMQ8146: WebSphere MQ queue manager not available.
- You cannot specify an MQSC command as a parameter of the **runmqsc** command. For example, this is not valid:
runmqsc DEFINE QLOCAL(FRED)
- You cannot enter MQSC commands before you issue the **runmqsc** command.
- You cannot run control commands from **runmqsc**. For example, you cannot issue the **strmqm** command to start a queue manager while you are running MQSC commands interactively. If you do this, you receive error messages similar to the following:

```
runmqsc
.
.
Starting MQSC for queue manager jupiter.queue.manager.

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

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

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* 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 5 on page 44.

Problems with MQSC commands

```
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\qmgs\saturn\ssl\key)
SSLRKEYC(0)                   STATACLS(QMGR)
STATCHL(OFF)                  STATINT(1800)
STATMQI(OFF)                  STATQ(OFF)
STRSTPEV(ENABLED)            SYNCPT
TRIGINT(999999999)
```

Figure 5. 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* 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);
```

Notes:

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 46.
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 47.

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.

Working with local queues

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 12, “The WebSphere MQ dead-letter queue handler,” on page 195.

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.

Notes:

1. When you use the LIKE attribute on a DEFINE command, you are copying the queue attributes only. You are not copying the messages on the queue.
2. If you define a local queue, without specifying LIKE, it is the same as DEFINE LIKE(SYSTEM.DEFAULT.LOCAL.QUEUE).

Changing local queue attributes

You can change queue attributes in two ways, using either the ALTER QLOCAL command or the DEFINE QLOCAL command with the REPLACE attribute. In “Defining a local queue” on page 45, 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

Working with local queues

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

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

In WebSphere MQ for Windows, the default file names and paths are:

Source

```
c:\Program Files\IBM\WebSphere MQ\tools\c\samples\
```

Executable

```
c:\Program files\IBM\WebSphere MQ\tools\c\samples\bin\amqsbcg.exe
```

In WebSphere MQ for UNIX, the default file names and paths are:

Source

```
/opt/mqm/samp/amqsbcg0.c (/usr/mqm/samp/amqsbcg0.c on AIX)
```

Executable

```
/opt/mqm/samp/bin/amqsbcg (/usr/mqm/samp/bin/amqsbcg on AIX)
```

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

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

Typical results from this command are shown in Figure 6 on page 49.

Enabling large queues

WebSphere MQ supports queues larger than 2 GB. On Windows systems, support for large files is available without any additional enablement. On AIX, HP-UX, Linux[®], and Solaris systems, you need to explicitly enable large file support before you can create queue files larger than 2 GB. See your operating system documentation for information on how to do this.

Some utilities, such as tar, cannot cope with files greater than 2 GB. Before enabling large file support, check your operating system documentation for information on restrictions on utilities you use.

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 45).
- A local definition of a remote queue (see “Creating a local definition of a remote queue” on page 73).

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* 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 manger. 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 will be 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.

Working with services

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 an instance of service object is started, the environment in which the service process is started is inherited from the environment of the queue manager. You can define additional environment variables to be set in the environment of the service process by adding the variables you want defining to the file, `service.env`.

The file, `service.env`, is located in `/var/mqm` on UNIX systems, and is the data directory selected during installation on Windows systems.

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 will automatically be 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 AIX systems, the install location is `/usr/mqm/`
- On Solaris, HP-UX, or Linux systems, the install location is `/opt/mqm`

Working with services

- On Windows systems, the install location is the install directory selected during the installation of WebSphere MQ

MQ_DATA_PATH

The location of the WebSphere MQ data directory:

- On UNIX systems, the WebSphere MQ data directory location is `/var/mqm/`
- On Windows systems, the location of the WebSphere MQ data directory is the data directory selected during the installation of WebSphere MQ

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.

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/runqmtrm') +
  STARTARG('-m +QMNAME+ -i 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.

- An instance of the server service object will execute when the queue manager is nest started. However, we will start an instance of the server service object immediately with the following MQSC command:

```
START SERVICE(S1)
```

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

```
DISPLAY SVSTATUS(S1)
```

- 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+ -i JUPITER.INITIATION.QUEUE')
```

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

- 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 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:

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

```
DEFINE SERVICE(S2) +
    CONTROL(QMGR) +
    SERVTYPE(COMMAND) +
    STARTCMD('/usr/bin/logger') +
    STARTARG('Queue manager +QMNAME+ starting') +
    STOPCMD('/usr/bin/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:

Working with services

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

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

Where:

+MQ_INSTALL_PATH+ is a token representing the installation directory.

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\runsrv.exe')  
  STARTARG('-m +QMNAME+ -d "C:\Program Files\Tools\ "')  
  STDOUT('C:\Program Files\Tools\+MQ_SERVICE_NAME+.out')
```

```
DEFINE SERVICE(S4) +  
  CONTROL(QMGR) +  
  SERVTYPE(SERVER) +  
  STARTCMD('C:\Program Files\Tools\runsrv.exe') +  
  STARTARG('-m +QMNAME+ -d "C:\Program Files\Tools\ "') +  
  STDOUT('C:\Program Files\Tools\+MQ_SERVICE_NAME+.out')
```

Where:

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

"C:\Program Files\Tools\ " is a string containing a space, which will be passed as a single string.

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 (UNIX) +
    APPLICID ('/u/admin/test/IRMP01') +
    USERDATA ('open, close, 235')
```

Where:

MOTOR.INSURANCE.QUOTE.PROCESS

Is the name of the process definition.

DESCR ('Insurance request message processing')

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 (UNIX)

Is the type of application to be started.

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

Is the name of the application executable file, specified as a fully qualified file name. In Windows systems, a typical APPLICID value would be c:\appl\test\irmp01.exe.

USERDATA ('open, close, 235')

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

Displaying attributes of a process definition

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

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

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

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

Chapter 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.

This chapter describes:

- How to use programmable command formats to automate administration tasks in PCF commands.
- Support for Microsoft's Active Directory Service Interfaces (ADSI) in "Active Directory Services Interfaces" on page 63.

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.

PCF commands

- 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 *WebSphere MQ Programmable Command Formats and Administration Interface*.

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 *WebSphere MQ Programmable Command Formats and Administration Interface*.

Using the MQAI to simplify the use of PCFs

The MQAI is an administration interface to WebSphere MQ that is available on the AIX, HP-UX, Linux, Solaris, and Windows platforms.

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

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 Interfaces provided by WebSphere MQ for Windows use the MQAI.

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 *WebSphere MQ Programmable Command Formats and Administration Interface*.

Active Directory Services Interfaces

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

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

WebSphere MQ support for the ADSI is implemented through the use of the **IBMMQSeries namespace**.

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

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

www.microsoft.com

For more information about Component Object Model (COM) interfaces, see *IBM WebSphere MQ for Windows, Version 6.0 Using the Component Object Model Interface*.

Client connection channels in the Active Directory

On Windows systems that support the Active Directory, WebSphere MQ publishes client connection channels in the Active Directory to provide dynamic client-server binding.

When client connection channel objects are defined, they are written into a client channel definition file, called AMQCLCHL.TAB by default. If the client connection channels use the TCP/IP protocol, the WebSphere MQ server also publishes them in the Active Directory. When the WebSphere MQ client determines how to connect to the server, it looks for a relevant client connection channel object definition using the following search order:

1. MQCONNX MQCD data structure
2. MQSERVER environment variable
3. client channel definition file
4. Active Directory

This order means that any current applications are not affected by any change. You can think of these entries in the Active Directory as records in the client channel definition file, and the WebSphere MQ client processes them in the same way. To configure and administer support for publishing client connection channel definitions in the Active Directory, use the **setmqscp** command, as described in “setmqscp (set service connection points)” on page 377.

Chapter 6. Administering remote WebSphere MQ objects

This chapter 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 67
- “Creating a local definition of a remote queue” on page 73
- “Using remote queue definitions as aliases” on page 76
- “Data conversion” on page 76

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
- The WebSphere MQ Explorer.
- 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 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

Channels, clusters, and remote queuing

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 “Channels” on page 122 for details of how to configure a channel to use pipelining.

“Preparing channels and transmission queues for remote administration” on page 68 tells you how to use these definitions to set up remote administration.

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

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 (CLUSDR) 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 *WebSphere MQ Queue Manager Clusters*.

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 *WebSphere MQ Programmable Command Formats and Administration Interface*.

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 UNIX systems, Windows systems, i5/OS, and z/OS.

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 7 on page 68 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 299.

Remote administration

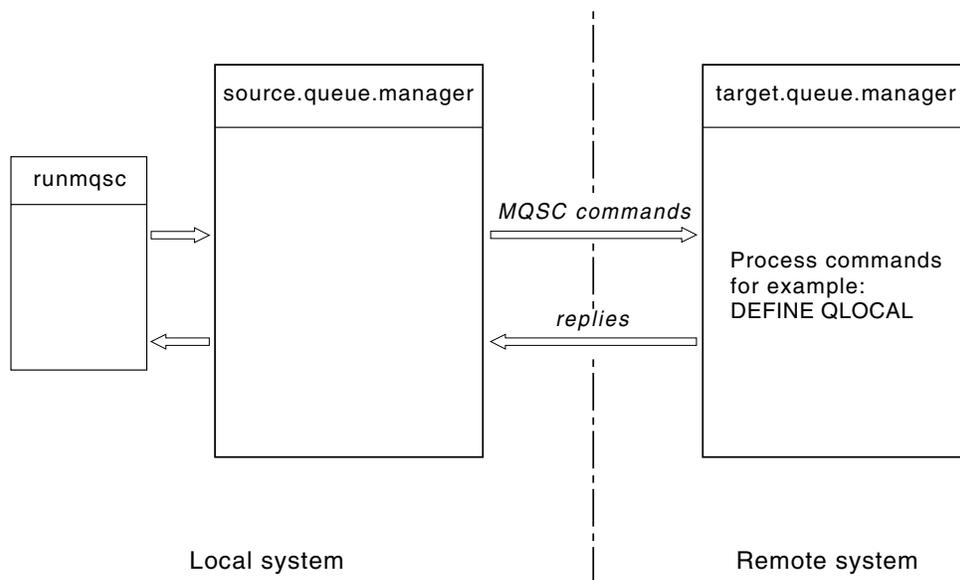


Figure 7. Remote administration using MQSC commands

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 8 on page 69 summarizes this configuration.

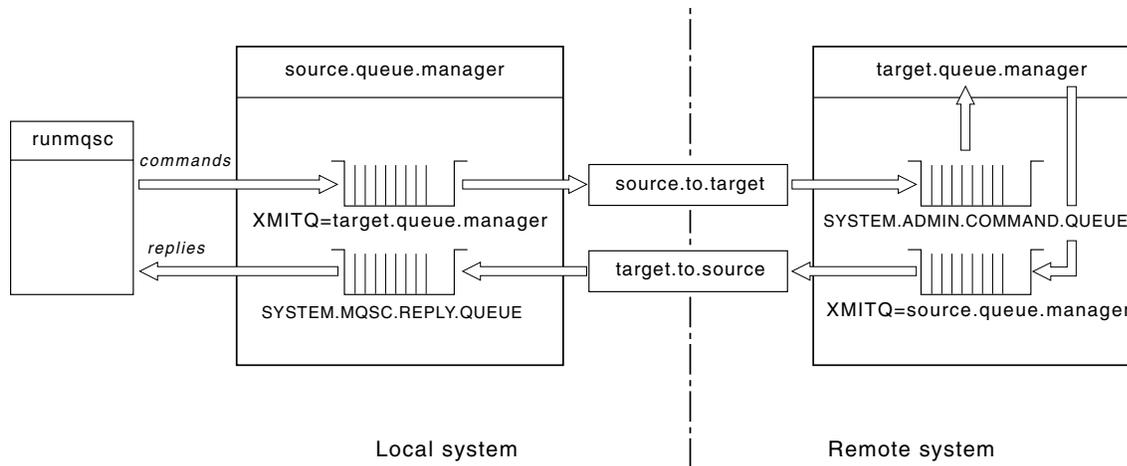


Figure 8. Setting up channels and queues for remote administration

See *WebSphere MQ Intercommunication* for more information about setting up channels.

Defining channels, listeners, 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 listener on the source queue manager:

```
DEFINE LISTENER ('source.queue.manager') +
  TRPTYPE (TCP)
```

4. 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)
```

Remote administration

3. Define the listener on the target queue manager:

```
DEFINE LISTENER ('target.queue.manager') +  
  TRPTYPE (TCP)
```

4. 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')
```

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

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

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')
```

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

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

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 *WebSphere MQ Intercommunication*. For information about automatically defining channels for clusters, see *WebSphere MQ Queue Manager Clusters*.

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.

There are separate control commands for starting and stopping the command server. Providing the command server is running, users of WebSphere MQ for Windows or WebSphere MQ for Linux (x86 platform) can perform the operations described in the following sections using the WebSphere MQ Explorer. For more information, see Chapter 7, “Administration using the WebSphere MQ Explorer,” on page 81.

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.”

Command server remote administration

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 z/OS

You can issue MQSC commands to a z/OS queue manager from a queue manager on the platforms described in this book. However, to do this, you must modify the **runmqsc** command and the channel definitions at the sender.

In particular, you add the **-x** flag to the **runmqsc** command on the source node to specify that the target queue manager is running under z/OS:

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

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 commands” on page 42.

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.

Local definition of remote queue

Field value	Description
<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 73.

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 *WebSphere MQ Intercommunication*.

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 *WebSphere MQ Intercommunication*.

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 “All queue managers” on page 109.

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

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` is used for the following purposes:

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

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

In WebSphere MQ for Windows, `ccsid.tbl` is located in directory `C:\Program Files\IBM\WebSphere MQ\conv\tabl` by default.

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

Default data conversion

If you set up channels between two machines on which data conversion is not normally supported, you must enable default data conversion for the channels to work.

To enable default data conversion, edit the `ccsid.tbl` file to specify a default EBCDIC CCSID and a default ASCII CCSID. Instructions on how to do this are included in the file. You must do this on all machines that will be connected using the channels. Restart the queue manager for the change to 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`.

Data conversion

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. If you use only the set of characters that is valid for WebSphere MQ object names (as defined in “Names of WebSphere MQ objects” on page 277) you will, in general, satisfy this requirement. Exceptions occur with EBCDIC CCSIDs 290, 930, 1279, and 5026 used in Japan, where the lowercase characters have different codes from those used in other EBCDIC CCSIDs.

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.

Part 3. Administration using the WebSphere MQ Explorer

Chapter 7. Administration using the WebSphere

MQ Explorer	81
What you can do with the WebSphere MQ Explorer	81
Remote queue managers	82
Deciding whether to use the WebSphere MQ Explorer	82
Setting up the WebSphere MQ Explorer	83
Prerequisite software	83
Required definitions for administration	83
Cluster membership	83
Security	84
Authorization to use the WebSphere MQ Explorer	84
Security for connecting to remote queue managers	85
Using a security exit	85
Using SSL security	85
Connecting via another queue manager	86
Data conversion	86
Using the WebSphere MQ Explorer	86
Showing and hiding queue managers and clusters	86
Using the WebSphere MQ Taskbar application (Windows only)	87
Using the WebSphere MQ alert monitor application (Windows only)	87
Security on Windows	87
Using Active directory (Windows only)	88
User rights required for AMQMSRVN	88
Changing the user name associated with WebSphere MQ Services	89
Controlling access (Windows only)	89
Using DCOMCNFG.EXE to change access permissions	89
Changing the password of the AMQMSRVN user account	90

Chapter 8. Extending the WebSphere MQ

Explorer	93
Who this chapter is for	93
What you need to know to understand this chapter	93
Introduction	93
Importing the simple Eclipse plug-in	94
Writing an Eclipse plug-in for the WebSphere MQ Explorer	94
Accessing Javadoc	95
Utilizing extension points	95
Register	95
Add tree node	96
Add content page	97
Add context menu item	98
Applying plug-ins to the WebSphere MQ Explorer	98

Chapter 7. Administration using the WebSphere MQ Explorer

This information applies to WebSphere MQ for Windows, and WebSphere MQ for Linux (x86 platform).

WebSphere MQ for Windows, and WebSphere MQ for Linux (x86 platform) provides an administration interface called the WebSphere MQ Explorer to perform administration tasks as an alternative to using control or MQSC commands. Appendix F, “Comparing command sets,” on page 575 shows you what you can do using the WebSphere MQ Explorer.

The WebSphere MQ Explorer allows you to perform local or remote administration of your network from a computer running Windows, or Linux (x86 platform), by pointing the WebSphere MQ Explorer at the queue managers and clusters you are interested in. The platforms and levels of WebSphere MQ that can be administered using the WebSphere MQ Explorer are described in “Remote queue managers” on page 82.

To configure remote WebSphere MQ queue managers so that WebSphere MQ Explorer can administer them, see “Required definitions for administration” on page 83.

It allows you to perform tasks, typically associated with setting up and fine tuning the working environment for WebSphere MQ, either locally or remotely within a Windows or Linux (x86 platform) system domain. It monitors the operation of WebSphere MQ servers and provides extensive error detection and recovery functions.

This chapter describes:

- “What you can do with the WebSphere MQ Explorer”
- “Setting up the WebSphere MQ Explorer” on page 83
- “Using the WebSphere MQ Explorer” on page 86
- “Security on Windows” on page 87

What you can do with the WebSphere MQ Explorer

With the WebSphere MQ Explorer, you can:

- Create and delete a queue manager (on your local machine only).
- Start and stop a queue manager (on your local machine only).
- Define, display, and alter the definitions of WebSphere MQ objects such as queues and channels.
- Browse the messages on a queue.
- Start and stop a channel.
- View status information about a channel, listener, queue, or service objects.
- View queue managers in a cluster.
- Check to see which applications, users, or channels have a particular queue open.
- Create a new queue manager cluster using the *Create New Cluster* wizard.

Introduction

- Add a queue manager to a cluster using the *Add Queue Manager to Cluster* wizard.
- Manage the authentication information object, used with Secure Sockets Layer (SSL) channel security.
- Create and delete channel initiators, trigger monitors, and listeners.
- Start or stop the command servers, channel initiators, trigger monitors, and listeners.
- Set specific services to start up automatically when a queue manager is started.
- Modify the properties of queue managers.
- Change the local default queue manager.
- Invoke the ikeyman GUI to manage secure sockets layer (SSL) certificates, associate certificates with queue managers, and configure and setup certificate stores (on your local machine only).
- Modify the parameters for any service, such as the TCP port number for a listener, or a channel initiator queue name.
- Start or stop the service trace.

The WebSphere MQ Explorer presents information in a style consistent with that of the WebSphere MQ Eclipse platform.

You perform administration tasks using a series of *Content Views* and *Property dialogs*.

Content View

A Content View is a panel that can display the following:

- Attributes, and administrative options relating to WebSphere MQ itself.
- Attributes, and administrative options relating to one or more related objects.
- Attributes, and administrative options for a cluster.

Property dialogs

A property dialog is a panel that displays attributes relating to an object in a series of fields, some of which you can edit.

You navigate through the WebSphere MQ Explorer using the *Navigator view*. The Navigator allows you to select the Content View you require.

Remote queue managers

From a Windows or Linux (x86 platform) system, the WebSphere MQ Explorer can connect to all supported queue managers with the following exceptions:

- WebSphere MQ for z/OS queue managers prior to Version 6.0.
- Currently supported MQSeries[®] V2 queue managers.

The WebSphere MQ Explorer handles the differences in the capabilities between the different command levels and platforms. However, if it encounters an attribute that it does not recognize, the attribute will not be visible.

Deciding whether to use the WebSphere MQ Explorer

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

Object names

If you use lowercase names for queue managers and other objects with the WebSphere MQ Explorer, when you work with the objects using MQSC commands, you must enclose the object names in single quotes, or WebSphere MQ will not recognize them.

Large queue managers

The WebSphere MQ Explorer works best with small queue managers. If you have a large number of objects on a single queue manager, you might experience delays while the WebSphere MQ Explorer extracts the required information to present in a view.

Clusters

WebSphere MQ clusters can potentially contain hundreds or thousands of queue managers. Because the WebSphere MQ Explorer presents the queue managers in a cluster using a tree structure. The physical size of a cluster does not affect the speed of the WebSphere MQ Explorer dramatically because the explorer does not connect to the queue managers in the cluster until you select them.

Setting up the WebSphere MQ Explorer

This section outlines the steps you need to take to set up the WebSphere MQ Explorer.

Prerequisite software

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

- The WebSphere MQ Eclipse platform (installed as part of WebSphere MQ for Windows or WebSphere MQ for Linux (x86 platform))

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

Required definitions for administration

Ensure that you have satisfied the following requirements before trying to use the WebSphere MQ Explorer. Check that:

1. A command server is running on every remotely administered queue manager.
2. A suitable TCP/IP listener object must be running on every remote queue manager. This can be the WebSphere MQ listener or, on UNIX systems, the `inetd` daemon.
3. A server-connection channel, by default named `SYSTEM.ADMIN.SVRCONN`, exists on all remote queue managers.

You can create the channel using the following MQSC command:

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

This command creates a basic channel definition. If you want a more sophisticated definition (to set up security, for example), you need additional parameters.

4. The system queue, `SYSTEM.MQEXPLORER.REPLY.MODEL`, must exist.

Cluster membership

If a queue manager is a member a cluster, then the cluster tree node will be populated automatically.

Cluster membership

If queue managers become members of clusters while the WebSphere MQ Explorer is running, then you must maintain the WebSphere MQ Explorer with up-to-date administration data about clusters so that it can communicate effectively with them and display correct cluster information when requested. In order to do this, the WebSphere MQ Explorer needs the following information:

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

With this information, the WebSphere MQ Explorer can:

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

Administration is not possible if:

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

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

Security

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

Authorization to use the WebSphere MQ Explorer

Any user can use the WebSphere MQ Explorer, however certain authorities are required to connect, access, and manage queue managers.

To perform local administrative tasks using the WebSphere MQ Explorer, a user is required to have the necessary authority to perform the administrative tasks. If the user is a member of the mqm group, the user has authority to perform all local administrative tasks.

To connect to a remote queue manager and perform remote administrative tasks using the WebSphere MQ Explorer, the user executing the WebSphere MQ Explorer is required to have:

- CONNECT authority on the target queue manager object
- INQUIRE authority on the target queue manager object
- OUTPUT authority on the queue, SYSTEM.ADMIN.COMMAND.QUEUE
- DISPLAY and INPUT authority on the queue, SYSTEM.MQEXPLORER.REPLY.MODEL
- Authority to perform the action selected

To connect to a remote queue manager on WebSphere MQ for z/OS and perform remote administrative tasks using the WebSphere MQ Explorer, the following must be provided:

- An RACF® profile for the system queue, SYSTEM.MQEXPLORER.REPLY.MODEL
- An RACF profile for the queues, AMQ.MQEXPLORER.*

In addition, the user executing the WebSphere MQ Explorer is required to have:

- RACF UPDATE authority to the system queue, SYSTEM.MQEXPLORER.REPLY.MODEL
- RACF UPDATE authority to the queues, AMQ.MQEXPLORER.*
- CONNECT authority on the target queue manager object
- Authority to perform the action selected
- READ authority to all the hlq.DISPLAY.object profiles in the MQCMD5 class

For information on how to grant authority to WebSphere MQ objects, see “Authority to work with WebSphere MQ objects” on page 130.

If a user attempts to perform an operation that they are not authorized to perform, then the target queue manager will invoke authorization failure procedures, and the operation will fail.

The default filter in the WebSphere MQ Explorer is to display all WebSphere MQ objects. If there are any WebSphere MQ objects that a user does not have DISPLAY authority to, then authorization failures are generated. If authority events are being recorded, then it is recommended that the user restrict the range of objects that are displayed to those that they have DISPLAY authority to.

Security for connecting to remote queue managers

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

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

Using a security exit

A more flexible approach is to install a security exit on the server-connection channel, typically named SYSTEM.ADMIN.SVRCONN on each queue manager that is to be administered remotely. For information on the supplied security exit, including detailed instructions on setting up and using it, see *WebSphere MQ for Windows, V6.0 Quick Beginnings*.

Using SSL security

The WebSphere MQ Explorer connects to remote queue managers using an MQI channel. If you want to secure the MQI channel using SSL security, you must establish the channel using a client channel definition table. For information how to establish an MQI channel using a client channel definition table, see the *WebSphere MQ Clients* book.

Connecting via another queue manager

The WebSphere MQ Explorer allows users to connect to a queue manager via an intermediate queue manager to which the WebSphere MQ Explorer is already connected. In this case, the WebSphere MQ Explorer puts PCF command messages to the intermediate queue manager, specifying the following:

- The *ObjectQMgrName* parameter in the object descriptor (MQOD) as the name of the target queue manager. For more information on queue name resolution, see the *WebSphere MQ Application Programming Guide*.
- The *UserIdentifier* parameter in the message descriptor (MQMD) as the local *userId*.

If the connection is then used to connect to the target queue manager via an intermediate queue manager, the *userId* is flowed in the *UserIdentifier* parameter of the message descriptor (MQMD) again. In order for the MCA listener on the target queue manager to accept this message, either the MCAUSER attribute must be set, or the *userId* must already exist with put authority.

The command server on the target queue manager puts messages to the transmission queue specifying the *userId* in the *UserIdentifier* parameter in the message descriptor (MQMD). For this put to succeed the *userId* must already exist on the target queue manager with put authority.

Data conversion

The WebSphere MQ Explorer works in CCSID 1208 (UTF-8). This enables the WebSphere MQ Explorer to display the data from remote queue managers correctly. Whether connecting to a queue manager directly, or via an intermediate queue manager, the WebSphere MQ Explorer requires all incoming messages to be converted to CCSID 1208 (UTF-8).

An error message is issued if you try to establish a connection between the WebSphere MQ Explorer and a queue manager with a CCSID that the WebSphere MQ Explorer does not recognize.

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

Using the WebSphere MQ Explorer

This section explains how to use the WebSphere MQ Explorer to do the following:

- Show or hide queue managers
- Use the WebSphere MQ Taskbar application (Windows only)
- Use the WebSphere MQ alert monitor application (Windows only)

Showing and hiding queue managers and clusters

The WebSphere MQ Explorer can display more than one queue manager at a time. The *Show/Hide Queue Manager* panel (selectable from the context menu for the Queue Managers tree node) allows you to choose whether you display information on another (remote) machine. Local queue managers are detected automatically.

To show a remote queue manager:

1. Right-click the **Queue Managers** tree node, then select *Show/Hide Queue Managers...*
2. Click **Add...** The Show/Hide Queue Managers panel is displayed.

- Fill in the name of the remote queue manager and the host name or IP address in the fields provided.

The host name or IP address is used to establish a client connection to the remote queue manager using either its default server connection channel, `SYSTEM.ADMIN.SVRCONN`, or a user defined server connection channel.

- Click **Finish**.

The Show/Hide Queue Managers panel also displays a list of all visible queue managers, and allows you to hide queue managers from the navigation view.

If the WebSphere MQ Explorer displays a queue manager that is a member of a cluster, the cluster is detected, and displayed automatically.

Using the WebSphere MQ Taskbar application (Windows only)

On Windows, the WebSphere MQ icon is in the system tray on the server and is overlaid with a color-coded status symbol, which can have one of the following meanings:

Green	Healthy; no alerts at present
Blue	Indeterminate; WebSphere MQ is starting up or shutting down
Yellow	Alert; one or more services are failing or have already failed

When you click on the icon with your right mouse button, a context menu is displayed. From this menu, select the WebSphere MQ Explorer option to bring up the WebSphere MQ Explorer.

Using the WebSphere MQ alert monitor application (Windows only)

The WebSphere MQ alert monitor is an error detection tool that identifies and records problems with WebSphere MQ on a local machine. The alert monitor displays information about the current status of the local installation of a WebSphere MQ server.

From the WebSphere MQ alert monitor, you can:

- Access the WebSphere MQ Explorer directly
- View information relating to all outstanding alerts
- Shut down the WebSphere MQ service on the local machine
- Route alert messages over the network to a configurable user account, or to a Windows workstation or server

Security on Windows

The WebSphere MQ Explorer uses *Component Object Model (COM)* technology to communicate between servers and between processes on a server.

The COM server application, *AMQMSRVN*, is shared between any client processes that need to use the WebSphere MQ COM Services.

Because *AMQMSRVN* must be shared between non-interactive and interactive logon sessions, you must launch it under a special user account. This special user account is called *MUSR_MQADMIN*. When you install WebSphere MQ and run the Prepare WebSphere MQ Wizard for the first time, it creates a local user account for *AMQMSRVN* called *MUSR_MQADMIN* with the required settings and

permissions. The password for MUSR_MQADMIN is randomly generated when the account is created, and used to configure the logon environment for AMQMSRVN.

The password is not known outside this *onetime* processing and is stored by the Windows operating system in a secure part of the Registry.

Using Active directory (Windows only)

In some network configurations, where user accounts are defined on domain controllers that are using Active Directory, the local user account MUSR_MQADMIN might not have the authority it requires to query the group membership of other domain user accounts. The Prepare WebSphere MQ Wizard identifies whether this is the case by carrying out tests and asking the user questions about the network configuration. If the local user account MUSR_MQADMIN does not have the required authority, the Prepare WebSphere MQ Wizard prompts the user for the account details of a domain user account with particular user rights. For the user rights that the domain user account requires see “User rights required for AMQMSRVN.” Once the user has entered valid account details for the domain user account into the Prepare WebSphere MQ Wizard, it configures AMQMSRVN to run under this account instead of the local user account MUSR_MQADMIN. The account details are held in the secure part of the Registry and cannot be read by users.

When the service is running, AMQMSRVN is launched and remains running for as long as the service is running. A WebSphere MQ administrator who logs onto the server after AMQMSRVN is launched can use the WebSphere MQ Explorer to administer queue managers on the server. This connects the WebSphere MQ Explorer to the existing AMQMSRVN process. These two actions need different levels of permission before they can work:

- The launch process requires a launch permission.
- The WebSphere MQ administrator requires Access permission.

User rights required for AMQMSRVN

The following table details the user rights required for the domain user account under which WebSphere MQ and specifically the AMQMSRVN DCOM object run.

Table 2. User rights required to launch AMQMSRVN

Logon as batch job	Enables WebSphere MQ Services COM server to run under this user account.
Logon as service	Enables users to set the WebSphere MQ service to logon using the configured account.
Shut down the system	Allows the WebSphere MQ Service to restart the server if configured to do so when recovery of a service fails.
Debug programs	Enables WebSphere MQ to contact processes that are secured, such as ASP and IIS applications.
Increase quotas	Required for operating system CreateProcessAsUser call.
Act as part of the operating system	Required for operating system LogonUser call.
Bypass traverse checking	Required for operating system LogonUser call.
Replace a process level token	Required for operating system LogonUser call.

Your domain user account must have these Windows user rights set as effective user rights as listed in the Local Security Policy application. If they are not, set

them using either the Local Security Policy application locally on the server, or by using the Domain Security Application domain wide.

Changing the user name associated with WebSphere MQ Services

You might need to change the user name associated with WebSphere MQ Services from MUSR_MQADMIN to something else. (For example, you might need to do this if your queue manager is associated with DB2[®], which does not accept user names of more than 8 characters.)

To change the user name :

1. Create a new user account (for example NEW_NAME)
2. Use the Prepare WebSphere MQ Wizard to enter the account details of the new user account. Alternatively, use the following command line to set the new account:

```
AMQMSRVN -user <domain>\>NEW_NAME -password <password>
```

Where NEW_NAME is the new user name you have chosen. This can be qualified by a domain name if required. WebSphere MQ allocates the correct security rights and group membership to the new user account

If for any reason you need to reset the user account back to the default MUSR_MQADMIN account, use the following command:

```
AMQMJPSE -r
```

Controlling access (Windows only)

When you install WebSphere MQ, default access permissions are set up for the AMQMSRVN process. These default access permissions grant access to the process to:

- mqm (local WebSphere MQ administrators group)
- Administrators (local administrators of this machine) (Windows only)

These permissions restrict access to the alert monitor task bar application and the **amqmdain** command to these users and groups only. Other users trying to access these functions are denied access. The WebSphere MQ Explorer uses **amqmdain** to start and end queue managers. If the WebSphere MQ Explorer is run by a user not authorized to run **amqmdain**, then an error is reported when the user attempts to start or stop a queue manager.

Before users run the WebSphere MQ Explorer, you must configure the access permissions of the objects involved. Use a tool called *DCOMCNFG.EXE*, shipped with Windows systems, to do this.

Using DCOMCNFG.EXE to change access permissions

To start DCOMCNFG.EXE on Windows 2000:

1. Click **Start**.
2. Click **Run**.
3. Type `dcomcnfg` in the open input field.
4. Click **OK**.

To start DCOMCNFG.EXE on Windows XP or Windows 2003:

1. Click **Start**.
2. Click **Run**.

Security on Windows

3. Type `dcomcnfg` in the open input field.
4. Click **OK**. The Component Services Window opens.
5. Expand **Component services**.
6. Expand **Computers**.
7. Expand **My Computer**.
8. Click the **DCOM Config** node.

A list of applications is displayed. From this list:

1. Find and highlight the **IBM MQSeries Services** entry.
2. On Windows 2000 click **Properties**.
On Windows XP or Windows 2003, right-click **IBM MQSeries Services**, then click **Properties**.
This displays information about the location of the DCOM server (`AMQMSRVN.EXE`), together with its identity and security properties.
3. Select the **Security** page to view or modify the launch, access, or configuration permissions.
4. Stop and restart the IBM MQSeries service from the Windows Services control panel for your changes to take effect. (If your changes affect a user who is currently logged on, that user must log off and on again). Do the following:
 - a. Open the control panel.
 - b. Double-click **Administrative Tools**. The Administrative Tools panel opens.
 - c. Double-click **Computer Management**. The Computer Management panel opens.
 - d. Expand **Services and Applications**.
 - e. Select **Services**. A list of services is displayed.
 - f. Right-click **IBM MQSeries**, and select **Properties**. The IBM MQSeries Properties page is displayed.
 - g. Click **Stop**. The IBM MQSeries service stops.
 - h. Click **Start**. The IBM MQSeries service restarts.

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

Changing the password of the AMQMSRVN user account

If AMQMSRVN is running under the local user account `MUSR_MQADMIN` (or another local user account), you can change the password for the account as follows:

1. Stop the MQSeries service from the Computer Management panel.
2. Close any WebSphere MQ programs that are using the AMQMSRVN COM server (this includes the alert monitor, task bar, and so on).
3. Change the `MUSR_MQADMIN` password in the same way that you would change an individual's password.
4. Use `DCOMCNFG.EXE` to bring up the properties pages for the IBM MQSeries services.
5. Select the Identity Page.
6. Modify the password given for the `MUSR_MQADMIN` user account.

If AMQMSRVN is running under a domain user account, you can also change the password for the account as follows:

1. Change the password for the domain account on the domain controller. You might need to ask your domain administrator to do this for you.
2. Use the Prepare WebSphere MQ Wizard to enter the account details including the new password.

The user account that AMQMSRVN runs under executes any MQSC commands that are issued by user interface applications, or performed automatically on system startup, shutdown, or service recovery. This user account must therefore have WebSphere MQ administration rights. By default it is added to the local **mqm** group on the server. If this membership is removed, the IBM MQSeries service will not work.

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

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

Security on Windows

Chapter 8. Extending the WebSphere MQ Explorer

This information applies to WebSphere MQ for Windows, and WebSphere MQ for Linux (x86 platform).

WebSphere MQ for Windows, and WebSphere MQ for Linux (x86 platform) provide an administration interface called the WebSphere MQ Explorer to perform administration tasks as an alternative to using control or MQSC commands. The WebSphere MQ Explorer presents information in a style consistent with that of the Eclipse framework and the other plug-in applications that Eclipse supports.

This chapter explains how the features, and perspectives, available with the WebSphere MQ Explorer can be extended to give further administrative control over your WebSphere MQ system. Through extending the WebSphere MQ Explorer system administrators have the ability to customize the WebSphere MQ Explorer to improve the way they administer WebSphere MQ.

Who this chapter is for

This chapter is for system administrators and system programmers who intend to extend, and customize, the functionality of the WebSphere MQ Explorer.

What you need to know to understand this chapter

To use this chapter, you need the following:

- A good understanding of the Eclipse framework and of the utilities associated with it.
- Experience in writing Eclipse plug-ins.

Information on how to write Eclipse plug-ins is available in the *Platform Plug-in Developers Guide*, found in Eclipse help.

1. From the WebSphere MQ Explorer toolbar click, **Help** -> **Help Contents**.
2. Click **Platform Plug-in Developers Guide**.

- A good understanding of the features available with the WebSphere MQ Explorer.

Introduction

The WebSphere MQ Explorer is Eclipse based, and as such acquires all its functionality, and perspective information, through various plug-ins supplied with WebSphere MQ. To extend the WebSphere MQ Explorer you are required to write one, or more, Eclipse plug-ins. Through writing a plug-in you can extend the function of the WebSphere MQ Explorer in the following ways:

- Add further menu options to existing context menus and associate actions with them.
- Add tree nodes to the navigation view, and associated content pages.

When writing a plug-in, you will need to supply:

The `plugin.xml` file

Use the `plugin.xml` file to specify *extension points*. Extension points provide the means by which plug-in developers can extend the functionality of the

Writing Eclipse plug-ins for the WebSphere MQ Explorer

WebSphere MQ Explorer. There are many types extension point available within the WebSphere MQ Explorer and Eclipse. Each type of extension point is used to extend the Explorer in a different way. Most extension points are associated with a Java™ jar file. For more information on the extension points available, see “Utilizing extension points” on page 95.

Java jar file

Use Java jar files contain the classes that provide the code needed to implement the added functionality specified by the extension points declared in the plugin.xml file. Every Java jar file is associated with at least one extension point.

WebSphere MQ supplies sample Eclipse plug-ins called *simple*, and *menu*. The simple plug-in utilizes all the extension points provided in the WebSphere MQ Explorer to extend the Explorer in a number of basic ways. The simple plug-in can be used as a basis for writing your own Eclipse plug-ins. For instructions of how to import the simple plug-in, see “Importing the simple Eclipse plug-in.”

Importing the simple Eclipse plug-in

To view the simple Eclipse plug-in with read and write access, do the following:

1. Open the WebSphere MQ Explorer.
2. Open the **Plug-in Development** perspective.
3. Create an empty plug-in project, giving it an appropriate name.
4. Open the properties dialog for the empty plug-in project.
5. Select the **Java Build Path** pane, then click the **Libraries** tab.
6. Click **Add External JARs...**
7. Navigate to C:\Program Files\IBM\WebSphere MQ\eclipse\plugins\com.ibm.mq.explorer.sample.simple_<version>, (or equivalent on Linux (x86 platform)), and select the file `simplesrc.zip`.
8. Click **Open**.

You have now imported the simple Eclipse plug-in.

Writing an Eclipse plug-in for the WebSphere MQ Explorer

This section details how to write an Eclipse plug-in for the WebSphere MQ Explorer. It is assumed that you have the prerequisite knowledge detailed in “What you need to know to understand this chapter” on page 93.

To write an Eclipse plug-in for the WebSphere MQ Explorer, you must utilize the extension points available to extend the functionality of the WebSphere MQ Explorer. The most common extension points are described, and accompanied by a number of code extracts from the simple plug-in to provide basic implementation examples. You must import the simple plug-in if you want access to the code that it contains. For instructions on how to import the simple plug-in, see “Importing the simple Eclipse plug-in.”

The environment in which the WebSphere MQ Explorer is extended is an event driven interface. For example, when a Register extension point is extended with an instance of a user-written class that extends the IExplorerNotify interface, the user-written class will be called back when an event occurs. For example, when a queue manager is created. Many of these notifications include a MQExtObject as

one of their arguments. An MQExtObject relates to the WebSphere MQ object that caused the event. A user-written class can call any of the MQExtObject public methods to find out about the object.

The IExplorerNotify interface, the associated MQExtObject, and other external definitions are documented in the WebSphere MQ Explorer JavaDoc. For information on how to access the WebSphere MQ Explorer JavaDoc, see “Accessing Javadoc.”

Accessing Javadoc

To access the WebSphere MQ Explorer Javadoc as HTML pages, do the following:

1. Navigate to C:\Program Files\IBM\WebSphere MQ\eclipse\plugins\com.ibm.mq.explorer.ui_<version>, (or equivalent on Linux (x86 platform)), and select the file doc.zip.
2. Extract the files from doc.zip in to an appropriate directory.
3. Double-click index.html.

The WebSphere MQ Explorer Javadoc is displayed in HTML.

Utilizing extension points

This sections describes how to implement the extension points available in Eclipse plug-ins for the WebSphere MQ Explorer.

For further information on using extension points see the WebSphere MQ Explorer help, as follows:

1. Click **Help->Help Contents**.
2. Expand **Platform Plug-in Developers Guide**.
3. Click **Programmers Guide**.

For information on how to include an extension point, see **Plugging into the workbench->Basic workbench extension points** in the Programmers Guide.

Through utilizing the available extension points, you can extend the function of the WebSphere MQ Explorer in the following ways:

- Register extension points.
- Add further menu options to existing context menus and associate actions with them.
- Add tree nodes to the navigation view, and associate content pages with them.

Multiple extension points of the same type can be included in a single plug-in. The extension points that you use will be dependent on the way in which you intend to extend the functionality of the WebSphere MQ Explorer. However, every plug-in for the WebSphere MQ Explorer must use the *register* extension point. For details on the register extension point, see “Register.”

Register

The *register* extension point is used for the following:

- To allow your plug-in to register itself with the WebSphere MQ Explorer. Every plug-in for the WebSphere MQ Explorer must include this extension point in plugin.xml. With out it, any function your plug-in adds to the WebSphere MQ Explorer will not be activated.
- To enable notify events. For information on notify events, see “Notify events” on page 96.

Writing Eclipse plug-ins for the WebSphere MQ Explorer

The following code extract is taken from the file, plugin.xml, from the simple plug-in and shows a basic implementation of the register extension point:

```
<extension
  id="com.ibm.mq.explorer.sample.simple"
  name="Simple Sample"
  point="com.ibm.mq.explorer.ui.registerplugin">
  <pluginDetails
    pluginId="com.ibm.mq.explorer.sample.simple"
    name="Simple"
    class="com.ibm.mq.explorer.sample.simple.SimpleNotify"
    enabledByDefault="true"
    description="a very simple sample plugin to Explorer"
    vendor="IBM">
  </pluginDetails>
</extension>
```

Enabling and disabling a plug-in: All plug-ins that contain the register extension point can be enabled, or disabled, within the WebSphere MQ Explorer by doing the following:

1. From the WebSphere MQ Explorer toolbar click, **Window -> Preferences**.
2. Expand **IBM WebSphere MQ**.
3. Click **Enable plug-ins**.
All registered plug-ins are displayed.
4. Select all plug-ins that should be enabled.
5. Click **OK**.

Notify events: Within the WebSphere MQ Explorer, when a WebSphere MQ object is created, or manipulated, a java object relating to the WebSphere MQ object can be generated. These java object can be used to find the name, type, and other externalized attributes of a WebSphere MQ object.

For java objects to be generated, the register extension point must specify a class. In the plugin.xml file from the simple plug-in, the class specified is as follows:
class="com.ibm.mq.explorer.sample.simple.SimpleNotify"

This class contains a number of object specific methods. When a WebSphere MQ object is created, or manipulated, the appropriate method from the notify class is called. This class can be used as a basis for writing your own class. For the methods that this class must contain refer to the WebSphere MQ Explorer JavaDoc. For information on how to access the WebSphere MQ Explorer JavaDoc, see "Accessing Javadoc" on page 95.

Add tree node

A *tree node* extension point is used to add a tree node to the navigation view and associate it with a content page.

The following code extract is taken from the file, plugin.xml, from the simple plug-in and shows a basic implementation of the tree node extension point:

```
<extension
  id="com.ibm.mq.explorer.samples.simpleTreeNode"
  name="Simple TreeNode"
  point="com.ibm.mq.explorer.ui.addtreenode">
  <treeNode
    pluginId="com.ibm.mq.explorer.sample.simple"
    name="com.ibm.mq.explorer.sample.simple"
    class="com.ibm.mq.explorer.sample.simple.SimpleTreeNodeFactory"
    treeNodeId="com.ibm.mq.explorer.sample.simple"
    sequence="888">
  </treeNode>
</extension>
```

As well as declaring the tree node extension point in plugin.xml, the following classes are needed:

- A class that contains a method that checks the id of any incoming tree node to determine whether to add sub nodes to it. This class must implement `com.ibm.mq.explorer.ui.extensions.ITreeNodeFactory`, and `IExecutableExtension`. For the methods that this class must contain refer to the WebSphere MQ Explorer JavaDoc. For information on how to access the WebSphere MQ Explorer JavaDoc, see “Accessing Javadoc” on page 95.

A working example of this class is available in the simple plug-in, called `SimpleTreeNodeFactory.java`

- A class that contains methods that return information about any new tree nodes, such as the name, id, and the associated content page class. This class must extend `com.ibm.mq.ui.extensions.TreeNode`. For the methods that this class must contain refer to the WebSphere MQ Explorer JavaDoc.

A working example of this class is available in the simple plug-in, called `SimpleTreeNode.java`.

Add content page

A *content page* extension point is used to add a content pages to the content view. A content page can be associated with a tree node.

The following code extract is taken from the file, plugin.xml, from the simple plug-in and shows a basic implementation of the content page extension point:

```
<extension
  id="com.ibm.mq.explorer.sample.simpleContentPage"
  name="Simple ContentPage"
  point="com.ibm.mq.explorer.ui.addcontentpage">
  <contentPage
    pluginId="com.ibm.mq.explorer.sample.simple"
    name="com.ibm.mq.explorer.sample.simple"
    class="com.ibm.mq.explorer.sample.simple.SimpleContentPageFactory"
    contentPageId="com.ibm.mq.explorer.sample.simple">
  </contentPage>
</extension>
```

As well as declaring the content page extension point in plugin.xml, the following classes are needed:

- A class that contains methods that perform a number of functions such as return the content page id, create the content page, and set the object to draw the page. This class must extend `com.ibm.mq.ui.extensions.ContentPage`. The class `com.ibm.mq.explorer.ui.extensions.ContentTitleBar` can be used to create a title for the content page consistent with the other content pages in the WebSphere MQ Explorer. For the methods that this class must contain refer to the WebSphere MQ Explorer JavaDoc. For information on how to access the WebSphere MQ Explorer JavaDoc, see “Accessing Javadoc” on page 95.

Writing Eclipse plug-ins for the WebSphere MQ Explorer

A working example of this class is available in the simple plug-in, called `SimpleContentPage.java`.

- A class that contains a method that returns an instance of the class extending `ContentPage`. This class must implement `com.ibm.mq.explorer.ui.extensions.IContentPageFactory`, and `IExecutableExtension`. For the methods that this class must contain refer to the WebSphere MQ Explorer JavaDoc.

A working example of this class is available in the simple plug-in, called `SimpleContentPageFactory.java`

Add context menu item

A *context menu* extension point is used to add context menu items to the WebSphere MQ Explorer.

The following code extract is taken from the file, `plugin.xml`, from the simple plug-in and shows a basic implementation of the context menu extension point:

```
<extension
  id="com.ibm.mq.explorer.sample.simple.object1"
  name="Object1"
  point="org.eclipse.ui.popupMenus">
  <objectContribution
    objectClass="com.ibm.mq.explorer.ui.extensions.MQExtObject"
    id="com.ibm.mq.explorer.sample.simple.obj1">
  <visibility>
  <and>
    <pluginState
      value="activated"
      id="com.ibm.mq.explorer.ui">
    </pluginState>
    <objectClass
      name="com.ibm.mq.explorer.ui.extensions.MQExtObject">
    </objectClass>
    <objectState
      name="PluginEnabled"
      value="com.ibm.mq.explorer.sample.simple">
    </objectState>
  </and>
  </visibility>
  <action
    label="Simple: Sample action on any MQExtObject"
    class="com.ibm.mq.explorer.sample.simple.MenuActions"
    menubarPath="additions"
    id="com.ibm.mq.explorer.sample.simple.obj.action1">
  </action>
  </objectContribution>
</extension>
```

Additional context menu items are added using the Eclipse extension point `org.eclipse.ui.popupMenus`. The `<visibility>` attribute in the above extract contains the elements that control the conditions under which the context menu item is displayed. These conditions include tests on the plug-in state, the type of object, and the state of the object. For example, a content menu item can be displayed for local queues only, or for remote queue managers only.

Applying plug-ins to the WebSphere MQ Explorer

To permanently apply updates to the WebSphere MQ Explorer provided by a plug-in, do the following:

1. With a file browser, find the plug-in file that will provide the functionality extensions to the WebSphere MQ Explorer.

Writing Eclipse plug-ins for the WebSphere MQ Explorer

2. Copy the plug-in file, and paste it into C:\Program Files\IBM\WebSphere MQ\eclipse\plugins, (or equivalent on Linux (x86 platform)).
3. Restart the WebSphere MQ Explorer.
The updates provided by the plug-in are applied to the WebSphere MQ Explorer.

Part 4. Configuring WebSphere MQ

Chapter 9. Configuring WebSphere MQ	103
Changing configuration information on Windows systems	103
Viewing configuration information	103
Changing configuration information on UNIX systems	104
Editing configuration files	104
When do you need to edit a configuration file?	104
Configuration file priorities.	105
The WebSphere MQ configuration file, mqs.ini	105
Queue manager configuration files, qm.ini	107
Attributes for changing WebSphere MQ configuration information	109
All queue managers	109
Client exit path	110
Default queue manager	110
Exit properties	111
Log defaults for WebSphere MQ	111
Advanced Configuration and Power Interface (ACPI).	114
API exits	115
Queue managers	115
Changing queue manager configuration information	115
Installable services.	116
Service components	117
Queue manager logs	118
Restricted mode	120
XA resource managers	121
Channels	122
LU62, NETBIOS, TCP, and SPX	124
Exit path	126
API exits	127
Queue manager error logs	127
Queue manager default bind type	128
Chapter 10. WebSphere MQ security	129
Authority to administer WebSphere MQ	129
Managing the mqm group	130
Authority to work with WebSphere MQ objects	130
When security checks are made	131
How access control is implemented by WebSphere MQ.	131
Identifying the user ID	132
Principals and groups	132
Windows security identifiers (SIDs)	133
Alternate-user authority	134
Context authority	134
Connecting to WebSphere MQ using Terminal Services	135
Creating and managing groups	136
Windows 2000	136
Creating a group and adding users	136
Displaying who is in a group	136
Removing a user from a group	136
Windows XP and Windows 2003	136
Creating a group	137
Adding a user to a group	137
Displaying who is in a group	137
Removing a user from a group	137
HP-UX	138
Creating a group	138
Adding a user to a group	138
Displaying who is in a group	138
Removing a user from a group	138
AIX	139
Creating a group	139
Adding a user to a group	139
Displaying who is in a group	139
Removing a user from a group	139
Solaris.	139
Creating a group	139
Adding a user to a group	140
Displaying who is in a group	140
Removing a user from a group	140
Linux	140
Creating a group	140
Adding a user to a group	140
Displaying who is in a group	141
Removing a user from a group	141
Using the OAM to control access to objects	141
Giving access to a WebSphere MQ object	141
Examples of using the command	142
Using the command with a different authorization service	142
Using OAM generic profiles	142
Using wildcard characters	143
Profile priorities	143
Dumping profile settings	144
Displaying access settings	145
Changing and revoking access to a WebSphere MQ object	146
Preventing security access checks.	146
Channel security	146
Protecting channel initiator definitions	147
Transmission queues	147
Channel exits	148
Protecting channels with SSL	148
How authorizations work	150
Authorizations for MQI calls	150
Authorizations for MQSC commands in escape PCFs	153
Authorizations for PCF commands	156
Guidelines for Windows 2000 and Windows 2003	161
When you get a 'group not found' error	162
When you have problems with WebSphere MQ and domain controllers	162
Windows 2000 domain with non-default, or Windows 2003 domain with default, security permissions	162

Configuring WebSphere MQ Services to run under a domain user	163
Applying security template files	163
Nested groups	164
Chapter 11. Transactional support	165
Introducing units of work	165
Scenario 1: Queue manager performs the coordination.	166
Database coordination	166
Restrictions	168
Switch load files	169
Configuring your system for database coordination.	170
DB2 configuration	173
Checking the DB2 environment variable settings	174
Creating the DB2 switch load file.	174
Adding resource manager configuration information for DB2	174
Changing DB2 configuration parameters	175
Oracle configuration	176
Checking the Oracle environment variable settings	176
Creating the Oracle switch load file	176
Adding resource manager configuration information for Oracle	177
Changing Oracle configuration parameters	177
Informix configuration	178
Ensuring Informix databases are created correctly	178
Checking the Informix environment variable settings	178
Creating the Informix switch load file	179
Adding resource manager configuration information for Informix	179
Sybase configuration	180
Checking the Sybase environment variable settings	180
Enabling Sybase XA support	180
Creating the Sybase switch load file	181
Adding resource manager configuration information for Sybase	181
Using multi-threaded programs with Sybase	182
Multiple database configurations	182
Security considerations	183
Administration tasks	183
In-doubt units of work	184
Displaying outstanding units of work with the dspmqtrn command.	184
Resolving outstanding units of work with the rsvmqtrn command	185
Mixed outcomes and errors.	186
Changing configuration information.	186
XA dynamic registration.	187
Error conditions	188
Summarizing XA calls	189
Scenario 2: Other software provides the coordination.	190
External syncpoint coordination	190

The WebSphere MQ XA switch structure	191
Using CICS	192
The CICS two-phase commit process	192
Using the Microsoft Transaction Server (COM+)	194

Chapter 12. The WebSphere MQ dead-letter queue handler	195
Invoking the DLQ handler	195
The sample DLQ handler, amqsdlq	196
The DLQ handler rules table	196
Control data.	196
Rules (patterns and actions)	198
The pattern-matching keywords	198
The action keywords	199
Rules table conventions	200
How the rules table is processed	202
Ensuring that all DLQ messages are processed	203
An example DLQ handler rules table	204

Chapter 13. Supporting the Microsoft Cluster Service (MSCS)	207
Introducing MSCS clusters	207
Setting up WebSphere MQ for MSCS clustering	209
Setup symmetry	209
MSCS security	209
Using multiple queue managers with MSCS	210
Cluster modes	210
Active/Passive mode.	210
Active/Active mode	211
Creating a queue manager for use with MSCS	211
Creating a queue manager from a command prompt	211
Creating a queue manager using the WebSphere MQ Explorer	211
Moving a queue manager to MSCS storage	212
Putting a queue manager under MSCS control	213
Removing a queue manager from MSCS control	215
Taking a queue manager offline from MSCS	216
Returning a queue manager from MSCS storage	216
Hints and tips on using MSCS.	216
Verifying that MSCS is working	216
Using the IBM MQSeries Service	217
Manual startup.	217
MSCS and queue managers	217
Creating a matching queue manager on the other node	217
Default queue managers.	217
Deleting a queue manager	217
Support for existing queue managers	217
Telling MSCS which queue managers to manage	218
Queue manager log files.	218
Multiple queue managers	218
Always use MSCS to manage clusters	218
Working in Active/Active mode	218
PostOnlineCommand and PreOfflineCommand	219
Using preferred nodes	219
Performance benchmarking.	219
WebSphere MQ MSCS support utility programs	219

Chapter 9. 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.

How you change this configuration information, and where WebSphere MQ stores your changes, is platform-specific:

- WebSphere MQ for Windows uses the WebSphere MQ Explorer to make changes to attribute information within the **Windows Registry**. You can also use `amqmdain` to set some Registry values, as described in "amqmdain (WebSphere MQ services control)" on page 285.
- Users on all other platforms change attribute values by editing the **WebSphere MQ configuration files**. On WebSphere MQ for Linux (x86 platform) the WebSphere MQ configuration files can be edited using the WebSphere MQ Explorer.

This chapter describes:

- "Changing configuration information on Windows systems"
- "Changing configuration information on UNIX systems" on page 104

It then describes:

- The attributes you can use to modify WebSphere MQ configuration (for all queue managers) in "Attributes for changing WebSphere MQ configuration information" on page 109
- The attributes you can use to modify the configuration of an individual queue manager in "Changing queue manager configuration information" on page 115

Changing configuration information on Windows systems

All WebSphere MQ configuration information is stored in the Windows Registry. There is a simple, or close correlation between the contents of the Windows Registry and the WebSphere MQ configuration files.

You edit configuration information from the WebSphere MQ Explorer (or by using the `amqmdain` command). **Do not** try to edit the Registry system file directly as this might adversely affect the smooth running of both your WebSphere MQ system and your Windows operating system.

Viewing configuration information

You can view a description of the keys used by the Windows Registry from the WebSphere MQ Help System. You can access the WebSphere MQ Help System from:

- An icon in the Windows Start menu
- Help in the WebSphere MQ Explorer

Changing configuration information on UNIX systems

On UNIX platforms, you can change WebSphere MQ configuration attributes within:

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

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

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 MQSC commands to fail. Also, applications cannot connect to a queue manager that is not defined in the WebSphere MQ configuration file.

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

On Linux (x86 platform) systems you can edit configuration information from the WebSphere MQ Explorer.

Editing configuration files

Before editing 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:

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

When do you need to edit a configuration file?

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

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

Configuration file priorities

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

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

The 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.

The mqs.ini file for WebSphere MQ for UNIX systems is in the /var/mqm directory. It contains:

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

Figure 9 on page 106 shows an example of a WebSphere MQ configuration file:

WebSphere MQ configuration file

```
#####  
#* Module Name: mqs.ini                                     *#  
#* Type       : WebSphere MQ Machine-wide Configuration File *#  
#* Function   : Define WebSphere MQ resources for an entire machine *#  
#####  
#* Notes      :                                           *#  
#* 1) This is the installation time default configuration *#  
#*                                                    *#  
#####  
AllQueueManagers:  
#####  
#* The path to the qmgrs directory, below which queue manager data *#  
#* is stored                                                         *#  
#####  
DefaultPrefix=/var/mqm  
  
LogDefaults:  
  LogPrimaryFiles=3  
  LogSecondaryFiles=2  
  LogFilePages=1024  
  LogType=CIRCULAR  
  LogBufferPages=0  
  LogDefaultPath=/var/mqm/log  
  
QueueManager:  
  Name=saturn.queue.manager  
  Prefix=/var/mqm  
  Directory=saturn!queue!manager  
  
QueueManager:  
  Name=pluto.queue.manager  
  Prefix=/var/mqm  
  Directory=pluto!queue!manager  
  
DefaultQueueManager:  
  Name=saturn.queue.manager  
  
ApiExitTemplate:  
  Name=OurPayrollQueueAuditor  
  Sequence=2  
  Function=EntryPoint  
  Module=/usr/ABC/auditor  
  Data=123  
  
ApiExitCommon:  
  Name=MQPoliceman  
  Sequence=1  
  Function=EntryPoint  
  Module=/usr/MQPolice/tmqp  
  Data=CheckEverything
```

Figure 9. Example of a WebSphere MQ configuration file for UNIX systems

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, the path and the name for a configuration file for a queue manager called QMNAME is:

```
/var/mqm/qmgrs/QMNAME/qm.ini
```

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 20.

Figure 10 on page 108 shows how groups of attributes might be arranged in a queue manager configuration file in WebSphere MQ for UNIX systems.

Queue manager configuration file

```
## 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=/var/mqm/exits
  ExitsDefaultPath64=/var/mqm/exits64

Service:
  Name=AuthorizationService
  EntryPoints=13

ServiceComponent:
  Service=AuthorizationService
  Name=MQSeries.UNIX.auth.service
  Module=/opt/mqm/bin/amqzfu 1
  ComponentDataSize=0

Log:
  LogPrimaryFiles=3
  LogSecondaryFiles=2
  LogFilePages=1024
  LogType=CIRCULAR
  LogBufferPages=0
  LogPath=/var/mqm/log/saturn!queue!manager/

XAResourceManager:
  Name=DB2 Resource Manager Bank
  SwitchFile=/usr/bin/db2swit
  XAOpenString=MQBankDB
  XACloseString=
  ThreadOfControl=THREAD

Channels: 2
  MaxChannels=20
  MaxActiveChannels=100
  MQIBindType=STANDARD

TCP:
  KeepAlive = Yes

QMErrorLog:
  ErrorLogSize=262144
  ExcludeMessage=7234
  SuppressMessage=9001,9002,9202
  SuppressInterval=30

ApiExitLocal:
  Name=ClientApplicationAPIchecker
  Sequence=3
  Function=EntryPoint
  Module=/usr/Dev/ClientAppChecker
  Data=9.20.176.20
```

Figure 10. Example queue manager configuration file for WebSphere MQ for UNIX systems

Notes for Figure 10:

1. /usr/mqm/bin/amqzfu on AIX

- For more information on the Channel stanza, see the *WebSphere MQ Intercommunication* manual.

Attributes for changing WebSphere MQ configuration information

The attributes described here modify the configuration of WebSphere MQ. On WebSphere MQ for Windows systems and on WebSphere MQ for Linux (x86 platform) systems, modify configuration information using the WebSphere MQ Explorer. On other systems, modify the information by editing the `mqs.ini` configuration file.

The following are detailed:

- “All queue managers”
- “Client exit path” on page 110
- “Default queue manager” on page 110
- “Exit properties” on page 111
- “Log defaults for WebSphere MQ” on page 111
- “Advanced Configuration and Power Interface (ACPI)” on page 114
- “API exits” on page 115
- “Queue managers” on page 115

All queue managers

Use the General and Extended WebSphere MQ properties page from the WebSphere MQ Explorer, or the `AllQueueManagers` stanza in the `mqs.ini` file to specify the following information about all queue managers.

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 36 on page 556).

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.

Note: Do not delete the `/var/mqm/errors` directory on UNIX systems, or the `\errors` directory on Windows systems.

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

ConvEBCDICNewline=`NL_TO_LF|TABLE|ISO`

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

Queue manager configuration file

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 3.

Table 3. 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*.

Client exit path

Use the Exits WebSphere MQ properties page from the WebSphere MQ Explorer, or the ClientExitPath stanza in the mqs.ini file to specify the default path for location of channel exits 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.

Default queue manager

Use the General WebSphere MQ properties page from the WebSphere MQ Explorer, or the DefaultQueueManager stanza in the mqs.ini file to specify the default path for location of channel exits on the client.

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.

Exit properties

Use the Exits WebSphere MQ properties page from the WebSphere MQ Explorer, or the ExitProperties stanza in the mq.ini file to specify 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 will fail and you run the risk of the integrity of the queue manager being compromised.

Log defaults for WebSphere MQ

Use the Default log settings WebSphere MQ properties page from the WebSphere MQ Explorer, or the LogDefaults stanza in the mq.ini file to specify information about log defaults for all queue managers. 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 299 for details of this command.

Once a queue manager has been created, the log attributes for that queue manager are taken from the settings described in “Queue manager logs” on page 118.

The default prefix (specified in “All queue managers” on page 109) and log path specified for the particular queue manager (specified in “Queue manager logs” on page 118) allow 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.

Queue manager configuration file

For information about calculating log sizes, see “Calculating the size of the log” on page 228.

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

LogPrimaryFiles=3 | 2-254 (Windows) | 2-510 (UNIX systems)

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.

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.

LogSecondaryFiles=2 | 1-253 (Windows) | 1-509 (UNIX systems)

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.

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.

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 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.

LogType=CIRCULAR | LINEAR

The type of log to be used. The default is CIRCULAR.

CIRCULAR

Start restart recovery using the log to roll back transactions that were in progress when the system stopped.

See “Circular logging” on page 224 for a fuller explanation of circular logging.

LINEAR

For 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 225 for a fuller explanation of linear logging.

If you want to change the default, you can either edit the `LogType` attribute, or specify linear logging using the `crtmqm` command. You cannot change the logging method after a queue manager has been created.

LogBufferPages=0|0-4096

The amount of memory allocated to buffer records for writing, specifying the size of the buffers in units of 4 KB pages.

The minimum number of buffer pages is 18 and the maximum is 4096. Larger buffers lead to higher throughput, especially for larger messages.

If you specify 0 (the default), the queue manager selects the size. In WebSphere MQ Version 6.0 this is 128 (512 KB).

If you specify a number between 1 and 17, the queue manager defaults to 18 (72 KB). If you specify a number between 18 and 4096, the queue manager uses the number specified to set the memory allocated.

The value is examined when the queue manager is created, and might 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

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

The default is:

- `<DefaultPrefix>\log` for WebSphere MQ for Windows where `<DefaultPrefix>` is the value specified on the `DefaultPrefix` attribute on the All Queue Managers WebSphere MQ properties page. This value is set at install time.
- `/var/mqm/log` for WebSphere MQ for UNIX systems

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

If you do not specify `-ld` on the `crtmqm` command, the value of the `LogDefaultPath` attribute in the `mqs.ini` file is used.

The queue manager name is appended to the directory name to ensure that multiple queue managers use different log directories.

When the queue manager is created, a `LogPath` value is created in the log attributes in the configuration information, giving the complete directory name for the queue manager's log. This value is used to locate the log when the queue manager is started or deleted.

LogWriteIntegrity=SingleWrite | DoubleWrite | TripleWrite

The method the logger uses to reliably write log records.

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:

Queue manager configuration file

- 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

This 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.

Advanced Configuration and Power Interface (ACPI)

Windows 2000 supports the Advanced Configuration and Power Interface (ACPI) standard. This enables Windows users with ACPI enabled hardware to stop and restart channels when the system enters and resumes from suspend mode. Use the ACPI WebSphere MQ properties page from the WebSphere MQ Explorer, to specify how WebSphere MQ is to behave when the system receives a suspend request.

DoDialog=Y | N

Displays the dialog at the time of a suspend request.

DenySuspend=Y | N

Denies the suspend request. This is used if DoDialog=N, or if DoDialog=Y and a dialog cannot be displayed, for example, because your laptop lid is closed.

CheckChannelsRunning=Y | N

Checks whether any channels are running. The outcome can determine the outcome of the other settings.

The following table outlines the effect of each combination of these parameters:

DoDialog	DenySuspend	CheckChannels Running	Action
N	N	N	Accept the suspend request.
N	N	Y	Accept the suspend request.
N	Y	N	Deny the suspend request.
N	Y	Y	If any channels are running deny the suspend request; if not accept the request.
Y	N	N	Display the dialog (see note below; accept the suspend request). This is the default.
Y	N	Y	If no channels are running accept the suspend request; if they are display the dialog (see note below; accept the request).
Y	Y	N	Display the dialog (see note below; deny the suspend request).
Y	Y	Y	If no channels are running accept the suspend request; if they are display the dialog (see note below; deny the request).

Note: In cases where the action is to display the dialog, if the dialog cannot be displayed (for example because your laptop lid is closed), the DenySuspend option is used to determine whether the suspend request is accepted or denied.

API exits

Use the Exits WebSphere MQ properties page from the WebSphere MQ Explorer, or the ApiExitTemplate and ApiExitCommon stanza in the mq.ini file to identify API exit routines for all queue managers. On Windows systems, you can also use the **amqmdain** command to change the Registry entries for API exits. (To identify API exit routines for individual queue managers, you use the ApiExitLocal stanza, as described in “API exits” on page 127.)

For a complete description of the attributes for these stanzas, see “Configuring API exits” on page 503.

Queue managers

On UNIX systems, 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 20 for more information about name transformation.)

On Windows systems, this information is held in the Registry. You cannot use the WebSphere MQ Explorer to change it directly.

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 attributes described here modify the configuration of an individual queue manager. They override any settings for WebSphere MQ. On WebSphere MQ for Windows systems and on WebSphere MQ for Linux (x86 platform) systems, modify configuration information using the WebSphere MQ Explorer. On other systems, modify the information by editing the qm.ini configuration file.

The following are detailed:

- “Installable services” on page 116
- “Service components” on page 117
- “Queue manager logs” on page 118
- “Restricted mode” on page 120
- “XA resource managers” on page 121

Queue manager configuration file

- “Channels” on page 122
- “LU62, NETBIOS, TCP, and SPX” on page 124
- “Exit path” on page 126
- “API exits” on page 127
- “Queue manager error logs” on page 127
- “Queue manager default bind type” on page 128

Installable services

There are significant implications to changing installable services and their components. For this reason, the installable services are read-only in the WebSphere MQ Explorer. To change installable services in on Windows systems, use **regedit** or on UNIX systems use the Service stanza in the qm.ini file.

For each component within a service, you must also specify the name and path of the module containing the code for that component. On UNIX systems, use the ServiceComponent stanza for this.

Name=AuthorizationService | NameService

The name of the required service.

AuthorizationService

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

The AuthorizationService stanza and its associated ServiceComponent stanza are added automatically when the queue manager is created. Add other ServiceComponent stanzas manually.

NameService

No name service is provided by default. If you require a name service, you must add the NameService stanza manually.

EntryPoints=number-of-entries

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

SecurityPolicy=Default | NTSIDsRequired (WebSphere MQ for Windows only)

The SecurityPolicy attribute applies only if the service specified is the default authorization service, that is, the OAM. The SecurityPolicy attribute allows you to specify the security policy for each queue manager. The possible values are:

Default

Use the default security policy to take effect. If a Windows security identifier (NT SID) is not passed to the OAM for a particular user ID, an attempt is made to obtain the appropriate SID by searching the relevant security databases.

NTSIDsRequired

Pass an NT SID to the OAM when performing security checks.

See “Windows security identifiers (SIDs)” on page 133 for more information.

SharedBindingsUserId=user-type

The SharedBindingsUserId attribute applies only if the service specified is the default authorization service, that is, the OAM. The SharedBindingsUserId attribute is used with relation to shared bindings only. This value allows you to specify whether the *UserIdentifier* field in the *IdentityContext* structure, from the MQZ_AUTHENTICATE_USER function, is the effective user Id or the real

user Id. For information on the MQZ_AUTHENTICATE_USER function, see “MQZ_AUTHENTICATE_USER – Authenticate user” on page 423. The possible values are:

Default

The value of the *UserIdentifier* field is set as the real user Id.

Real

The value of the *UserIdentifier* field is set as the real user Id.

Effective

The value of the *UserIdentifier* field is set as the effective user Id.

FastpathBindingsUserId=user-type

The *FastpathBindingsUserId* attribute applies only if the service specified is the default authorization service, that is, the OAM. The *FastpathBindingsUserId* attribute is used with relation to fastpath bindings only. This value allows you to specify whether the *UserIdentifier* field in the *IdentityContext* structure, from the MQZ_AUTHENTICATE_USER function, is the effective user Id or the real user Id. For information on the MQZ_AUTHENTICATE_USER function, see “MQZ_AUTHENTICATE_USER – Authenticate user” on page 423. The possible values are:

Default

The value of the *UserIdentifier* field is set as the real user Id.

Real

The value of the *UserIdentifier* field is set as the real user Id.

Effective

The value of the *UserIdentifier* field is set as the effective user Id.

IsolatedBindingsUserId =user-type

The *IsolatedBindingsUserId* attribute applies only if the service specified is the default authorization service, that is, the OAM. The *IsolatedBindingsUserId* attribute is used with relation to isolated bindings only. This value allows you to specify whether the *UserIdentifier* field in the *IdentityContext* structure, from the MQZ_AUTHENTICATE_USER function, is the effective user Id or the real user Id. For information on the MQZ_AUTHENTICATE_USER function, see “MQZ_AUTHENTICATE_USER – Authenticate user” on page 423. The possible values are:

Default

The value of the *UserIdentifier* field is set as the effective user Id.

Real

The value of the *UserIdentifier* field is set as the real user Id.

Effective

The value of the *UserIdentifier* field is set as the effective user Id.

For more information about installable services and components, see Part 7, “WebSphere MQ installable services and the API exit,” on page 395.

For more information about security services in general, see Chapter 10, “WebSphere MQ security,” on page 129.

Service components

You need to specify service component information when you add a new installable service. On Windows systems use **regedit**, and on UNIX systems use the *ServiceComponent* stanza in the *qm.ini* file.

Queue manager configuration file

The authorization service stanza is present by default, and the associated component, the OAM, is active.

Service=*service_name*

The name of the required service. This must match the value specified on the Name attribute of the Service configuration information.

Name=*component_name*

The descriptive name of the service component. This must be unique and contain only 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. We recommend that this name begins with a company trademark or similar distinguishing string.

Module=*module_name*

The name of the module to contain the code for this component. This must be a full path name.

ComponentDataSize=*size*

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 Part 7, “WebSphere MQ installable services and the API exit,” on page 395.

Queue manager logs

Use the Log queue manager properties page from the WebSphere MQ Explorer, or the Log stanza in the qm.ini file, to specify information about logging on this queue manager.

By default, these settings are inherited from the settings specified for the default log settings for the queue manager (described in “Log defaults for WebSphere MQ” on page 111). Change these settings only if you want to configure this queue manager in a different way.

For information about calculating log sizes, see “Calculating the size of the log” on page 228.

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

LogPrimaryFiles=3|2-254 (Windows)|2-510 (UNIX systems)

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.

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.

LogSecondaryFiles=2|1-253 (Windows)|1-509 (UNIX systems)

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.

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.

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 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.

LogType=CIRCULAR | LINEAR

The type of logging to be used by the queue manager. 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 “Log defaults for WebSphere MQ” on page 111 for information about creating a queue manager with the type of logging you require.

CIRCULAR

Start restart recovery using the log to roll back transactions that were in progress when the system stopped.

See “Circular logging” on page 224 for a fuller explanation of circular logging.

LINEAR

For 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 225 for a fuller explanation of linear logging.

LogBufferPages=0 | *0-4096*

The amount of memory allocated to buffer records for writing, specifying the size of the buffers in units of 4 KB pages.

The minimum number of buffer pages is 18 and the maximum is 4096. Larger buffers lead to higher throughput, especially for larger messages.

If you specify 0 (the default), the queue manager selects the size. In WebSphere MQ Version 6.0 this is 128 (512 KB).

If you specify a number between 1 and 17, the queue manager defaults to 18 (72 KB). If you specify a number between 18 and 4096, the queue manager uses the number specified to set the memory allocated.

The value is examined when the queue manager is created or started, and might 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*

The directory in which the log files for a queue manager reside. This must exist on a local device to which the queue manager can write and, preferably,

Queue manager configuration file

on a different drive from the message queues. Specifying a different drive gives added protection in case of system failure.

The default is:

- C:\Program Files\IBM\WebSphere MQ\log in WebSphere MQ for Windows.
- /var/mqm/log in WebSphere MQ for UNIX systems.

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 is used.

In WebSphere MQ for UNIX systems, user ID **mqm** and group **mqm** must have full authorities to the log files. If you change the locations of these files, you must give these authorities yourself. This is not required if the log files are in the default locations supplied with the product.

LogWriteIntegrity=SingleWrite | DoubleWrite | TripleWrite

The method the logger uses to reliably write log records.

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

This 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.

Restricted mode

This option applies to UNIX systems only. The **RestrictedMode** stanza is set by the **-g** option on the **crtmqm** command. Do **not** change this stanza after the queue manager has been created. If you do not use the **-g** option, the stanza is not created in the **qm.ini** file.

ApplicationGroup

The name of the group with members that are allowed to:

- Run MQI applications
- Update all IPCC resources
- Change the contents of some queue manager directories

XA resource managers

Use the XA resource manager queue manager properties page from the WebSphere MQ Explorer, or the XAResourceManager stanza in the qm.ini file, to specify the following information about the resource managers involved in global units of work coordinated by the queue manager.

Add XA resource manager configuration information manually for each instance of a resource manager participating in global units of work; no default values are supplied.

See “Database coordination” on page 166 for more information about resource manager attributes.

Name=*name* (mandatory)

This attribute identifies the resource manager instance.

The Name value can be up to 31 characters in length. 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 can 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 the configuration information, once the associated queue manager has started and the resource manager name is in effect.

SwitchFile=*name* (mandatory)

The fully-qualified name of the load file containing the resource manager’s XA switch structure.

XAOpenString=*string* (optional)

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 resource manager configuration information for DB2” on page 174
- “Adding resource manager configuration information for Oracle” on page 177
- “Adding resource manager configuration information for Sybase” on page 181
- “Adding resource manager configuration information for Informix” on page 179

and consult your resource manager documentation for the appropriate string.

XACloseString=*string* (optional)

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 resource manager configuration information for DB2” on page 174
- “Adding resource manager configuration information for Oracle” on page 177
- “Adding resource manager configuration information for Sybase” on page 181

Queue manager configuration file

- “Adding resource manager configuration information for Informix” on page 179

and consult your database documentation for the appropriate string.

ThreadOfControl=THREAD | PROCESS

This attribute is mandatory for WebSphere MQ for Windows. The queue manager uses this value for serialization when it needs to call the resource manager from one of its own multithreaded processes.

THREAD

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

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.

Channels

Use the Channels queue manager properties page from the WebSphere MQ Explorer, or the CHANNELS stanza in the qm.ini file, to specify information about channels.

MaxChannels=100 | number

The maximum number of channels allowed. The default is 100.

MaxActiveChannels=MaxChannels_value

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

The maximum number of initiators.

MQIBindType=FASTPATH | SHARED

The binding for applications:

FASTPATH

Channels connect using MQCONN FASTPATH; there is no agent process.

SHARED

Channels connect using SHARED.

PipeLineLength=1 | number

The maximum number of concurrent threads a channel will use. The default is 1. Any value greater than 1 is treated as 2.

When you use pipelining, configure the queue managers at both ends of the channel to have a *PipeLineLength* greater than 1.

Note: Pipelining is only effective for TCP/IP channels.

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 end 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 requesting it to end.
2. If the previous channel server does not respond to this request by the time the `AdoptNewMCATimeout` wait interval expires, the process (or the thread) for the previous channel server is ended.
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.

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 `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 might behave in an unpredictable fashion with `FASTPATH` channels. Exercise great caution when enabling the `AdoptNewMCA` attribute for `FASTPATH` channels.

AdoptNewMCATimeout=60 | 1 – 3600

The amount of time, in seconds, that the new process waits for the old process to end. Specify a value in the range 1 – 3600. The default value is 60.

AdoptNewMCACheck=QM | ADDRESS | NAME | ALL

The type of checking required when enabling the `AdoptNewMCA` attribute. If possible, perform all three of the following checks to protect your channels from being shut down, inadvertently or maliciously. At the very least, check that the channel names match.

Specify one or more values, separated by commas or blanks, to tell the listener process to:

Queue manager configuration file

QM

Check that the queue manager names match.

ADDRESS

Check the communications address. For example, the TCP/IP address.

NAME

Check that the channel names match.

ALL

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.

LU62, NETBIOS, TCP, and SPX

Use the following queue manager properties pages, or stanzas in the `qm.ini` file, to specify network protocol configuration parameters. They override the default attributes for channels.

LU62 (WebSphere MQ for Windows only)

Use the LU6.2 queue manager properties page from the WebSphere MQ Explorer, or the LU62 stanza in the `qm.ini` file, to specify SNA LU 6.2 protocol configuration parameters.

TPName

The TP name to start on the remote site.

Library1=*DLLName 1*

The name of the APPC DLL.

The default value is WCPIC32.

Library2=*DLLName2*

The same as Library1, used if the code is stored in two separate libraries.

The default value is WCPIC32.

NETBIOS (WebSphere MQ for Windows only)

Use the Netbios queue manager properties page from the WebSphere MQ Explorer, or the NETBIOS stanza in the `qm.ini` file, to specify NetBIOS protocol configuration parameters.

LocalName=*name*

The name by which this machine is known on the LAN.

AdapterNum=*0 | adapter_number*

The number of the LAN adapter. The default is adapter 0.

NumSess=*1 | number_of_sessions*

The number of sessions to allocate. The default is 1.

NumCmds=*1 | number_of_commands*

The number of commands to allocate. The default is 1.

NumNames=*1 | number_of_names*

The number of names to allocate. The default is 1.

Library1=*DLLName1*

The name of the NetBIOS DLL.

The default value is NETAPI32.

TCP

Use the TCP queue manager properties page from the WebSphere MQ Explorer, or the TCP stanza in the qm.ini file, to specify Transmission Control Protocol / Internet Protocol (TCP/IP) configuration parameters.

Port=1414 | *port_number*

The default port number, in decimal notation, for TCP/IP sessions. The *well known* port number for WebSphere MQ is 1414.

Library1=DLLName1 (WebSphere MQ for Windows only)

The name of the TCP/IP sockets DLL.

The default is WSOCK32.

KeepAlive=YES|NO

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

Override the default number of outstanding requests for the TCP/IP listener.

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 4.

Table 4. Default outstanding connection requests (TCP)

Platform	Default ListenerBacklog value
Windows Server	100
Windows Workstation	5
Linux	100
Solaris	100
HP-UX	20
AIX V4.2 or later	100
AIX V4.1 or earlier	10

Note: Some operating systems support a larger value than the default shown. Use this to avoid reaching the connection limit.

If the backlog reaches the values shown in Table 4, the TCP/IP connection is rejected and the channel cannot start. For message 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 retries the connection at a later time.

SPX (WebSphere MQ for Windows only)

Use the SPX queue manager properties page from the WebSphere MQ Explorer, or the SPX stanza in the qm.ini file, to specify SPX protocol configuration parameters.

Socket=5E86 | *socket_number*

The SPX socket number in hexadecimal notation. The default is X'5E86'.

BoardNum=0 | *adapter_number*

The LAN adapter number. The default is adapter 0.

Queue manager configuration file

KeepAlive=YES|NO

Switch the KeepAlive function on or off.

KeepAlive=YES causes SPX to check periodically that the other end of the connection is still available. If it is not, the channel is closed.

Library1=DLLName1

The name of the SPX DLL.

The default is WSOCK32.DLL.

Library2=DLLName2

The same as LibraryName1, used if the code is stored in two separate libraries.

The default is WSOCK32.DLL.

ListenerBacklog=number

Override the default number of outstanding requests for the SPX listener.

When receiving on SPX, a maximum number of outstanding connection requests is set. This can be considered to be a *backlog* of requests waiting on the SPX socket for the listener to accept the request. The default listener backlog values are shown in Table 5.

Table 5. Default outstanding connection requests (SPX)

Platform	Default ListenerBacklog value
Windows Server	100
Windows Workstation	5

Note: Some operating systems support a larger value than the default shown. Use this to avoid reaching the connection limit.

If the backlog reaches the values shown in Table 5, the SPX connection is rejected and the channel cannot start. For message 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.

Exit path

Use the Exits queue manager properties page from the WebSphere MQ Explorer, or the ExitPath stanza in the qm.ini file to specify the path for user exit programs.

ExitDefaultPath=string

The ExitDefaultPath attribute specifies the location of:

- 32-bit channel exits for clients
- 32-bit 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

The exit path for clients is held in the WebSphere MQ configuration information (as described in “Client exit path” on page 110).

API exits

Use the Exits queue manager properties page from the WebSphere MQ Explorer, or the `ApiExitLocal` stanza in the `qm.ini` file to identify API exit routines for a queue manager. On Windows systems, you can also use the `amqmdain` command to change the Registry entries for API exits. (To identify API exit routines for all queue managers, you use the `ApiExitCommon` and `ApiExitTemplate` stanzas, as described in “API exits” on page 115.)

For a complete description of the attributes for these stanzas, see “Configuring API exits” on page 503.

Queue manager error logs

Use the Extended queue manager properties page from the WebSphere MQ Explorer, or 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 1048576 and 2147483648 bytes. If **ErrorLogSize** is not specified, the default value of 262144 bytes (256KB) is used.

ExcludeMessages=*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
- 9524 - Remote queue manager unavailable
- 9528 - User requested closure of channel
- 9999 - Channel program ended abnormally

SuppressMessages=*msgIds*

Specifies messages that will be 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
- 9524 - Remote queue manager unavailable
- 9528 - User requested closure of channel
- 9999 - Channel program ended abnormally

If the same message id is specified in both **SuppressMessages** and **ExcludeMessages**, the message is excluded.

SuppressInterval=*length*

Specifies the time interval, in seconds, in which messages specified in

Queue manager configuration file

SuppressMessages will be 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.

Queue manager default bind type

Use the Extended queue manager properties page from the WebSphere MQ Explorer, or the `Connection` stanza in the `qm.ini` file to specify the default bind type.

DefaultBindType=SHARED | ISOLATED

If `DefaultBindType` is set to `ISOLATED`, applications and the queue manager run in separate processes, and no resources are shared between them.

If `DefaultBindType` is set to `SHARED`, applications and the queue manager run in separate processes, however some resources are shared between them.

The default is `SHARED`.

Chapter 10. WebSphere MQ security

WebSphere MQ queue managers transfer information that is potentially valuable, so you need to use an authority system to ensure that unauthorized users cannot access your queue managers. Consider the following types of security controls:

Who can administer WebSphere MQ

You can define the set of users who can issue commands to administer WebSphere MQ.

Who can use WebSphere MQ objects

You can define which users (usually applications) can use MQI calls and PCF commands to do the following:

- Who can connect to a queue manager.
- Who can access objects (queues, process definitions, namelists, channels, client connection channels, listeners, services, and authentication information objects), and what type of access they have to those objects.
- Who can access WebSphere MQ messages.
- Who can access the context information associated with a message.

Channel security

You need to ensure that channels used to send messages to remote systems can access the required resources.

You can use standard operating facilities to grant access to program libraries, MQI link libraries, and commands. However, the directory containing queues and other queue manager data is private to WebSphere MQ; do not use standard operating system commands to grant or revoke authorizations to MQI resources.

Authority to administer WebSphere MQ

WebSphere MQ administrators have authority to use all WebSphere MQ commands (including the commands to grant WebSphere MQ authorities for other users)

To be a WebSphere MQ administrator, you must be a member of a special group called the *mqm* group (or a member of the Administrators group on Windows systems; see below). The *mqm* group is created automatically when WebSphere MQ is installed; add further users to the group to allow them to perform administration (including the root user on UNIX systems). All members of this group have access to all resources. This access can be revoked only by removing a user from the *mqm* group.

On UNIX platforms, a special user ID of *mqm* is also created, for use by the product only. It must never be available to non-privileged users. All WebSphere MQ objects are owned by user ID *mqm*.

On Windows systems, members of the Administrators group can also administer any queue manager. You can also create a domain *mqm* group on the domain controller that contains all privileged user IDs active within the domain, and add it to the local *mqm* group. Some commands, for example **crtmqm**, manipulate authorities on WebSphere MQ objects and so need authority to work with these objects (as described below). Members of the *mqm* group have authority to work

Administration authority

with all objects, but there might be circumstances on Windows systems when authority is denied if you have a local user and a domain-authenticated user with the same name. This is described in “Principals and groups” on page 132.

You do not need to be a member of the mqm group to do the following:

- Issue commands from an application program that issues PCF commands, or MQSC commands within an Escape PCF command, unless the commands manipulate channel initiators. (These commands are described in “Protecting channel initiator definitions” on page 147).
- Issue MQI calls from an application program (unless you want to use the fastpath bindings on the **MQCONN** call).
- Use the **crtmqcvx** command to create a fragment of code that performs data conversion on data type structures.
- Use the **dspmq** command to display queue managers.
- Use the **dspmqtrc** command to display WebSphere MQ formatted trace output.

Managing the mqm group

Security administrators add users who need to administer WebSphere MQ to the mqm group. This includes the root user on UNIX systems. They might also need to remove users who no longer need this authority. These tasks are described in “Creating and managing groups” on page 136.

If your domain controller runs on Windows 2000, your domain administrator might have to set up a special account for WebSphere MQ to use. This is described in the *WebSphere MQ for Windows, V6.0 Quick Beginnings*.

Authority to work with WebSphere MQ objects

Queue managers, queues, process definitions, namelists, channels, client connection channels, listeners, services and authentication information objects are all accessed from applications that use MQI calls or PCF commands. These resources are all protected by WebSphere MQ, and applications need to be given permission to access them. The entity making the request might be a user, an application program that issues an MQI call, or an administration program that issues a PCF command. The identifier of the requester is referred to as the *principal*.

Different groups of principals can be granted different types of access authority to the same object. For example, for a specific queue, one group might be allowed to perform both put and get operations; another group might be allowed only to browse the queue (**MQGET** with browse option). Similarly, some groups might have put and get authority to a queue, but not be allowed to alter attributes of the queue or delete it.

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

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

Full access permission to an object is automatically given to the user ID that created the object and to all members of the mqm group (and to the members of the local Administrators group on Windows systems).

When security checks are made

The security checks made for a typical application are as follows:

Connecting to the queue manager (MQCONN or MQCONNX calls)

This is the first time that the application is associated with a particular queue manager. The queue manager interrogates the operating environment to discover the user ID associated with the application. WebSphere MQ then verifies that the user ID is authorized to connect to the queue manager and retains the user ID for future checks.

Users do not have to sign on to WebSphere MQ; WebSphere MQ assumes that users have signed on to the underlying operating system and have been authenticated by that.

Opening the object (MQOPEN or MQPUT1 calls)

WebSphere MQ objects are accessed by opening the object and issuing commands against it. All resource checks are performed when the object is opened, rather than when it is actually accessed. This means that the **MQOPEN** request must specify the type of access required (for example, whether the user wants only to browse the object or perform an update like putting messages onto a queue).

WebSphere MQ checks the resource that is named in the **MQOPEN** request. For an alias or remote queue object, the authorization used is that of the object itself, not the queue to which the alias or remote queue resolves. This means that the user does not need permission to access it. Limit the authority to create queues to privileged users. If you do not, users might bypass the normal access control simply by creating an alias. If a remote queue is referred to explicitly with both the queue and queue manager names, the transmission queue associated with the remote queue manager is checked.

The authority to a dynamic queue is based on that of the model queue from which it is derived, but is not necessarily the same. This is described in Note 1 on page 152.

The user ID used by the queue manager for access checks is the user ID obtained from the operating environment of the application connected to the queue manager. A suitably authorized application can issue an **MQOPEN** call specifying an alternative user ID; access control checks are then made on the alternative user ID. This does not change the user ID associated with the application, only that used for access control checks.

Putting and getting messages (MQPUT or MQGET calls)

No access control checks are performed.

Closing the object (MQCLOSE)

No access control checks are performed, unless the **MQCLOSE** will result in a dynamic queue being deleted. In this case, there is a check that the user ID is authorized to delete the queue.

How access control is implemented by WebSphere MQ

WebSphere MQ uses the security services provided by the underlying operating system. An access control interface called the Authorization Service Interface is part of WebSphere MQ. WebSphere MQ supplies an implementation of an access control manager (conforming to the Authorization Service Interface) known as the *Object Authority Manager (OAM)*. This is automatically installed and enabled for each queue manager you create, unless you specify otherwise (as described in

Administration authority

“Preventing security access checks” on page 146). The OAM can be replaced by any user or vendor written component that conforms to the Authorization Service Interface.

The OAM exploits the security features of the underlying operating system, using operating system user and group IDs. Users can access WebSphere MQ objects only if they have the correct authority. “Using the OAM to control access to objects” on page 141 describes how to grant and revoke this authority.

The OAM maintains an access control list (ACL) for each resource that it controls. Authorization data is stored on a local queue called `SYSTEM.AUTH.DATA.QUEUE`. Access to this queue is restricted to users in the `mqm` group, and additionally on Windows, to users in the Administrators group, and users logged in with the `SYSTEM` ID. User access to the queue cannot be changed.

WebSphere MQ supplies commands to create, and maintain access control lists. For more information on these commands, see “Using the OAM to control access to objects” on page 141.

WebSphere MQ passes the OAM a request containing a principal, a resource name, and an access type. The OAM grants or rejects access based on the ACL that it maintains. WebSphere MQ follows the decision of the OAM; if the OAM cannot make a decision, WebSphere MQ does not allow access.

Identifying the user ID

The OAM needs to be able to identify who is requesting access to a particular resource. WebSphere MQ uses the term *principal* to refer to this identifier. The principal is established when the application first connects to the queue manager; it is determined by the queue manager from the user ID associated with the connecting application. (If the application is running in a transaction monitor environment, the connection is through the X/Open XA interface, which does not provide a mechanism for passing user IDs, so WebSphere MQ assumes a default user name.)

On UNIX systems, the authorization routines checks either the real (logged-in) user ID, or the effective user ID associated with the application. The user ID checked can be dependent on the bind type, for details see “Installable services” on page 116.

WebSphere MQ propagates the user ID received from the system in the message header (MQMD structure) of each message as identification of the user. This identifier is part of the message context information and is described in “Context authority” on page 134. Applications cannot alter this information unless they have been authorized to change context information.

Principals and groups

Principals can belong to groups. You can grant access to a particular resource to groups rather than to individuals, to reduce the amount of administration required. For example, you might define a group consisting of users who want to run a particular application. Other users can be given access to all the resources they require simply by adding their user ID to the appropriate group. This is described in “Creating and managing groups” on page 136.

A principal can belong to more than one group (its group set) and has the aggregate of all the authorities granted to each group in its group set. These

authorities are cached, so any changes you make to the principal's group membership are not recognized until the queue manager is restarted, unless you issue the MQSC command REFRESH SECURITY (or the PCF equivalent).

UNIX systems

All ACLs are based on groups. When a user is granted access to a particular resource, the user ID's primary group is included in the ACL, not the individual user ID, and authority is granted to all members of that group. Because of this, be aware that you could inadvertently change the authority of a principal by changing the authority of another principal in the same group.

Note: To add a user to an ACL or any group, WebSphere MQ on UNIX systems requires the user ID to have a maximum length of eight characters.

All users are nominally assigned to the default user group *nobody* and by default, no authorizations are given to this group. You can change the authorization in the *nobody* group to grant access to WebSphere MQ resources to users without specific authorizations.

Windows systems

ACLs are based on both user IDs and groups. Checks are the same as for UNIX systems except that individual user IDs can appear in the ACL as well. You can have different users on different domains with the same user ID; WebSphere MQ allows user IDs to be qualified by a domain name so that these users can be given different levels of access. Group names always refer to local groups, so you don't need to qualify them with a domain name.

User IDs can contain up to 20 characters, domain names up to 15 characters, and group names up to 64 characters.

The OAM first checks the local security database, then the database of the primary domain, and finally the database of any trusted domains. The first user ID encountered is used by the OAM for checking. Each of these user IDs might have different group memberships on a particular computer.

Some control commands (for example, **crtmqm**) change authorities on WebSphere MQ objects using the Object Authority Manager (OAM). Because the OAM searches the security databases in the order given above to determine the authority rights for a given user ID, the authority determined by the OAM might override the fact that a user ID is a member of the local mqm group. For example, if you issue **crtmqm** from a user ID authenticated by a domain controller that has membership of the local mqm group through a global group, the command fails if the system has a local user of the same name who is not in the local mqm group.

Windows security identifiers (SIDs)

On Windows systems, the security identifier (SID) is used to supplement the user ID. The SID contains information that identifies the full user account details on the Windows security account manager (SAM) database where the user is defined. When a message is created on WebSphere MQ for Windows, WebSphere MQ stores the SID in the message descriptor. When WebSphere MQ for Windows performs authorization checks, it uses the SID to query the full information from the SAM database. (The SAM database in which the user is defined must be accessible for this query to succeed.)

Administration authority

By default, if a Windows SID is not supplied with an authorization request, WebSphere MQ identifies the user based on the user name alone. It does this by searching the security databases in the following order:

1. The local security database
2. The security database of the primary domain
3. The security database of trusted domains

If the user name is not unique, incorrect WebSphere MQ authority might be granted. To prevent this problem, include an SID in each authorization request; the SID is used by WebSphere MQ to establish user credentials.

To specify that all authorization requests must include an SID, use **regedit**. Set the SecurityPolicy to NTSIDsRequired.

Alternate-user authority

You can specify that a user ID can use the authority of another user when accessing a WebSphere MQ object. This is called *alternate-user authority*, and you can use it on any WebSphere MQ object.

Alternate-user authority is essential where a server receives requests from a program and wants to ensure that the program has the required authority for the request. The server might have the required authority, but it needs to know whether the program has the authority for the actions it has requested.

For example, assume that 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 a different 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.

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

Who the message came from. It consists of the UserIdentifier, AccountingToken, and ApplIdentityData fields.

Origin section

Where the message came from, and when it was put onto the queue. It consists of the PutApplType, PutApplName, PutDate, PutTime, and ApplOriginData fields.

Applications can specify the context data when either an **MQOPEN** or **MQPUT** call is made. This data might be generated by the application, passed on from another message, or 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.

See the *WebSphere MQ Application Programming Guide* for information about the context options, and the *WebSphere MQ Application Programming Reference* for descriptions of the message descriptor fields relating to context.

Connecting to WebSphere MQ using Terminal Services

If you are connecting using Terminal Services to a machine running one of:

- Windows 2000 with service pack 4 or later
- Windows XP with service pack 2 or later
- Windows 2003

and you have problems creating or starting a queue manager this might be because of the introduction of a new user right, “Create global objects”, in these operating systems.

The “Create global objects” user right limits the users authorized to create objects in the global namespace. In order for an application to create a global object, it must either be running in the global namespace, or the user under which the application is running must have the “Create global objects” user right applied to it.

When you connect using Terminal Services, applications run in their own local namespace. If you attempt to create a queue manager using the WebSphere MQ Explorer or the `crtmqm` control command, or attempt to start a queue manager using the `strmqm` control command, it will result in an authorization failure. This will create a WebSphere MQ FDC with Probe ID XY132002.

Starting a queue manager using the WebSphere MQ Explorer, or using the `amqmdain` control command, will work correctly because these commands do not directly start the queue manager. Instead the commands send the request to start the queue manager to a separate process running in the global namespace.

If you need to create or start a queue manager when connected via Terminal Services you must have the “Create global objects” user right. To get this right your administrator must do the following:

1. Open the control panel.
2. Double-click **Administrative Tools**. The Administrative Tools panel opens.
3. Double-click **Local Security Policy**.
4. Expand **Local Policies**.
5. Click **User Rights Assignment**.
6. Add the new user or group to the “Create global objects” policy.

Administrators have the “Create global objects” user right applied by default, so if you are an administrator you will be able to create and start queue managers when connected via Terminal Services without altering your user rights.

Creating and managing groups

This section tells you how to create groups and add users to them. It also describes how to remove a user from a group. Any changes you make to a principal's group membership are not recognized until the queue manager is restarted, unless you issue the MQSC command REFRESH SECURITY (or the PCF equivalent).

Windows 2000

The following instructions lead you through the process of administering groups on a workstation or member server machine. For domain controllers, users and groups are administered through Active Directory. For more details on using Active Directory refer to the appropriate operating system instructions.

On Windows 2000 use the Computer management panel to work with user and groups.

Creating a group and adding users

1. Open the control panel.
2. Double-click **Administrative Tools**. The Administrative Tools panel opens.
3. Double-click **Computer Management**. The Computer Management panel opens.
4. Expand **Local Users and Groups**.
5. Right-click **Groups**, and select **New Group...** The New Group panel is displayed.
6. Type an appropriate name in the Group name field, then click **Add...** The Select Users or Groups panel is displayed.
7. Select a user, then click **Add...** To add more users to the group, repeat this step.
8. Click **OK**. The New Group panel is displayed.
9. Click **Create**.

You have now created a group, and added users to it.

Displaying who is in a group

1. From the Computer Management panel, expand **Local Users and Groups**.
2. Select **Groups**.
3. Double-click a group. The group properties panel is displayed.

The group members are displayed.

Removing a user from a group

1. From the Computer Management panel, expand **Local Users and Groups**.
2. Select **Groups**.
3. Double-click a group. The group properties panel is displayed, showing the group members.
4. Select the user you want to remove, then click **Remove**. To remove more users from the group, repeat this step.
5. Click **OK**. The New Group panel is displayed.

You have now removed users from a group.

Windows XP and Windows 2003

The following instructions lead you through the process of administering groups on a workstation or member server machine. For domain controllers, users and

groups are administered through Active Directory. For more details on using Active Directory refer to the appropriate operating system instructions.

On Windows XP and Windows 2003 use the Computer management panel to work with user and groups.

Creating a group

1. Open the control panel.
2. Double-click **Administrative Tools**. The Administrative Tools panel opens.
3. Double-click **Computer Management**. The Computer Management panel opens.
4. Expand **Local Users and Groups**.
5. Right-click **Groups**, and select **New Group....** The New Group panel is displayed.
6. Type an appropriate name in the Group name field, then click **Create**.
7. Click **Close**.

You have now created a group.

Adding a user to a group

1. From the Computer Management panel, expand **Local Users and Groups**.
2. Select **Users**.
3. Double-click the user that you want to add to a group. The user properties panel is displayed.
4. Select the **Member Of** tab.
5. Select the group that you want to add the user to. If the group you want is not visible:
 - a. Click **Add....** The Select Groups panel is displayed.
 - b. Type the group name in the field provided.
 - c. Click **OK**. The user properties panel is displayed, showing the group you added.
 - d. Select the group.
6. Click **OK**. The Computer Management panel is displayed.

You have now added a user to a group.

Displaying who is in a group

1. From the Computer Management panel, expand **Local Users and Groups**.
2. Select **Groups**.
3. Double-click a group. The group properties panel is displayed.

The group members are displayed.

Removing a user from a group

1. From the Computer Management panel, expand **Local Users and Groups**.
2. Select **Users**.
3. Double-click the user that you want to add to a group. The user properties panel is displayed.
4. Select the **Member Of** tab.
5. Select the group that you want to remove the user from, then click **Remove**.
6. Click **OK**. The Computer Management panel is displayed.

You have now removed a user from a group.

HP-UX

On HP-UX, providing you are not using NIS or NIS+, use the System Administration Manager (SAM) to work with groups.

Creating a group

1. From the System Administration Manager (SAM), double click Accounts for Users and Groups.
2. Double click Groups.
3. Select Add from the Actions pull down to display the Add a New Group panel.
4. Enter the name of the group and select the users that you want to add to the group.
5. Click Apply to create the group.

You have now created a group.

Adding a user to a group

1. From the System Administration Manager (SAM), double click Accounts for Users and Groups.
2. Double click Groups.
3. Highlight the name of the group and select Modify from the Actions pull down to display the Modify an Existing Group panel.
4. Select a user that you want to add to the group and click Add.
5. If you want to add other users to the group, repeat step 4 for each user.
6. When you have finished adding names to the list, click OK.

You have now added a user to a group.

Displaying who is in a group

1. From the System Administration Manager (SAM), double click Accounts for Users and Groups.
2. Double click Groups.
3. Highlight the name of the group and select Modify from the Actions pull down to display the Modify an Existing Group panel, showing a list of the users in the group.

The group members are displayed.

Removing a user from a group

1. From the System Administration Manager (SAM), double click Accounts for Users and Groups.
2. Double click Groups.
3. Highlight the name of the group and select Modify from the Actions pull down to display the Modify an Existing Group panel.
4. Select a user that you want to remove from the group and click Remove.
5. If you want to remove other users from the group, repeat step 4 for each user.
6. When you have finished removing names from the list, click OK.

You have now removed a user from a group

AIX

On AIX, providing you are not using NIS or NIS+, use SMITTY to work with groups.

Creating a group

1. From SMITTY, select Security and Users and press Enter.
2. Select Groups and press Enter.
3. Select Add a Group and press Enter.
4. Enter the name of the group and the names of any users that you want to add to the group, separated by commas.
5. Press Enter to create the group.

You have now created a group.

Adding a user to a group

1. From SMITTY, select Security and Users and press Enter.
2. Select Groups and press Enter.
3. Select Change / Show Characteristics of Groups and press Enter.
4. Enter the name of the group to show a list of the members of the group.
5. Add the names of the users that you want to add to the group, separated by commas.
6. Press Enter to add the names to the group.

You have now added a user to a group.

Displaying who is in a group

1. From SMITTY, select Security and Users and press Enter.
2. Select Groups and press Enter.
3. Select Change / Show Characteristics of Groups and press Enter.
4. Enter the name of the group to show a list of the members of the group.

The group members are displayed.

Removing a user from a group

1. From SMITTY, select Security and Users and press Enter.
2. Select Groups and press Enter.
3. Select Change / Show Characteristics of Groups and press Enter.
4. Enter the name of the group to show a list of the members of the group.
5. Delete the names of the users that you want to remove from the group.
6. Press Enter to remove the names from the group.

You have now removed a user from a group.

Solaris

On Solaris, providing you are not using NIS or NIS+, use the `/etc/group` file to work with groups.

Creating a group

The file `/etc/group` file will hold group information.

To create a new group, type the following command:

Administration authority

```
groupadd group-name
```

Where *group-name* is the name of the group.

Adding a user to a group

To add a member to a supplementary group, execute the `usermod` command and list the supplementary groups that the user is currently a member of, and the supplementary groups that the user is to become a member of. For example, if the user is a member of the group `groupa`, and is to become a member of `groupb` also, the following command is used:

```
usermod -G groupa,groupb user-name
```

Where *user-name* is the user name.

Displaying who is in a group

To display who is a member of a group, look at the entry for that group in the `/etc/group` file.

Removing a user from a group

To remove a member from a supplementary group, execute the `usermod` command listing the supplementary groups that you want the user to remain a member of. For example, if the user's primary group is `users` and the user is also a member of the groups `mqm`, `groupa` and `groupb`, to remove the user from the `mqm` group, the following command is used:

```
usermod -G groupa,groupb user-name
```

Where *user-name* is the user name.

Linux

On Linux, providing you are not using NIS or NIS+, use the `/etc/group` file to work with groups.

Creating a group

The file `/etc/group` file will hold group information.

To create a new group, type the following command:

```
groupadd -g group-ID group-name
```

Where *group-ID* is the numeric identifier of the group, and *group-name* is the name of the group.

Adding a user to a group

To add a member to a supplementary group, execute the `usermod` command and list the supplementary groups that the user is currently a member of, and the supplementary groups that the user is to become a member of. For example, if the user is a member of the group `groupa`, and is to become a member of `groupb` also, the following command is used:

```
usermod -G groupa,groupb user-name
```

Where *user-name* is the user name.

Displaying who is in a group

To display who is a member of a group, type the following command:

```
getent group group-name
```

Where *group-name* is the name of the group.

Removing a user from a group

To remove a member from a supplementary group, execute the `usermod` command listing the supplementary groups that you want the user to remain a member of. For example, if the user's primary group is `users` and the user is also a member of the groups `mqm`, `groupa` and `groupb`, to remove the user from the `mqm` group, the following command is used:

```
usermod -G groupa,groupb user-name
```

Where *user-name* is the user name.

Using the OAM to control access to objects

The OAM provides a command interface for granting and revoking authority to WebSphere MQ objects. You must be suitably authorized to use these commands, as described in "Authority to administer WebSphere MQ" on page 129. User IDs that are authorized to administer WebSphere MQ have *super user* authority to the queue manager, which means that you do not have to grant them further permission to issue any MQI requests or commands.

Giving access to a WebSphere MQ object

Use the `setmqaut` control command, or the `MQCMD_SET_AUTH_REC` PCF command to give users, and groups of users, access to WebSphere MQ objects. For a full definition of the `setmqaut` control command and its syntax, see "setmqaut (set or reset authority)" on page 368, and for a full definition of the `MQCMD_SET_AUTH_REC` PCF command and its syntax, see the *WebSphere MQ Programmable Command Formats and Administration Interface* book.

The queue manager must be running to use this command. When you have changed access for a principal, the changes are reflected immediately by the OAM.

To give users access to an object, you need to specify:

- The name of the queue manager that owns the objects you are working with; if you do not specify the name of a queue manager, the default queue manager is assumed.
- The name and type of the object (to identify the object uniquely). You specify the name as a *profile*; this is either the explicit name of the object, or a generic name, including wildcard characters. For a detailed description of generic profiles, and the use of wildcard characters within them, see "Using OAM generic profiles" on page 142.
- One or more principals and group names to which the authority applies.

If a user ID contains spaces, enclose it in single quotes when you use this command. On Windows systems, you can qualify a user ID with a domain name. If the actual user ID contains an `@` symbol, replace this with `@@` to show that it is part of the user ID, not the delimiter between the user ID and the domain name.

Administration authority

- A list of authorizations. Each item in the list specifies a type of access that is to be granted to that object (or revoked from it). Each authorization in the list is specified as a keyword, prefixed with a plus sign (+) or a minus sign (-). Use a plus sign to add the specified authorization, and a minus sign to remove the authorization. There must be no spaces between the + or - sign and the keyword.

You can specify any number of authorizations in a single command. For example, the list of authorizations to permit a user or group to put messages on a queue and to browse them, but to revoke access to get messages is:

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

Examples of using the command

The following examples show how to use the **setmqaut** command to grant and revoke permission to use an object:

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

In this example:

- saturn.queue.manager is the queue manager name
- queue is the object type
- RED.LOCAL.QUEUE is the object name
- groupa is the identifier of the group whose authorizations are to change
- +browse -get +put is the authorization list for the specified queue
 - +browse adds authorization to browse messages on the queue (to issue **MQGET** with the browse option)
 - -get removes authorization to get (**MQGET**) messages from the queue
 - +put adds authorization to put (**MQPUT**) messages on the queue

The following command revokes put authority on the queue MyQueue from principal fvuser and from groups groupa and groupb. On UNIX systems, this command also revokes put authority for all principals in the same primary group as fvuser.

```
setmqaut -m saturn.queue.manager -t queue -n MyQueue -p fvuser  
-g groupa -g groupb -put
```

Using the command with a different authorization service

If you are using your own authorization service instead of the OAM, you can specify the name of this service on the **setmqaut** command to direct the command to this service. You must specify this parameter if you have multiple installable components running at the same time; if you do not, the update is made to the first installable component for the authorization service. By default, this is the supplied OAM.

Using OAM generic profiles

OAM generic profiles enable you to set the authority a user has to many objects at once, rather than having to issue separate **setmqaut** commands against each individual object when it is created. Using generic profiles in the **setmqaut** command enables you to set a generic authority for all objects that fit that profile.

The rest of this section describes the use of generic profiles in more detail:

- “Using wildcard characters” on page 143
- “Profile priorities” on page 143
- “Dumping profile settings” on page 144

Using wildcard characters

What makes a profile generic is the use of special characters (wildcard characters) in the profile name. For example, the ? wildcard character matches any single character in a name. So, if you specify ABC.?EF, the authorization you give to that profile applies to any objects with the names ABC.DEF, ABC.CEF, ABC.BEF, and so on.

The wildcard characters available are:

- ? Use the question mark (?) instead of any single character. For example, AB.?D would apply to the objects AB.CD, AB.ED, and AB.FD.
- * Use the asterisk (*) as:
 - A *qualifier* in a profile name to match any one qualifier in an object name. A qualifier is the part of an object name delimited by a period. For example, in ABC.DEF.GHI, the qualifiers are ABC, DEF, and GHI. For example, ABC.*.JKL would apply to the objects ABC.DEF.JKL, and ABC.GHI.JKL. (Note that it would **not** apply to ABC.JKL; * used in this context always indicates one qualifier.)
 - A character within a qualifier in a profile name to match zero or more characters within the qualifier in an object name. For example, ABC.DE*.JKL would apply to the objects ABC.DE.JKL, ABC.DEF.JKL, and ABC.DEGH.JKL.
- ** Use the double asterisk (**) **once** in a profile name as:
 - The entire profile name to match all object names. For example if you use -t prcs to identify processes, then use ** as the profile name, you change the authorizations for all processes.
 - As either the beginning, middle, or ending qualifier in a profile name to match zero or more qualifiers in an object name. For example, **.ABC identifies all objects with the final qualifier ABC.

Note: When using wildcard characters on UNIX systems, you **must** enclose the profile name in quotes.

Profile priorities

An important point to understand when using generic profiles is the priority that profiles are given when deciding what authorities to apply to an object being created. For example, suppose that you have issued the commands:

```
setmqaut -n AB.* -t q +put -p fred
setmqaut -n AB.C* -t q +get -p fred
```

The first gives put authority to all queues for the principal fred with names that match the profile AB.*; the second gives get authority to the same types of queue that match the profile AB.C*.

Suppose that you now create a queue called AB.CD. According to the rules for wildcard matching, either setmqaut could apply to that queue. So, does it have put or get authority?

To find the answer, you apply the rule that, whenever multiple profiles can apply to an object, **only the most specific applies**. The way that you apply this rule is by comparing the profile names from left to right. Wherever they differ, a non-generic character is more specific than a generic character. So, in the example above, the queue AB.CD has **get** authority (AB.C* is more specific than AB.*).

When you are comparing generic characters, the order of *specificity* is:

Administration authority

1. ?
2. *
3. **

Dumping profile settings

The **dmpmqaut** control command and the **MQCMD_INQUIRE_AUTH_RECS** PCF command, enable you to dump the current authorizations associated with a specified profile. For a full definition of the **dmpmqaut** control command and its syntax, see “dmpmqaut (dump authority)” on page 305, and for a full definition of the **MQCMD_INQUIRE_AUTH_RECS** PCF command and its syntax, see the *WebSphere MQ Programmable Command Formats and Administration Interface* book.

The following examples show the use of the **dmpmqaut** control command 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: 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
```

For detailed information on the command, see “dmpmqaut (dump authority)” on page 305.

Displaying access settings

Use the **dspmqa** control command, or the **MQCMD_INQUIRE_ENTITY_AUTH** PCF command to view the authorizations that a specific principal or group has for a particular object. The queue manager must be running to use this command. When you change access for a principal, the changes are reflected immediately by the OAM. Authorization can be displayed for only one group or principal at a time. For a full definition of the **dmpmqaut** control command and its syntax, see “dspmqa (display authority)” on page 312, and for a full definition of the **MQCMD_INQUIRE_ENTITY_AUTH** PCF command and its syntax, see the *WebSphere MQ Programmable Command Formats and Administration Interface* book.

Administration authority

The following example shows the use of the **dspmqaout** control command to display the authorizations of a process definition named **Annuities** in group **GpAdmin** that is on queue manager **QueueMan1**.

```
dspmqaout -m QueueMan1 -t process -n Annuities -g GpAdmin
```

Changing and revoking access to a WebSphere MQ object

To change the level of access that a user or group has to an object, use the **setmqaut** command. To revoke the access of a particular user that is a member of a group that has authorization, remove the user from the group, as described in “Creating and managing groups” on page 136.

The user ID that creates a WebSphere MQ object is granted full control authorities to that object. If you remove this user ID from the local **mqm** group (or the **Administrators** group on Windows systems) these authorities are not revoked. Use the **setmqaut** control command or the **MQCMD_DELETE_AUTH_REC** PCF command to revoke access to an object for the user ID that created it, after removing it from the **mqm** or **Administrators** group. For a full definition of the **setmqaut** control command and its syntax, see “setmqaut (set or reset authority)” on page 368, and for a full definition of the **MQCMD_INQUIRE_ENTITY_AUTH** PCF command and its syntax, see the *WebSphere MQ Programmable Command Formats and Administration Interface* book.

Preventing security access checks

If you decide that you do not want to perform security checks (for example, in a test environment), you can disable the OAM in one of two ways:

- Set the operating system environment variable **MQSNOAUT** as follows, before you create a queue manager (if you do this, you cannot add an OAM later):
 - On Windows systems:

```
SET MQSNOAUT=yes
```
 - On UNIX systems:

```
export MQSNOAUT=yes
```
- Use the WebSphere MQ Explorer or edit the queue manager configuration file to remove the service. (if you do this, you cannot add an OAM later)

If you use **setmqaut**, or **dspmqaout**, while the OAM is disabled, the following will occur:

- The OAM will not validate the specified principal, or group, meaning the command will accept invalid values.
- The OAM will not perform security checks and will indicate that all principals, and groups, are authorized to perform all applicable object operations.

Channel security

Message channel agents (MCAs) are WebSphere MQ applications and need access to various WebSphere MQ resources.

- The user ID associated with a sending channel needs access to the queue manager, the transmission queue, the dead-letter queue, and any resources required by channel exits.
- The user ID associated with the receiving channel needs to open the target queues to put messages onto them. This involves the MQI, so access control checks might need to be made. You can specify whether these checks are made

against the user ID associated with the MCA (as described below), or the user ID associated with the message (from the MQMD context field).

The PUTAUT parameter of the channel definition specifies which user ID is used for these checks.

- If you use the user ID of the MCA, this user ID will already be defined on the local system.
- If you use the user ID associated with the message, it is likely that this is a user ID from a remote system. This remote system user ID must be recognized by the target system and have the authority to connect to the queue manager, make inquiries, set attributes, and set context options (+connect, +inq, +set, and +setall). It must also have authority to put messages and set context information (+put and +setall) for the destination and dead-letter queues.

The user ID associated with the MCA depends on the type of MCA.

Caller MCA

These are MCAs that initiate a channel. They can be started as individual processes, as threads of the channel initiator, or as threads of a process pool. The user ID used is that associated with the parent process (the channel initiator), or the process causing the MCA to be started.

Responder MCA

These are MCAs that are started as a result of a request by a caller MCA. They can be started as individual processes, as threads of the listener, or as threads of a process pool. The user ID can be any one of the following (in this order of preference):

1. On APPC, the caller MCA can indicate the user ID to be used for the responder MCA. This is called the network user ID and applies only to channels started as individual processes. This is set using the USERID parameter of the channel definition.
2. If the USERID parameter is not used, the channel definition of the responder MCA can specify the user ID that the MCA is to use. This is set using the MCAUSER parameter of the channel definition.
3. If the user ID has not been set by either of the methods above, the user ID of the process that starts the MCA or the user ID of the parent process (the listener) is used.

Protecting channel initiator definitions

WebSphere MQ channel initiators are not WebSphere MQ objects; access to them is not controlled by the OAM. WebSphere MQ does not allow users or applications to manipulate these objects, unless their user ID is a member of the mqm group. If you have an application that issues the PCF command **StartChannelInitiator**, the user ID specified in the message descriptor of the PCF message must be a member of the mqm group on the target queue manager.

A user ID must also be a member of the mqm group on the target machine to issue the equivalent MQSC commands through the Escape PCF command or using **runmqsc** in indirect mode.

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 8 on page 151.

Channel exits

You can use channel exits for added security. A security exit forms a secure connection between two security exit programs, where one program is for the sending message channel agent (MCA), and one is for the receiving MCA.

On WebSphere MQ for Windows, there is a security exit for both the WebSphere MQ client and the WebSphere MQ server. This is a channel exit program that provides authentication for WebSphere MQ channels by using the Security Services Programming Interface (SSPI). The supplied channel exit programs provide either one-way or two-way (mutual) authentication of a partner system when a session is being established. For a particular channel, each exit program has an associated principal. A connection between two exit programs is an association between the two principals.

The exit source code file is called `amqsspin.c`, and is stored in the `C:\Program files\IBM\WebSphere MQ\tools\c\samples` directory. If you modify the source code, you must recompile the modified source. The source code does not include any provision for tracing or error handling. If you choose to modify and use the source code, add your own tracing and error-handling routines. You compile and link the file in the same way as any other channel exit, except that SSPI headers need to be accessed at compile time, and SSPI libraries need to be accessed at link time. See *WebSphere MQ for Windows, V6.0 Quick Beginnings* for more information.

See *WebSphere MQ Intercommunication* for more information about channel exits.

Protecting channels with SSL

The Secure Sockets Layer (SSL) protocol provides out of the box channel security, with protection against eavesdropping, tampering, and impersonation. WebSphere MQ support for SSL enables you to specify, on the channel definition, that a particular channel uses SSL security. You can also specify details of the kind of security you want, such as the encryption algorithm you want to use.

SSL support in WebSphere MQ uses the queue manager authentication information object and various MQSC commands and queue manager and channel parameters that define the SSL support required in detail.

The following MQSC commands support SSL:

ALTER AUTHINFO

Modifies the attributes of an authentication information object.

DEFINE AUTHINFO

Creates a new authentication information object.

DELETE AUTHINFO

Deletes an authentication information object.

DISPLAY AUTHINFO

Displays the attributes for a specific authentication information object.

The following queue manager parameters support SSL:

SSLCRLNL

Allows access to a certificate revocation list. The `SSLCRLNL` attribute specifies a namelist. The namelist contains zero or more authentication information objects. Each authentication information object gives access to an LDAP server.

SSLCRYP

On Windows and UNIX systems, sets the SSLCryptoHardware queue manager attribute. This attribute is the name of the parameter string that you can use to configure the cryptographic hardware you have on your system.

SSLEV

Determines whether an SSL event message will be reported if a channel using SSL fails to establish an SSL connection.

SSLFIPS

Specifies whether only FIPS-certified algorithms are to be used if cryptography is carried out in WebSphere MQ. If cryptographic hardware is configured, the cryptographic modules used are those provided by the hardware product, and these may, or may not, be FIPS-certified to a particular level. This depends on the hardware product in use.

SSLKEYR

On Windows and UNIX systems, associates a key repository with a queue manager. The key database is held in a *GSKit* key database. (The IBM Global Security Kit (GSKit) enables you to use SSL security on Windows and UNIX systems systems.)

SSLRKEYC

The number of unencrypted bytes sent and received within an SSL conversation before the secret key is renegotiated. The number of bytes includes control information sent by the MCA.

The following channel parameters support SSL:

SSLCAUTH

Defines whether WebSphere MQ requires and validates a certificate from the SSL client.

SSLCIPH

Specifies the encryption strength and function (CipherSpec), for example NULL_MD5 or RC4_MD5_US. The CipherSpec must match at both ends of channel.

SSLPEER

Specifies the distinguished name (unique identifier) of allowed partners.

This book describes the **setmqaut**, **dspmqaut**, **dmpmqaut**, **rcrmqobj**, **rcdmqimg**, and **dspmqfls** commands to support the authentication information object. It also describes the **amqtcert** command for migrating certificates on Windows systems, and the **IKEYCMD** command for managing certificates on UNIX systems. See the following sections:

- “setmqaut (set or reset authority)” on page 368
- “dspmqaut (display authority)” on page 312
- “dmpmqaut (dump authority)” on page 305
- “rcrmqobj (recreate object)” on page 351
- “rcdmqimg (record media image)” on page 349
- “dspmqfls (display files)” on page 317
- “amqtcert (transfer certificates)” on page 291
- Chapter 18, “Managing keys and certificates,” on page 387

For an overview of channel security using SSL, see *WebSphere MQ Security*.

Administration authority

For details of MQSC commands associated with SSL, see the *WebSphere MQ Script (MQSC) Command Reference*.

For details of PCF commands associated with SSL, see *WebSphere MQ Programmable Command Formats and Administration Interface*.

How authorizations work

The authorization specification tables starting on page 151 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, queue manager, namelist, authentication information, channel, client connection channel, listener, or service.

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, MQZAO_SET_ALL_CONTEXT corresponds to the keyword +setall, and so on. These constants are defined in the header file cmqzc.h, supplied with the product.

Authorizations for MQI calls

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

Four MQI calls might 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 might 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 resolving a name that is not a queue manager alias, unless the queue manager alias definition is opened directly; that is, its name is displayed in the *ObjectName* field of the object descriptor. Authority is always needed for the object being opened. In some cases additional queue-independent authority, obtained through an authorization for the queue manager object, is required.

Table 6 on page 151, Table 7 on page 151, Table 8 on page 151, and Table 9 on page 152 summarize the authorizations needed for each call. In the tables *Not applicable* means that authorization checking is not relevant to this operation; *No check* means that no authorization checking is performed.

Note: You will find no mention of namelists, channels, client connection channels, listeners, services, or authentication information objects in these tables. This is because none of the authorizations apply to these objects, except for MQOO_INQUIRE, for which the same authorizations apply as for the other objects.

The special authorization MQZAO_ALL_MQI includes all the authorizations in the tables that are relevant to the object type, except MQZAO_DELETE and MQZAO_DISPLAY, which are classed as administration authorizations.

Table 6. Security authorization needed for MQCONN calls

Authorization required for:	Queue object (1)	Process object	Queue manager object
MQCONN	Not applicable	Not applicable	MQZAO_CONNECT

Table 7. Security authorization needed for MQOPEN calls

Authorization required for:	Queue object (1)	Process object	Queue manager object
MQOO_INQUIRE	MQZAO_INQUIRE	MQZAO_INQUIRE	MQZAO_INQUIRE
MQOO_BROWSE	MQZAO_BROWSE	Not applicable	No check
MQOO_INPUT_*	MQZAO_INPUT	Not applicable	No check
MQOO_SAVE_ALL_CONTEXT (2)	MQZAO_INPUT	Not applicable	Not applicable
MQOO_OUTPUT (Normal queue) (3)	MQZAO_OUTPUT	Not applicable	Not applicable
MQOO_PASS_IDENTITY_CONTEXT (4)	MQZAO_PASS_IDENTITY_CONTEXT	Not applicable	No check
MQOO_PASS_ALL_CONTEXT (4, 5)	MQZAO_PASS_ALL_CONTEXT	Not applicable	No check
MQOO_SET_IDENTITY_CONTEXT (4, 5)	MQZAO_SET_IDENTITY_CONTEXT	Not applicable	MQZAO_SET_IDENTITY_CONTEXT (6)
MQOO_SET_ALL_CONTEXT (4, 7)	MQZAO_SET_ALL_CONTEXT	Not applicable	MQZAO_SET_ALL_CONTEXT (6)
MQOO_OUTPUT (Transmission queue) (8)	MQZAO_SET_ALL_CONTEXT	Not applicable	MQZAO_SET_ALL_CONTEXT (6)
MQOO_SET	MQZAO_SET	Not applicable	No check
MQOO_ALTERNATE_USER_AUTHORITY	(9)	(9)	MQZAO_ALTERNATE_USER_AUTHORITY (9, 10)

Table 8. Security authorization needed for MQPUT1 calls

Authorization required for:	Queue object (1)	Process object	Queue manager object
MQPMO_PASS_IDENTITY_CONTEXT	MQZAO_PASS_IDENTITY_CONTEXT (11)	Not applicable	No check
MQPMO_PASS_ALL_CONTEXT	MQZAO_PASS_ALL_CONTEXT (11)	Not applicable	No check
MQPMO_SET_IDENTITY_CONTEXT	MQZAO_SET_IDENTITY_CONTEXT (11)	Not applicable	MQZAO_SET_IDENTITY_CONTEXT (6)
MQPMO_SET_ALL_CONTEXT	MQZAO_SET_ALL_CONTEXT (11)	Not applicable	MQZAO_SET_ALL_CONTEXT (6)
(Transmission queue) (8)	MQZAO_SET_ALL_CONTEXT	Not applicable	MQZAO_SET_ALL_CONTEXT (6)
MQPMO_ALTERNATE_USER_AUTHORITY	(12)	Not applicable	MQZAO_ALTERNATE_USER_AUTHORITY (10)

Authorization specification tables

Table 9. Security authorization needed for MQCLOSE calls

Authorization required for:	Queue object (1)	Process object	Queue manager object
MQCO_DELETE	MQZAO_DELETE (13)	Not applicable	Not applicable
MQCO_DELETE_PURGE	MQZAO_DELETE (13)	Not applicable	Not applicable

Notes for the tables:

1. If opening a model queue:
 - MQZAO_DISPLAY authority is needed for the model queue, in addition to the authority to open the model queue for the type of access for which you are opening.
 - MQZAO_CREATE authority is not needed to create the dynamic queue.
 - The user identifier used to open the model queue is automatically granted all the queue-specific authorities (equivalent to MQZAO_ALL) for the dynamic queue created.
2. MQOO_INPUT_* must also be specified. This is valid for a local, model, or alias queue.
3. This check is performed for all output cases, except transmission queues (see note 8).
4. MQOO_OUTPUT must also be specified.
5. MQOO_PASS_IDENTITY_CONTEXT is also implied by this option.
6. This authority is required for both the queue manager object and the particular queue.
7. MQOO_PASS_IDENTITY_CONTEXT, MQOO_PASS_ALL_CONTEXT, and MQOO_SET_IDENTITY_CONTEXT are also implied by this option.
8. 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).
9. At least one of MQOO_INQUIRE (for any object type), or MQOO_BROWSE, MQOO_INPUT_*, MQOO_OUTPUT, or MQOO_SET (for queues) 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.
10. This authorization allows any *AlternateUserId* to be specified.
11. An MQZAO_OUTPUT check is also carried out if the queue does not have a *Usage* queue attribute of MQUS_TRANSMISSION.
12. The check carried out is as for the other options specified, using the supplied alternate-user identifier for the specific-named queue authority, and the current application authority for the MQZAO_ALTERNATE_USER_IDENTIFIER check.
13. 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** call that returned the object handle being used.

Otherwise, there is no check.

Authorizations for MQSC commands in escape PCFs

This section summarizes the authorizations needed for each MQSC command contained in Escape PCF.

Not applicable means that authorization checking is not relevant to this operation.

The user ID under which the program that submits the command is running must also have the following authorities:

- MQZAO_CONNECT authority to the queue manager
- DISPLAY authority on the queue manager in order to perform PCF commands
- Authority to issue the MQSC command within the text of the Escape PCF command

ALTER *object*

Object	Authorization required
Queue	MQZAO_CHANGE
Process	MQZAO_CHANGE
Queue manager	MQZAO_CHANGE
Namelist	MQZAO_CHANGE
Authentication information	MQZAO_CHANGE
Channel	MQZAO_CHANGE
Client connection channel	MQZAO_CHANGE
Listener	MQZAO_CHANGE
Service	MQZAO_CHANGE

CLEAR QLOCAL

Object	Authorization required
Queue	MQZAO_CLEAR
Process	Not applicable
Queue manager	Not applicable
Namelist	Not applicable
Authentication information	Not applicable
Channel	Not applicable
Client connection channel	Not applicable
Listener	Not applicable
Service	Not applicable

DEFINE *object* NOREPLACE (1)

Object	Authorization required
Queue	MQZAO_CREATE (2)
Process	MQZAO_CREATE (2)
Queue manager	Not applicable
Namelist	MQZAO_CREATE (2)
Authentication information	MQZAO_CREATE (2)

Authorization specification tables

Channel	MQZAO_CREATE (2)
Client connection channel	MQZAO_CREATE (2)
Listener	MQZAO_CREATE (2)
Service	MQZAO_CREATE (2)

DEFINE *object* REPLACE (1, 3)

Object	Authorization required
Queue	MQZAO_CHANGE
Process	MQZAO_CHANGE
Queue manager	Not applicable
Namelist	MQZAO_CHANGE
Authentication information	MQZAO_CHANGE
Channel	MQZAO_CHANGE
Client connection channel	MQZAO_CHANGE
Listener	MQZAO_CHANGE
Service	MQZAO_CHANGE

DELETE *object*

Object	Authorization required
Queue	MQZAO_DELETE
Process	MQZAO_DELETE
Queue manager	Not applicable
Namelist	MQZAO_DELETE
Authentication information	MQZAO_DELETE
Channel	MQZAO_DELETE
Client connection channel	MQZAO_DELETE
Listener	MQZAO_DELETE
Service	MQZAO_DELETE

DISPLAY *object*

Object	Authorization required
Queue	MQZAO_DISPLAY
Process	MQZAO_DISPLAY
Queue manager	MQZAO_DISPLAY
Namelist	MQZAO_DISPLAY
Authentication information	MQZAO_DISPLAY
Channel	MQZAO_DISPLAY
Client connection channel	MQZAO_DISPLAY
Listener	MQZAO_DISPLAY
Service	MQZAO_DISPLAY

PING CHANNEL

Object	Authorization required
Queue	Not applicable
Process	Not applicable
Queue manager	Not applicable
Namelist	Not applicable
Authentication information	Not applicable
Channel	MQZAO_CONTROL
Client connection channel	Not applicable
Listener	Not applicable
Service	Not applicable

RESET CHANNEL

Object	Authorization required
Queue	Not applicable
Process	Not applicable
Queue manager	Not applicable
Namelist	Not applicable
Authentication information	Not applicable
Channel	MQZAO_CONTROL_EXTENDED
Client connection channel	Not applicable
Listener	Not applicable
Service	Not applicable

RESOLVE CHANNEL

Object	Authorization required
Queue	Not applicable
Process	Not applicable
Queue manager	Not applicable
Namelist	Not applicable
Authentication information	Not applicable
Channel	MQZAO_CONTROL_EXTENDED
Client connection channel	Not applicable
Listener	Not applicable
Service	Not applicable

START CHANNEL/LISTENER/SERVICE

Object	Authorization required
Queue	Not applicable
Process	Not applicable
Queue manager	Not applicable

Authorization specification tables

Namelist	Not applicable
Authentication information	Not applicable
Channel	MQZAO_CONTROL
Client connection channel	Not applicable
Listener	MQZAO_CONTROL
Service	MQZAO_CONTROL

STOP CHANNEL/LISTENER/SERVICE

Object	Authorization required
Queue	Not applicable
Process	Not applicable
Queue manager	Not applicable
Namelist	Not applicable
Authentication information	Not applicable
Channel	MQZAO_CONTROL
Client connection channel	Not applicable
Listener	MQZAO_CONTROL
Service	MQZAO_CONTROL

Note:

1. 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.
2. 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.
3. This applies if the object to be replaced already exists. If it does not, the check is as for DEFINE *object* NOREPLACE.

Authorizations for PCF commands

This section summarizes the authorizations needed for each PCF command.

No check means that no authorization checking is carried out; *Not applicable* means that authorization checking is not relevant to this operation.

The user ID under which the program that submits the command is running must also have the following authorities:

- MQZAO_CONNECT authority to the queue manager
- DISPLAY authority on the queue manager in order to perform PCF commands

The special authorization MQZAO_ALL_ADMIN includes all the authorizations in 157 that are relevant to the object type, except MQZAO_CREATE, which is not specific to a particular object or object type.

Change object

Object	Authorization required
Queue	MQZAO_CHANGE
Process	MQZAO_CHANGE
Queue manager	MQZAO_CHANGE
Namelist	MQZAO_CHANGE
Authentication information	MQZAO_CHANGE
Channel	MQZAO_CHANGE
Client connection channel	MQZAO_CHANGE
Listener	MQZAO_CHANGE
Service	MQZAO_CHANGE

Clear Queue

Object	Authorization required
Queue	MQZAO_CLEAR
Process	Not applicable
Queue manager	Not applicable
Namelist	Not applicable
Authentication information	Not applicable
Channel	Not applicable
Client connection channel	Not applicable
Listener	Not applicable
Service	Not applicable

Copy object (without replace) (1)

Object	Authorization required
Queue	MQZAO_CREATE (2)
Process	MQZAO_CREATE (2)
Queue manager	Not applicable
NamelistMQZAO_CREATE	MQZAO_CREATE (2)
Authentication information	MQZAO_CREATE (2)
Channel	MQZAO_CREATE (2)
Client connection channel	MQZAO_CREATE (2)
Listener	MQZAO_CREATE (2)
Service	MQZAO_CREATE (2)

Copy object (with replace) (1, 4)

Object	Authorization required
Queue	MQZAO_CHANGE
Process	MQZAO_CHANGE
Queue manager	Not applicable

Authorization specification tables

Namelist	MQZAO_CHANGE
Authentication information	MQZAO_CHANGE
Channel	MQZAO_CHANGE
Client connection channel	MQZAO_CHANGE
Listener	MQZAO_CHANGE
Service	MQZAO_CHANGE

Create *object* (without replace) (3)

Object	Authorization required
Queue	MQZAO_CREATE (2)
Process	MQZAO_CREATE (2)
Queue manager	Not applicable
Namelist	MQZAO_CREATE (2)
Authentication information	MQZAO_CREATE (2)
Channel	MQZAO_CREATE (2)
Client connection channel	MQZAO_CREATE (2)
Listener	MQZAO_CREATE (2)
Service	MQZAO_CREATE (2)

Create *object* (with replace) (3, 4)

Object	Authorization required
Queue	MQZAO_CHANGE
Process	MQZAO_CHANGE
Queue manager	Not applicable
Namelist	MQZAO_CHANGE
Authentication information	MQZAO_CHANGE
Channel	MQZAO_CHANGE
Client connection channel	MQZAO_CHANGE
Listener	MQZAO_CHANGE
Service	MQZAO_CHANGE

Delete *object*

Object	Authorization required
Queue	MQZAO_DELETE
Process	MQZAO_DELETE
Queue manager	MQZAO_DELETE
Namelist	MQZAO_DELETE
Authentication information	MQZAO_DELETE
Channel	MQZAO_DELETE
Client connection channel	MQZAO_DELETE
Listener	MQZAO_DELETE

Authorization specification tables

Service	MQZAO_DELETE
---------	--------------

Inquire *object*

Object	Authorization required
Queue	MQZAO_DISPLAY
Process	MQZAO_DISPLAY
Queue manager	MQZAO_DISPLAY
Namelist	MQZAO_DISPLAY
Authentication information	MQZAO_DISPLAY
Channel	MQZAO_DISPLAY
Client connection channel	MQZAO_DISPLAY
Listener	MQZAO_DISPLAY
Service	MQZAO_DISPLAY

Inquire *object names*

Object	Authorization required
Queue	No check
Process	No check
Queue manager	No check
Namelist	No check
Authentication information	No check
Channel	No check
Client connection channel	No check
Listener	No check
Service	No check

Ping Channel

Object	Authorization required
Queue	Not applicable
Process	Not applicable
Queue manager	Not applicable
Namelist	Not applicable
Authentication information	Not applicable
Channel	MQZAO_CONTROL
Client connection channel	Not applicable
Listener	Not applicable
Service	Not applicable

Authorization specification tables

Reset Channel

Object	Authorization required
Queue	Not applicable
Process	Not applicable
Queue manager	Not applicable
Namelist	Not applicable
Authentication information	Not applicable
Channel	MQZAO_CONTROL_EXTENDED
Client connection channel	Not applicable
Listener	Not applicable
Service	Not applicable

Reset Queue Statistics

Object	Authorization required
Queue	MQZAO_DISPLAY and MQZAO_CHANGE
Process	Not applicable
Queue manager	Not applicable
Namelist	Not applicable
Authentication information	Not applicable
Channel	Not applicable
Client connection channel	Not applicable
Listener	Not applicable
Service	Not applicable

Resolve Channel

Object	Authorization required
Queue	Not applicable
Process	Not applicable
Queue manager	Not applicable
Namelist	Not applicable
Authentication information	Not applicable
Channel	MQZAO_CONTROL_EXTENDED
Client connection channel	Not applicable
Listener	Not applicable
Service	Not applicable

Start Channel/Listener/Service

Object	Authorization required
Queue	Not applicable
Process	Not applicable
Queue manager	Not applicable

Namelist	Not applicable
Authentication information	Not applicable
Channel	MQZAO_CONTROL
Client connection channel	Not applicable
Listener	MQZAO_CONTROL
Service	MQZAO_CONTROL

Stop Channel/Listener/Service

Object	Authorization required
Queue	Not applicable
Process	Not applicable
Queue manager	Not applicable
Namelist	Not applicable
Authentication information	Not applicable
Channel	MQZAO_CONTROL
Client connection channel	Not applicable
Listener	MQZAO_CONTROL
Service	MQZAO_CONTROL

Note:

1. For Copy commands, MQZAO_DISPLAY authority is also needed for the From object.
2. 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.
3. For Create commands, MQZAO_DISPLAY authority is also needed for the appropriate SYSTEM.DEFAULT.* object.
4. This applies if the object to be replaced already exists. If it does not, the check is as for Copy or Create without replace.

Guidelines for Windows 2000 and Windows 2003

This information applies only when you are running WebSphere MQ in a Windows 2000 or Windows 2003 environment.

WebSphere MQ for Windows runs on Windows 2003, Windows 2000, and Windows XP, but the operation of WebSphere MQ security can be affected by differences between the platforms.

WebSphere MQ security relies on calls to the operating system API for information about user authorizations and group memberships. Some functions do not behave identically on the Windows systems. This section includes descriptions of how those differences might affect WebSphere MQ security when you are running WebSphere MQ in a Windows 2000 or Windows 2003 environment.

When you get a 'group not found' error

This problem can arise because WebSphere MQ loses access to the local mqm group when Windows 2000 or Windows 2003 servers are promoted to, or demoted from, domain controllers. The symptom is an error indicating the lack of a local mqm group, for example:

```
>crtmqm qm0  
AMQ8066:Local mqm group not found.
```

Altering the state of a machine between server and domain controller can affect the operation of WebSphere MQ, because WebSphere MQ uses a locally-defined mqm group. When a server is promoted to be a domain controller, the scope changes from local to domain local. When the machine is demoted to server, all domain local groups are removed. This means that changing a machine from server to domain controller and back to server loses access to a local mqm group.

To remedy this problem, recreate the local mqm group using the standard Windows 2000 or Windows 2003 management tools. Because all group membership information is lost, you must reinstate privileged WebSphere MQ users in the newly-created local mqm group. If the machine is a domain member, you must also add the domain mqm group to the local mqm group to grant privileged domain WebSphere MQ user IDs the required level of authority.

When you have problems with WebSphere MQ and domain controllers

This section describes problems that can arise with security settings when Windows 2000 servers are promoted to domain controllers.

While promoting Windows 2000 or Windows 2003 servers to domain controllers, you are presented with the option of selecting a default or non-default security setting relating to user and group permissions. This option controls whether arbitrary users are able to retrieve group memberships from the active directory. Because WebSphere MQ relies on group membership information to implement its security policy, it is important that the user ID that is performing WebSphere MQ operations can determine the group memberships of other users.

On Windows 2000, when a domain is created using the default security option, the default user ID created by WebSphere MQ during the installation process (MUSR_MQADMIN) can obtain group memberships for other users as required. The product then installs normally, creating default objects, and the queue manager can determine the access authority of local and domain users if required.

On Windows 2000, when a domain is created using the non-default security option, or on Windows 2003 when a domain is created using the default security option, the user ID created by WebSphere MQ during the installation (MUSR_MQADMIN) cannot always determine the required group memberships. In this case, you need to know:

- How Windows 2000 with non-default security permissions behaves
- How to allow domain mqm group members to read group membership
- How to configure WebSphere MQ Services to run under a domain user

Windows 2000 domain with non-default, or Windows 2003 domain with default, security permissions

If a **local** user installs WebSphere MQ, the Prepare WebSphere MQ Wizard detects that the local user (MUSR_MQADMIN) created for the WebSphere MQ services

(AMQMSRVN) can retrieve the group membership information of the installing user. The Prepare WebSphere MQ Wizard asks the user questions about the network configuration to determine whether there are other user accounts defined on domain controllers running Windows 2000. If so, the WebSphere MQ services need to run under a domain user account with particular settings and authorities. The Prepare WebSphere MQ Wizard prompts the user for the account details of this user. Its online help provides details of the domain user account required that can be sent to the domain administrator.

If a **domain** user installs WebSphere MQ, the Prepare WebSphere MQ Wizard detects that the local user (MUSR_MQADMIN) created for the WebSphere MQ services (AMQMSRVN) cannot retrieve the group membership information of the installing user. In this case, the Prepare WebSphere MQ Wizard always prompts the user for the account details of the domain user account for the WebSphere MQ services to use.

When WebSphere MQ services needs to use a domain user account, WebSphere MQ cannot operate correctly until this has been configured using the Prepare WebSphere MQ Wizard. This configuration includes creating default objects such as the Default Configuration. The Prepare WebSphere MQ Wizard does not allow the user to continue with other tasks, such as creating the Default Configuration, until the WebSphere MQ services have been configured with a suitable account.

If a Windows 2000 domain has been configured with non-default security permissions, the usual solution to enable WebSphere MQ to work correctly is to configure it with a suitable domain user account, as described above.

See the *WebSphere MQ for Windows, V6.0 Quick Beginnings* for more information.

Configuring WebSphere MQ Services to run under a domain user

Use the Prepare WebSphere MQ Wizard to enter the account details of the domain user account. Alternatively, use the following command line to set the domain user account:

```
AMQMSRVN -user [domain]\[userid] -password [password]
```

In either case, WebSphere MQ allocates the correct security rights and group membership to the new user account.

Applying security template files

Windows 2000 supports text-based security template files that you can use to apply uniform security settings to one or more computers with the Security Configuration and Analysis MMC snap-in. In particular, Windows 2000 supplies several templates that include a range of security settings with the aim of providing specific levels of security. These include compatible, basic, secure, and highly-secure.

Be aware that applying one of these templates might affect the security settings applied to WebSphere MQ files and directories. If you want to use the highly-secure template, configure your machine before you install WebSphere MQ. If you apply the highly-secure template to a machine on which WebSphere MQ is already installed, all the permissions you have specifically set on the WebSphere MQ files and directories are removed. This means that you lose Administrator and mqm group access and, when applicable, Everyone group access from the error directories.

Nested groups

Windows 2000 and Windows 2003 domain controllers by default are placed in function level *2000 mixed*. When using this functional level users cannot add or nest local groups.

You can place Windows 2000 domain controllers in functional level *2000 native*, or Windows 2003 domain controllers in functional levels *2000 native* or *Server 2003*. This allows users to add or nest local groups, and also to perform multiple nesting of global and universal groups. The WebSphere MQ security model does not support either nested local groups, or multiple nesting of global and universal groups. This means that local and domain local groups are supported, as are any immediately nested global or universal groups.

Chapter 11. Transactional support

This chapter introduces transactional support. The work required to enable your applications to use WebSphere MQ in conjunction with a database product spans the areas of application programming and system administration. Use the information here together with Chapter 13, "Committing and backing out units of work" of the *WebSphere MQ Application Programming Guide*.

We start by introducing the units of work that form transactions, then describe the ways in which you enable WebSphere MQ to coordinate transactions with databases.

Introducing units of work

This section introduces the general concepts of commit, backout, syncpoint, and unit of work. If you are familiar with these transaction processing terms, you can skip to "Scenario 1: Queue manager performs the coordination" on page 166, where we start to deal in more detail with their use with WebSphere MQ.

When a program puts messages on queues within a unit of work, those messages are made visible to other programs only when the program *commits* the unit of work. To commit a unit of work, all updates must be successful to preserve data integrity.

If the program detects an error and decides not to make the put operation permanent, it can *back out* the unit of work. When a program performs a backout, WebSphere MQ restores the queues by removing the messages that were put on the queues by that unit of work.

Similarly, when a program gets messages from one or more queues within a unit of work, those messages remain on the queues until the program commits the unit of work, but the messages are not available to be retrieved by other programs. The messages are permanently deleted from the queues when the program commits the unit of work. If the program backs out the unit of work, WebSphere MQ restores the queues by making the messages available to be retrieved by other programs.

The decision to commit or back out the changes is taken, in the simplest case, at the end of a task. However, it can be more useful for an application to synchronize data changes at other logical points within a task. These logical points are called syncpoints (or synchronization points) and the period of processing a set of updates between two syncpoints is called a *unit of work*. Several MQGET calls and MQPUT calls can be part of a single unit of work.

With WebSphere MQ, we need to distinguish between *local* and *global* units of work:

Local units of work

Are those in which the only actions are puts to, and gets from, WebSphere MQ queues, and the coordination of each unit of work is provided within the queue manager using a *single-phase commit* process.

Use local units of work when the only resources to be updated are the queues managed by a single WebSphere MQ queue manager. Updates are committed using the MQCMIT verb or backed out using MQBACK.

Transactional support

There are no system administration tasks, other than log management, involved in using local units of work. In your applications, where you use the MQPUT and MQGET calls with MQCMIT and MQBACK, try using the MQPMO_SYNCPOINT and MQGMO_SYNCPOINT options. (For information on log management, see “Managing log files” on page 230.)

Global units of work

Are those in which other resources, such as tables in a relational database, are also updated. When more than one *resource manager* is involved, there is a need for *transaction manager* software that uses a *two-phase commit* process to coordinate the global unit of work.

Use global units of work when you also need to include updates to relational database manager software, such as DB2, Oracle, Sybase, and Informix.

We define two scenarios for global units of work:

1. In the first, the queue manager itself acts as the transaction manager. In this scenario, MQI verbs control the global units of work; they are started in applications using the MQBEGIN verb and then committed using MQCMIT or backed out using MQBACK.
2. In the second, the transaction manager role is performed by other software, such as TXSeries, Encina[®], or Tuxedo. In this scenario, an API provided by the transaction manager software is used to control the unit of work (for example, EXEC CICS SYNCPOINT for TXSeries).

The following sections describe all the steps necessary to use global units of work, organized by the two scenarios:

- “Scenario 1: Queue manager performs the coordination” below
- “Scenario 2: Other software provides the coordination” on page 190

Scenario 1: Queue manager performs the coordination

This section describes this scenario, including:

- “Database coordination”
- “DB2 configuration” on page 173
- “Oracle configuration” on page 176
- “Informix configuration” on page 178
- “Sybase configuration” on page 180
- “Multiple database configurations” on page 182
- “Security considerations” on page 183
- “Administration tasks” on page 183

Database coordination

When the queue manager coordinates global units of work itself, it becomes possible to integrate database updates within the 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 the two-phase commit protocol described in *X/Open Distributed Transaction Processing: The XA Specification*. 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 the participants,

including the queue manager itself, are prepared to commit, are all the queue and database updates committed. If any participant cannot prepare its updates, the unit of work is backed out instead.

In general, a global unit of work is implemented in an application by the following method (in pseudocode):

```
MQBEGIN
MQGET (include the flag MQGMO_SYNCPOINT in the message options)
MQPUT (include the flag MQPMO_SYNCPOINT in the message options)
SQL INSERT
MQCMIT
```

The purpose of MQBEGIN is to denote the beginning of a global unit of work. The purpose of MQCMIT is to denote the end of the global unit of work, and to complete it with all participating resource managers, using the two-phase commit protocol.

When the unit of work (also known as a *transaction*) is completed successfully using MQCMIT, all actions taken within that unit of work are made permanent or irreversible. If, for any reason, the unit of work fails, all actions are instead backed out. It is not acceptable for one action comprising a unit of work to be made permanent while another is forgotten. This is the principle of a unit of work: either all actions within the unit of work are made permanent or none of them are.

Notes:

1. The application programmer can force a unit of work to be backed out by calling MQBACK. The unit of work is also backed out by the queue manager if the application or database *crashes* before MQCMIT is called.
2. If an application calls MQDISC without calling MQCMIT, the queue manager behaves as if MQCMIT had been called, and commits the unit of work.

In between MQBEGIN and MQCMIT, the queue manager does not make any calls to the database to update its resources. That is, the only way a database's tables are changed is by your code (for example, the SQL INSERT in the pseudocode above).

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 successfully prepared to commit, but has yet to receive a commit or backout decision, the queue manager remembers the outcome of the unit of work until that outcome has been successfully delivered to the database. Similarly, if the queue manager terminates with incomplete commit operations outstanding, these are remembered over queue manager restart. If an application terminates unexpectedly, the integrity of the unit of work is not compromised, but the outcome depends on where in the process the application terminated, as described in Table 11 on page 168.

What happens when the database or application program crashes is summarized in the tables below:

Table 10. What happens when a database server crashes

Before the application call to MQCMIT.	The unit of work is backed out.
During the application call to MQCMIT, before all databases have indicated that they have successfully prepared.	The unit of work is backed out with a reason code of MQRC_BACKED_OUT.

Database coordination

Table 10. What happens when a database server crashes (continued)

During the application call to MQCMIT, after all databases have indicated that they have successfully prepared, but before all have indicated that they have successfully committed.	The unit of work is held in recoverable state by the queue manager, with a reason code of MQRC_OUTCOME_PENDING.
During the application call to MQCMIT, after all databases have indicated that they have successfully committed.	The unit of work is committed with a reason code of MQRC_NONE.
After the application call to MQCMIT.	The unit of work is committed with a reason code of MQRC_NONE.

Table 11. What happens when an application program crashes

Before the application call to MQCMIT.	The unit of work is backed out.
During the application call to MQCMIT, before the queue manager has received the application's MQCMIT request.	The unit of work is backed out.
During the application call to MQCMIT, after the queue manager has received the application's MQCMIT request.	The queue manager tries to commit using two-phase commit (subject to the database products successfully executing and committing their parts of the unit of work).

In the case where the reason code on return from MQCMIT is MQRC_OUTCOME_PENDING, the unit of work is remembered by the queue manager until it has been able to reestablish contact with the database server, and tell it to commit its part of the unit of work. Refer to "Administration tasks" on page 183 for information on how and when recovery is done.

The queue manager communicates with database managers using the XA interface as described in *X/Open Distributed Transaction Processing: The XA Specification*. Examples of these function calls are `xa_open`, `xa_start`, `xa_end`, `xa_prepare`, and `xa_commit`. We use the terms *transaction manager* and *resource manager* in the same sense as they are used in the XA specification.

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 use of MQBEGIN in a client application fails, as described in the *WebSphere MQ Application Programming Reference*. A program that calls MQBEGIN must run as a *server* application on the same machine as the queue manager.

Note: A *server* application is a program that has been linked with the necessary WebSphere MQ server libraries; a *client* application is a program that has been linked with the necessary WebSphere MQ client libraries. See *WebSphere MQ Clients* and the *WebSphere MQ Application Programming Guide* for details on compiling and linking your programs.

- The database server can reside on a different machine from the queue manager server, as long as the database client is installed on the same machine as the queue manager, and it supports this function. Consult the database product's documentation to determine whether their client software can be used for two-phase commit systems.

- Although the queue manager behaves as a resource manager (for the purposes of being involved in Scenario 2 global units of work), it is not possible to make one queue manager coordinate another queue manager within its Scenario 1 global units of work.

Switch load files

The switch load file is a shared library (a DLL on Windows systems) that is loaded by the code in your WebSphere MQ application and the queue manager. Its purpose is to simplify the loading of the database's client shared library, and to return the pointers to the XA functions.

The details of the switch load file must be specified before the queue manager is started. The details are placed in the `qm.ini` file (UNIX systems), or the Registry (Windows systems).

- On Windows or Linux (x86 platform) systems use the WebSphere MQ Explorer. On Windows systems the Registry is updated. On Linux (x86 platform) systems the file, `qm.ini`, is updated.
- On all other systems edit the file, `qm.ini`, directly.

The C source for the switch load file is supplied with the WebSphere MQ installation if it supports Scenario 1 global units of work. The source contains a function called `MQStart`. When the switch load file is loaded, the queue manager calls this function, which returns the address of a structure called an *XA switch*.

The XA switch structure exists in the database client shared library, and contains a number of function pointers, as described in Table 12:

Table 12. XA switch function pointers

Function pointer name	XA function	Purpose
<code>xa_open_entry</code>	<code>xa_open</code>	Connect to database
<code>xa_close_entry</code>	<code>xa_close</code>	Disconnect from database
<code>xa_start_entry</code>	<code>xa_start</code>	Start a branch of a global unit of work
<code>xa_end_entry</code>	<code>xa_end</code>	Suspend a branch of a global unit of work
<code>xa_rollback_entry</code>	<code>xa_rollback</code>	Roll back a branch of a global unit of work
<code>xa_prepare_entry</code>	<code>xa_prepare</code>	Prepare to commit a branch of a global unit of work
<code>xa_commit_entry</code>	<code>xa_commit</code>	Commit a branch of a global unit of work
<code>xa_recover_entry</code>	<code>xa_recover</code>	Discover from the database whether it has an in-doubt unit of work
<code>xa_forget_entry</code>	<code>xa_forget</code>	Allow a database to forget a branch of a global unit of work
<code>xa_complete_entry</code>	<code>xa_complete</code>	Complete a branch of a global unit of work

During the first `MQBEGIN` call in your application, the WebSphere MQ code that executes as part of `MQBEGIN` loads the switch load file, and calls the `xa_open`

Restrictions

function in the database shared library. Similarly, during queue manager startup, and on other subsequent occasions, some queue manager processes load the switch load file and call `xa_open`.

You can reduce the number of `xa_*` calls by using *dynamic registration*. For a complete description of this optimization technique, see “XA dynamic registration” on page 187.

Configuring your system for database coordination

There are several tasks that you must perform before a database manager can participate in global units of works coordinated by the queue manager. These are described here as follows:

- “Installing and configuring the database product”
- “Creating switch load files”
- “Adding configuration information to the queue manager” on page 171
- “Writing and modifying your applications” on page 173
- “Testing the system” on page 173

Installing and configuring the database product: The steps involved in installing and configuring your database product are, of course, described in that product’s own documentation. Installation issues are well beyond the scope of this chapter, but we can list general configuration issues, as they relate to the interoperation between WebSphere MQ and the database.

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. (A connection that is not a *fastpath* connection is a *standard* connection in this context. For more information, see “Connecting to a queue manager using the MQCONNX call” in the *WebSphere MQ Application Programming Guide*.)

When the application issues **MQBEGIN**, both it and the agent process call the `xa_open` function in the database client library. In response to this, the database client library code *connects* to the database that is to be involved in the unit of work *from both the application and queue manager processes*. These database connections are maintained as long as 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, because two connections are being made to the database to support the one application program.

Client/server configuration: The database client library that is loaded into the WebSphere MQ queue manager and application processes **must** be able to send to and receive from its server. Ensure that:

- The database’s client/server configuration files have the correct details
- The relevant environment variables are set in the environment of the queue manager **and** the application processes

Creating switch load files: WebSphere MQ comes with a sample makefile, used to build switch load files for the supported database managers. This makefile, together with all the associated C source files required to build the switch load files, is installed in the following directories:

- For WebSphere MQ for Windows, in the `C:\Program Files\IBM\WebSphere MQ\tools\c\samples\xatm\` directory

- For WebSphere MQ for UNIX systems, in the /opt/mqm/samp/xatm/ directory (/usr/mqm/samp/xatm on AIX)

The sample source modules used to build the switch load files are:

- For DB2, db2swit.c
- For Oracle, oraswit.c
- For Informix, infswit.c
- For Sybase, sybswit.c

When you generate switch load files it is recommended that 32-bit switch load files are installed in /var/mqm/exits and 64-bit switch load files are installed in /var/mqm/exits64.

If you have 32-bit queue managers then the sample make file, xaswit.mak, will install a 32-bit switch load file in /var/mqm/exits.

If you have 64-bit queue managers then the sample make file, xaswit.mak, will install a 32-bit switch load file in /var/mqm/exits, and a 64-bit switch load file in /var/mqm/exits64.

Adding configuration information to the queue manager: When you have created a switch load file for your database manager, and placed it in a safe location, you must specify that location to your queue manager.

- On Windows and Linux (x86 platform) systems use the WebSphere MQ Explorer. Specify the details of the switch load file in the queue manager properties panel, under XA resource manager.
- On all other systems specify the details of the switch load file in the XAResourceManager stanza in the queue manager's qm.ini file.

Add an XAResourceManager stanza for the database that your queue manager is going to coordinate. The most common case is for there to be only one database, and therefore only one XAResourceManager stanza. More complicated configurations involving multiple databases, are discussed in "Multiple database configurations" on page 182. The attributes of the XAResourceManager stanza are as follows:

Name=name

User-chosen string that identifies the resource manager. In effect, it gives a name to the XAResourceManager stanza. The name is mandatory and can be up to 31 characters in length.

The name you choose must be unique; there must be only one XAResourceManager stanza with this name in this qm.ini file. The name should also be meaningful, because the queue manager uses it to refer to this resource manager both in queue manager error log messages and in output when the **dspmqtrn** command is used. (See "Displaying outstanding units of work with the dspmqtrn command" on page 184 for more information.)

Once you have chosen a name, and have started the queue manager, do not change the Name attribute. This is discussed in more detail in "Changing configuration information" on page 186.

SwitchFile=name

This is the name of the XA switch load file you built earlier. This is a mandatory attribute. The code in the queue manager and WebSphere MQ application processes tries to load the switch load file on two occasions:

1. At queue manager startup

Database connections

2. When you make the first call to MQBEGIN in your WebSphere MQ application process

The security and permissions attributes of your switch load file must allow these processes to perform this action.

XAOpenString=string

This is a string of data that WebSphere MQ code passes in its calls to the database manager's xa_open function. This is an optional attribute; if it is omitted a zero-length string is assumed.

The code in the queue manager and WebSphere MQ application processes call the xa_open function on two occasions:

1. At queue manager startup
2. When you make the first call to MQBEGIN in your WebSphere MQ application process

The format for this string is particular to each database product, and will be described in the documentation for that product. In general, the xa_open string contains authentication information (user name and password) to allow a connection to the database in both the queue manager and the application processes.

XACloseString=string

This is a string of data that WebSphere MQ code passes in its calls to the database manager's xa_close function. This is an optional attribute; if it is omitted a zero-length string is assumed.

The code in the queue manager and WebSphere MQ application processes call the xa_close function on two occasions:

1. At queue manager startup
2. When you make a call to MQDISC in your WebSphere MQ application process, having earlier made a call to MQBEGIN

The format for this string is particular to each database product, and will be described in the documentation for that product. In general, the string is empty, and it is common to omit the XACloseString attribute from the XAResourceManager stanza.

ThreadOfControl=THREAD | PROCESS

The ThreadOfControl value can be THREAD or PROCESS. The queue manager uses it for serialization purposes. This is an optional attribute; if it is omitted, the value PROCESS is assumed.

If the database client code allows threads to call the XA functions without serialization, the value for ThreadOfControl can be THREAD. The queue manager assumes that it can call the XA functions in the database client shared library from multiple threads at the same time, if necessary.

If the database client code does not allow threads to call its XA functions in this way, the value for ThreadOfControl must be PROCESS. In this case, the queue manager serializes all calls to the database client shared library so that only one call at a time is made from within a particular process. You probably also need to ensure that your application performs similar serialization if it runs with multiple threads.

Note that this issue, of the database product's ability to cope with multi-threaded processes in this way, is an issue for that product's vendor. Consult the database product's documentation for details on whether you can set the ThreadOfControl attribute to THREAD or PROCESS. We recommend

that, if you can, you set ThreadOfControl to THREAD. If in doubt, the *safer* option is to set it to PROCESS, although you will lose the potential performance benefits of using THREAD.

Writing and modifying your applications: The sample application programs for Scenario 1 global units of work that are supplied with a WebSphere MQ installation are described in the *WebSphere MQ Application Programming Guide*.

In general, a global unit of work is implemented in an application by the following method (in pseudocode):

```
MQBEGIN
MQGET
MQPUT
SQL INSERT
MQCMIT
```

The purpose of MQBEGIN is to denote the beginning of a global unit of work. The purpose of MQCMIT is to denote the end of the global unit of work, and to complete it with all participating resource managers, using the two-phase commit protocol.

In between MQBEGIN and MQCMIT, the queue manager does not make any calls to the database to update its resources. That is, the only way a database's tables are changed is by your code (for example, the SQL INSERT in the pseudocode above).

The role of the queue manager, as far as the database is concerned, is to tell it when a global unit of work has started, when it has ended, and whether the global unit of work should be committed or rolled-back.

As far as your application is concerned, the queue manager performs two roles: a resource manager (where the resources are messages on queues) and the transaction manager for the global unit of work.

We recommend that you start with the supplied sample programs, and work through the various WebSphere MQ and database API calls that are being made in those programs. The API calls concerned are fully documented in the *WebSphere MQ Application Programming Guide*, the *WebSphere MQ Application Programming Reference*, and (in the case of the database's own API) the database's own documentation.

Testing the system: You only know whether your application and system are correctly configured by running them during testing. You can test the system's configuration (the successful communication between queue manager and database) by building and running one of the supplied sample programs.

DB2 configuration

The supported levels of DB2 are defined at:

<http://www.ibm.com/software/ts/mqseries/platforms/supported.html>

Note: 32-bit instances of DB2 are not supported on platforms where the queue manager is 64-bit.

Do the following:

1. Check the environment variable settings.

DB2 configuration

2. Create the DB2 switch load file.
3. Add resource manager configuration information.
4. Change DB2 configuration parameters if necessary.

Read this information in conjunction with the general information provided in “Configuring your system for database coordination” on page 170.

Checking the DB2 environment variable settings

Ensure that your DB2 environment variables are set for queue manager processes *as well as in* your application processes. In particular, you must always set the DB2INSTANCE environment variable *before* you start the queue manager. The DB2INSTANCE environment variable identifies the DB2 instance containing the DB2 databases that are being updated. For example:

- On UNIX systems, use:

```
export DB2INSTANCE=db2inst1
```
- On Windows systems, use:

```
set DB2INSTANCE=DB2
```

Warning: If you run `db2profile` on UNIX platforms, the environment variable LIBPATH and LD_LIBRARY_PATH are set. It is advisable to unset these environment variables, see appropriate *Quick Beginnings* Guide.

Creating the DB2 switch load file

The easiest way to create the DB2 switch load file is to use the sample file `xaswit.mak`, which WebSphere MQ provides to build the switch load files for a variety of database products.

On Windows systems, you can find `xaswit.mak` in the directory `C:\Program Files\IBM\WebSphere MQ\tools\c\samples\xatm`. To create the DB2 switch load file with Microsoft Visual C++, use:

```
nmake /f xaswit.mak db2swit.dll
```

The generated switch file is placed in `c:\Program Files\IBM\WebSphere MQ\exits`.

On AIX, you can find `xaswit.mak` in the directory `/usr/mqm/samp/xatm`; on other UNIX systems, you can find it in the directory `/opt/mqm/samp/xatm`.

Edit `xaswit.mak` to *uncomment* the lines appropriate to the version of DB2 you are using. Then execute the makefile using the command:

```
make -f xaswit.mak db2swit
```

The generated 32-bit switch load file is placed in `/var/mqm/exits`.

The generated 64-bit switch load file is placed in `/var/mqm/exits64`.

Adding resource manager configuration information for DB2

The next step is to modify the configuration information for the queue manager, as described in “Adding configuration information to the queue manager” on page 171, to declare DB2 as a participant in global units of work.

- On Windows and Linux (x86 platform) systems use the WebSphere MQ Explorer. Specify the details of the switch load file in the queue manager properties panel, under XA resource manager.
- On all other systems specify the details of the switch load file in the XAResourceManager stanza in the queue manager’s `qm.ini` file.

Figure 11 is a UNIX sample, showing an XAResourceManager entry where the database to be coordinated is called mydbname, this name being specified in the XAOpenString:

```
XAResourceManager:
  Name=mydb2
  SwitchFile=db2swit
  XAOpenString=mydbname,myuser,mypasswd,toc=t
  ThreadOfControl=THREAD
```

Figure 11. Sample XAResourceManager entry for DB2 on UNIX platforms

Notes:

1. ThreadOfControl=THREAD cannot be used with DB2 versions prior to version 8. It is recommended that you always set ThreadOfControl and the XAOpenString parameter toc to one of the following combinations:
 - ThreadOfControl=THREAD and toc=t
 - ThreadOfControl=PROCESS and toc=p

Changing DB2 configuration parameters

For each DB2 database that the queue manager is coordinating, you need to:

Set database privileges

The queue manager processes run with effective user and group mqm on UNIX systems. On Windows systems, they run as the user that started the queue manager. This can be one of:

1. The user who issued the **strmqm** command, or
2. The user under which the IBM MQSeries Service COM server runs

By default, this user is called MUSR_MQADMIN.

If you have not specified a user name and password on the xa_open string, **the user under which the queue manager is running** is used by DB2 to authenticate the xa_open call. If this user (for example, user mqm on UNIX systems) does not have minimal privileges in the database, the database refuses to authenticate the xa_open call.

The same considerations apply to your application process. If you have not specified a user name and password on the xa_open string, the user under which your application is running is used by DB2 to authenticate the xa_open call that is made during the first MQBEGIN. Again, this user must have minimal privileges in the database for this to work.

For example, give the mqm user connect authority in the mydbname database by issuing the following DB2 commands:

```
db2 connect to mydbname
db2 grant connect on database to user mqm
```

See “Security considerations” on page 183 for more information about security.

Change the tp_mon_name parameter

For DB2 for Windows systems only, change the TP_MON_NAME configuration parameter to name the DLL that DB2 uses to call the queue manager for dynamic registration.

DB2 configuration

Use the command `db2 update dbm cfg using TP_MON_NAME mqmax` to name MQMAX.DLL as the library that DB2 uses to call the queue manager. This must be present in a directory within PATH.

Reset the `maxappls` parameter

You might need to review your setting for the `maxappls` parameter, which limits the maximum number of applications that can be connected to a database. Refer to “Database connections” on page 170.

Oracle configuration

Do the following:

1. Check environment variable settings.
2. Create the Oracle switch load file.
3. Add resource manager configuration information.
4. Change the Oracle configuration parameters, if necessary.

A current list of levels of Oracle supported by WebSphere MQ is provided at: <http://www.ibm.com/software/ts/mqseries/platforms/supported.html>

Checking the Oracle environment variable settings

Ensure that your Oracle environment variables are set for queue manager processes *as well as in* your application processes. In particular, always set the following environment variables **before** starting the queue manager:

ORACLE_HOME

The Oracle home directory. For example, on UNIX systems, use:

```
export ORACLE_HOME=/opt/oracle/product/8.1.6
```

On Windows systems, use:

```
set ORACLE_HOME=c:\oracle\ora81
```

ORACLE_SID

The Oracle SID being used. If you are using Net8 for client/server connectivity, you might not need to set this environment variable. Consult your Oracle documentation.

An example of setting this, on UNIX systems, is:

```
export ORACLE_SID=sid1
```

The equivalent on Windows systems is:

```
set ORACLE_SID=sid1
```

Creating the Oracle switch load file

The easiest way to create the Oracle switch load file is to use the sample file `xaswit.mak`, which WebSphere MQ provides to build the switch load files for a variety of database products.

On Windows systems, you can find `xaswit.mak` in the directory `C:\Program Files\IBM\WebSphere MQ\tools\c\samples\xatm`. To create the Oracle switch load file with Microsoft Visual C++, use:

```
nmake /f xaswit.mak oraswit.dll
```

The generated switch file is placed in `c:\Program Files\IBM\WebSphere MQ\exits`.

On AIX, you can find `xaswit.mak` in the directory `/usr/mqm/samp/xatm`; on other UNIX systems, you can find it in the directory `/opt/mqm/samp/xatm`.

Edit `xaswit.mak` to *uncomment* the lines appropriate to the version of Oracle you are using. Then execute the makefile using the command:

```
make -f xaswit.mak oraswit
```

The generated 32-bit switch load file is placed in `/var/mqm/exits`.

The generated 64-bit switch load file is placed in `/var/mqm/exits64`.

Adding resource manager configuration information for Oracle

The next step is to modify the configuration information for the queue manager, as described in “Adding configuration information to the queue manager” on page 171, to declare Oracle as a participant in global units of work.

- On Windows and Linux (x86 platform) systems use the WebSphere MQ Explorer. Specify the details of the switch load file in the queue manager properties panel, under XA resource manager.
- On all other systems specify the details of the switch load file in the XAResourceManager stanza in the queue manager’s `qm.ini` file.

Figure 12 is a UNIX sample showing an XAResourceManager entry. It is recommend that you add a `LogDir` to the XA open string so that all error and tracing information is logged to the same place.

```
XAResourceManager:
  Name=myoracle
  SwitchFile=oraswit
  XAOpenString=Oracle_XA+Acc=P/myuser/mypasswd+SesTm=35+LogDir=/tmp+threads=true
  ThreadOfControl=THREAD
```

Figure 12. Sample XAResourceManager entry for Oracle on UNIX platforms

Notes:

1. In Figure 12, the `xa_open` string has been used with four parameters. Additional parameters can be included as described in Oracle’s documentation.
2. When using Oracle with WebSphere MQ V5.3 for AIX, you must add the `+SqlNet` clause to the `xa_open` string. WebSphere MQ V5.3 for AIX, uses AIX extended shared memory facilities and the `+SqlNet` client/server connectivity solution must be used to enable Oracle’s client library to reach the Oracle server. Additional text must be added to the Oracle `listener` and `tnsnames` files, see Oracle’s XA and Net9 documentation for details.
3. When using the WebSphere MQ parameter `ThreadOfControl=THREAD` you must use the Oracle parameter `+threads=true` in the XAResourceManager stanza.

See the *Oracle8 Server Application Developer’s Guide* for more information on the `xa_open` string.

Changing Oracle configuration parameters

For each Oracle database that the queue manager is coordinating, you need to:

Review your maximum sessions

You might need to review your `LICENSE_MAX_SESSIONS` and `PROCESSES` settings to take into account the additional connections required by processes belonging to the queue manager. Refer to “Database connections” on page 170 for more details.

Oracle configuration

Set database privileges

The Oracle user name specified in the `xa_open` string must have privileges to access the `DBA_PENDING_TRANSACTIONS` view, as described in the Oracle documentation.

The necessary privilege can be given using the following example command:

```
grant select on DBA_PENDING_TRANSACTIONS to myuser;
```

Informix configuration

Do the following:

1. Ensure that you have installed the appropriate Informix client SDK:
 - 32-bit queue managers and applications require a 32-bit Informix client SDK.
 - 64-bit queue managers and applications require a 64-bit Informix client SDK.
2. Ensure Informix databases are created correctly.
3. Check environment variable settings.
4. Build the Informix switch load file.
5. Add resource manager configuration information.

A current list of levels of Informix supported by WebSphere MQ is provided at:

<http://www.ibm.com/software/ts/mqseries/platforms/supported.html>

Ensuring Informix databases are created correctly

Every Informix database that is to be coordinated by a WebSphere MQ queue manager must be created specifying the `log` parameter. For example:

```
create database mydbname with log;
```

WebSphere MQ queue managers are unable to coordinate Informix databases that do not have the `log` parameter specified on creation. If a queue manager attempts to coordinate an Informix database that does not have the `log` parameter specified on creation, the `xa_open` call to Informix will fail, and a number of FFST errors will be generated.

Checking the Informix environment variable settings

Ensure that your Informix environment variables are set for queue manager processes *as well as in* your application processes. In particular, always set the following environment variables **before** starting the queue manager:

INFORMIXDIR

The directory of the Informix product installation.

- For 32-bit UNIX applications, use the following command:

```
export INFORMIXDIR=/opt/informix/32-bit
```
- For 64-bit UNIX applications, use the following command:

```
export INFORMIXDIR=/opt/informix/64-bit
```
- For Windows applications, use the following command:

```
set INFORMIXDIR=c:\informix
```

For systems that have 64-bit queue managers that must support both 32-bit and 64-bit applications, you need both the Informix 32-bit and 64-bit client SDKs installed. The sample make file `xaswit.mak`, used for creating a switch load file also sets both product installation directories.

INFORMIXSERVER

The name of the Informix server. For example, on UNIX systems, use:

```
export INFORMIXSERVER=hostname_1
```

On Windows systems, use:

```
set INFORMIXSERVER=hostname_1
```

ONCONFIG

The name of the Informix server configuration file. For example, on UNIX systems, use:

```
export ONCONFIG=onconfig.hostname_1
```

On Windows systems, use:

```
set ONCONFIG=onconfig.hostname_1
```

Creating the Informix switch load file

The easiest way to create the Informix switch load file is to use the sample file `xaswit.mak`, which WebSphere MQ provides to build the switch load files for a variety of database products.

On Windows systems, you can find `xaswit.mak` in the directory `C:\Program Files\IBM\WebSphere MQ\tools\c\samples\xatm`. To create the Informix switch load file with Microsoft Visual C++, use:

```
nmake /f xaswit.mak infswit.dll
```

The generated switch file is placed in `c:\Program Files\IBM\WebSphere MQ\exits`.

On AIX, you can find `xaswit.mak` in the directory `/usr/mqm/samp/xatm`; on other UNIX systems, you can find it in the directory `/opt/mqm/samp/xatm`.

Edit `xaswit.mak` to *uncomment* the lines appropriate to the version of Informix you are using. Then execute the makefile using the command:

```
make -f xaswit.mak infswit
```

The generated 32-bit switch load file is placed in `/var/mqm/exits`.

The generated 64-bit switch load file is placed in `/var/mqm/exits64`.

Adding resource manager configuration information for Informix

The next step is to modify the configuration information for the queue manager, as described in “Adding configuration information to the queue manager” on page 171, to declare Informix as a participant in global units of work.

- On Windows and Linux (x86 platform) systems use the WebSphere MQ Explorer. Specify the details of the switch load file in the queue manager properties panel, under XA resource manager.
- On all other systems specify the details of the switch load file in the XAResourceManager stanza in the queue manager’s `qm.ini` file.

Figure 13 on page 180 is a UNIX sample, showing an XAResourceManager entry where the database to be coordinated is called `mydbname`, this name being specified in the `XAOpenString`:

Informix configuration

```
XAResourceManager:  
  Name=myinformix  
  SwitchFile=infswit  
  XAOpenString=mydbname  
  ThreadOfControl=PROCESS
```

Figure 13. Sample XAResourceManager entry for Informix on UNIX platforms

Note: In Figure 13, ThreadOfControl is specified as PROCESS. This value is required for the queue manager to use the Informix XA switch-load file. However, also note that the supported levels of Informix do not allow multi-threaded XA applications.

Sybase configuration

Do the following:

1. Ensure you have installed the Sybase XA libraries, for example by installing the XA DTM option.
2. Check environment variable settings.
3. Enable Sybase XA support.
4. Create the Sybase switch load file.
5. Add resource manager configuration information.

A current list of levels of Sybase supported by WebSphere MQ is provided at: <http://www.ibm.com/software/ts/mqseries/platforms/supported.html>

Checking the Sybase environment variable settings

Ensure that your Sybase environment variables are set for queue manager processes *as well as in* your application processes. In particular, always set the following environment variables **before** starting the queue manager:

SYBASE

The location of the Sybase product installation. For example, on UNIX systems, use:

```
export SYBASE=/sybase
```

On Windows systems, use:

```
set SYBASE=c:\sybase
```

SYBASE_OCS

The directory under SYBASE where you have installed the Sybase client files. For example, on UNIX systems, use:

```
export SYBASE_OCS=OCS-12_0
```

On Windows systems, use:

```
set SYBASE_OCS=OCS-12_0
```

Enabling Sybase XA support

Within the Sybase XA configuration file ($\$SYBASE/\$SYBASE_OCS/xa_config$), define a Logical Resource Manager (LRM) for each connection to the Sybase server that is being updated.

An example of the contents of $\$SYBASE/\$SYBASE_OCS/xa_config$ is shown in Figure 14 on page 181.

```
# The first line must always be a comment

[xa]

LRM=lrmname
server=servername
```

Figure 14. Example contents of `$SYBASE/$SYBASE_OCS/xa_config`

Creating the Sybase switch load file

The easiest way to create the Sybase switch load file is to use the sample files supplied with WebSphere MQ.

On Windows systems, you can find `xaswit.mak` in the directory `C:\Program Files\IBM\WebSphere MQ\tools\c\samples\xatm`. To create the Sybase switch load file with Microsoft Visual C++, use:

```
nmake /f xaswit.mak sybswit.dll
```

The generated switch file is placed in `c:\Program Files\IBM\WebSphere MQ\exits`.

On AIX, you can find `xaswit.mak` in the directory `/usr/mqm/samp/xatm`; on other UNIX systems, you can find it in the directory `/opt/mqm/samp/xatm`.

Edit `xaswit.mak` to *uncomment* the lines appropriate to the version of Sybase you are using. Then execute the makefile using the command:

```
make -f xaswit.mak sybswit
```

The generated 32-bit switch load file is placed in `/var/mqm/exits`.

The generated 64-bit switch load file is placed in `/var/mqm/exits64`.

Adding resource manager configuration information for Sybase

The next step is to modify the configuration information for the queue manager, as described in “Adding configuration information to the queue manager” on page 171, to declare Sybase as a participant in global units of work.

- On Windows and Linux (x86 platform) systems use the WebSphere MQ Explorer. Specify the details of the switch load file in the queue manager properties panel, under XA resource manager.
- On all other systems specify the details of the switch load file in the `XAResourceManager` stanza in the queue manager’s `qm.ini` file.

Figure 15 shows a UNIX sample, which uses the database associated with the `lrmname` LRM definition in the Sybase XA configuration file, `$SYBASE/$SYBASE_OCS/xa_config`. Include a log file name if you want XA function calls to be logged:

```
XAResourceManager:
  Name=mysybase
  SwitchFile=sybswit
  XAOpenString=-Uuser -Ppassword -Nlrmname -L/tmp/sybase.log -Txa
  ThreadOfControl=THREAD
```

Figure 15. Sample `XAResourceManager` entry for Sybase on UNIX platforms

Using multi-threaded programs with Sybase

If you are using multi-threaded programs with WebSphere MQ global units of work incorporating updates to Sybase, you *must* use the value `THREAD` for the `ThreadOfControl` parameter as shown in Figure 15 on page 181.

Also ensure that you link your program (and the switch load file) with the threadsafe Sybase libraries (the `_r` versions).

Multiple database configurations

If you want to configure the queue manager so that updates to multiple databases can be included within global units of work, add an `XAResourceManager` stanza for each database.

If the databases are all managed by the same database manager, each stanza defines a separate database. Each stanza specifies the same `SwitchFile`, but the contents of the `XAOpenString` are different because it specifies the name of the database being updated. For example, the stanzas shown in Figure 16 configure the queue manager with the DB2 databases `MQBankDB` and `MQFeeDB` on UNIX systems.

```
XAResourceManager:  
  Name=DB2 MQBankDB  
  SwitchFile=db2swit  
  XAOpenString=MQBankDB  
  
XAResourceManager:  
  Name=DB2 MQFeeDB  
  SwitchFile=db2swit  
  XAOpenString=MQFeeDB
```

Figure 16. Sample `XAResourceManager` entries for multiple DB2 databases

If the databases to be updated are managed by different database managers, add an `XAResourceManager` stanza for each. In this case, each stanza specifies a different `SwitchFile`. For example, if `MQFeeDB` is managed by Oracle instead of DB2, use the following stanzas on UNIX systems:

```
XAResourceManager:  
  Name=DB2 MQBankDB  
  SwitchFile=db2swit  
  XAOpenString=MQBankDB  
  
XAResourceManager:  
  Name=Oracle MQFeeDB  
  SwitchFile=oraswit  
  XAOpenString=Oracle_XA+Acc=P/myuser/mypassword+SesTm=35+LogDir=/tmp/ora.log+DB=MQFeeDB
```

Figure 17. Sample `XAResourceManager` entries for a DB2 and Oracle database

In principle, there is no limit to the number of database instances that can be configured with a single queue manager.

Note: For information on support for including Informix databases in multiple database updates within global units of work, check the product readme file.

Security considerations

The following information is provided for guidance only. In all cases, refer to the documentation provided with the database manager to determine the security implications of running your database under the XA model.

An application process denotes the start of a global unit of work using the **MQBEGIN** verb. The first **MQBEGIN** call that an application issues connects to all participating databases by calling their client library code at the `xa_open` entry point. All the database managers provide a mechanism for supplying a user ID and password in their `XAOpenString`. This is the only time that authentication information flows.

Note that, on UNIX platforms, fastpath applications must run with an effective user ID of `mqm` while making MQI calls.

Administration tasks

In normal operations, only a minimal amount of administration is necessary after you have completed the configuration steps. The administration job is made easier because the queue manager 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 regains contact with a previously unavailable database automatically as new global units of work are started with **MQBEGIN**. It does this by calling the `xa_open` function in the database client library. If this `xa_open` call fails, **MQBEGIN** returns with a completion code of `MQCC_WARNING` and a reason code of `MQRC_PARTICIPANT_NOT_AVAILABLE`. You can retry the **MQBEGIN** call later.

Do not continue to attempt a global unit of work that involves updates to a database that has indicated failure during **MQBEGIN**. There will not be a connection to that database through which updates can be made. Your only options are to end the program, or to retry **MQBEGIN** periodically in the hope that the database might become available again.

Administration tasks

Alternatively, you can use the **rsvmqtrn** command to resolve explicitly all in-doubt units of work.

In-doubt units of work

A database might 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 server receives the outcome from the queue manager (commit or roll back), 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 can use facilities provided by the database manager to commit or roll back the database updates manually. In the *X/Open Distributed Transaction Processing: The XA Specification*, this is called making a *heuristic* decision. Use it only as a last resort because of the possibility of compromising data integrity; you might, for example, mistakenly roll back the database updates when all other participants have committed their updates.

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, you can use the **dspmqtrn** command to check the state of outstanding global units of work involving that database.

The **dspmqtrn** command displays only those units of work in which one or more participants are in doubt, awaiting the decision from the queue manager to commit or roll back the prepared updates.

For each of these global units of work, the state of each participant is displayed in the output from **dspmqtrn**. 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.

When the queue manager is restarted, it asks each database having an XAResourceManager stanza for a list of its in-doubt global units of work. If the database has not been restarted, or is otherwise unavailable, the queue manager cannot yet deliver to the database the final outcomes for those units of work. The outcome of the in-doubt units of work is delivered to the database at the first opportunity when the database is again available.

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 unit of work, 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 unit of work.

Figure 18 shows the result of issuing the following command:

```
dspmqrn -m MY_QMGR
```

```
AMQ7107: Resource manager 0 is MQSeries.
AMQ7107: Resource manager 1 is DB2 MQBankDB.
AMQ7107: Resource manager 2 is DB2 MQFeeDB.

AMQ7056: Transaction number 0,1.
      XID: formatID 5067085, gtrid_length 12, bqual_length 4
           gtrid [3291A506000201374657374]
           bqual [00000001]
AMQ7105: Resource manager 0 has committed.
AMQ7104: Resource manager 1 has prepared.
AMQ7104: Resource manager 2 has prepared.
```

Figure 18. Sample *dspmqrn* output

The output in Figure 18 shows that there are three resource managers associated with the queue manager. The first is resource manager 0, which is the queue manager itself. The other two resource manager instances are the MQBankDB and MQFeeDB DB2 databases.

The example shows only a single in-doubt unit of work. A message is issued for all three resource managers, which means that updates were made to the queue manager and both DB2 databases within the unit of work.

The updates made to the queue manager, resource manager 0, have been *committed*. The updates to the DB2 databases are in *prepared* state, which means that DB2 must have become unavailable before it was called to commit the updates to the *MQBankDB* and *MQFeeDB* databases.

The in-doubt unit of work has an external identifier called an XID (*transaction id*). This is a piece of data given to DB2 by the queue manager to identify its portion of the global unit of work.

Resolving outstanding units of work with the **rsvmqtrn** command

The output shown in Figure 18 shows a single in-doubt unit of work in which the commit decision has yet to be delivered to both DB2 databases.

To complete this unit of work, the queue manager and DB2 need to resynchronize when DB2 next becomes available. The queue manager uses the start of new units of work as an opportunity to regain contact with DB2. Alternatively, you can instruct the queue manager to resynchronize explicitly using the **rsvmqtrn** command.

Do this soon after DB2 has been restarted, so that any database locks associated with the in-doubt unit of work are released as quickly as possible. Use the **-a**

Administration tasks

option, which tells the queue manager to resolve all in-doubt units of work. In the following example, DB2 has restarted, so the queue manager can 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.

Changing configuration information

After the queue manager has successfully started to coordinate global units of work, do not change any of the resource manager configuration information.

If you need to change the configuration information you can do so at any time, but the changes do not take effect until after the queue manager has been restarted.

If you remove the resource manager configuration information for a database, you are effectively removing the ability for the queue manager to contact that database manager.

Never change the *Name* attribute in any of your resource manager configuration information. This attribute uniquely identifies that database manager instance to the queue manager. If you change this unique identifier, the queue manager assumes that the database 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 need to remove a database from your configuration permanently, ensure that the database is not in doubt before you restart the queue manager. Database products provide commands for listing

in-doubt transactions. If there are any in-doubt transactions, first allow the queue manager to resynchronize with the database. Do this by starting the queue manager. You can verify that resynchronization has taken place by using the **rsvmqtrn** command or the database's own command for viewing in-doubt units of work. Once you are satisfied that resynchronization has taken place, end the queue manager and remove the database's configuration information.

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, use the **-r** option of the **rsvmqtrn** command to instruct the queue manager to forget about the database's participation in its in-doubt transactions. The queue manager forgets about such transactions only when in-doubt transactions have been completed with all participants.

There are times when you might need to remove some resource manager configuration information temporarily. On UNIX systems this is best achieved by commenting out the stanza so that it can be easily reinstated at a later time. You might decide to do this if there are errors every time the queue manager contacts a particular database or database manager. Temporarily removing the resource manager configuration information concerned allows the queue manager to start global units of work involving all the other participants. Here is an example of a commented-out XAResourceManager stanza follows:

```
# This database has been temporarily removed
#XAResourceManager:
# Name=mydb2
# SwitchFile=db2swit
# XAOpenString=mydbname,myuser,mypassword,toc=t
# ThreadOfControl=THREAD
```

Figure 19. Commented- out XAResourceManager stanza on UNIX systems

On Windows systems, use the WebSphere MQExplorer to delete the information about the database manager instance. Take great care to type in the correct name in the *Name* field when reinstating it. If you mistype the name, you may face in-doubt problems, as described in "Changing configuration information" on page 186.

XA dynamic registration

The XA specification provides a way of reducing the number of `xa_*` calls that a transaction manager makes to a resource manager. This optimization is known as *dynamic registration*. Dynamic registration is supported by DB2. Other databases might support it; consult the documentation for your database product for details.

Why is the dynamic registration optimization useful? In your application, some global units of work might contain updates to database tables; others might not contain such updates. When no persistent update has been made to a database's tables, there is no need to include that database in the commit protocol that occurs during MQCMIT.

Whether or not your database supports dynamic registration, your application calls `xa_open` during the first MQBEGIN call on a WebSphere MQ connection. It also calls `xa_close` on the subsequent MQDISC call. The pattern of subsequent XA calls depends on whether the database supports dynamic registration:

Administration tasks

If your database does not support dynamic registration...

Every global unit of work involves several XA function calls made by WebSphere MQ code into the database client library, regardless of whether you made a persistent update to the tables of that database within your unit of work. These include:

- `xa_start` and `xa_end` from the application process. These are used to declare the beginning and end of a global unit of work.
- `xa_prepare`, `xa_commit`, and `xa_rollback` from the queue manager agent process, `amqzlaa0`. These are used to deliver the outcome of the global unit of work: the commit or rollback decision.

In addition, the queue manager agent process also calls `xa_open` during the first `MQBEGIN`.

If your database supports dynamic registration...

The WebSphere MQ code makes only those XA function calls that are necessary. For a global unit of work that has **not** involved persistent updates to database resources, there are **no** XA calls to the database. For a global unit of work that **has** involved such persistent updates, the calls are to:

- `xa_end` from the application process to declare the end of the global unit of work.
- `xa_prepare`, `xa_commit`, and `xa_rollback` from the queue manager agent process, `amqzlaa0`. These are used to deliver the outcome of the global unit of work: the commit or rollback decision.

For dynamic registration to work, it is vital that the database has a way of telling WebSphere MQ when it has performed a persistent update that it wants to be included in the current global unit of work. WebSphere MQ provides the `ax_reg` function for this purpose.

The database's client code that runs in your application process finds the `ax_reg` function and calls it, to *dynamically register* the fact it has done persistent work within the current global unit of work. In response to this `ax_reg` call, WebSphere MQ records that the database has participated. If this is the first `ax_reg` call on this WebSphere MQ connection, the queue manager agent process calls `xa_open`.

The database client code make this `ax_reg` call when it is running in your process, for example, during an SQL `UPDATE` call or whatever call in the database's client API is responsible

Error conditions

There is an opportunity here for a confusing failure in the queue manager. Here is a common example. If you forget to set your database environment variables properly before starting your queue manager, the queue manager's calls to `xa_open` fail. No global units of work can be used.

To avoid this, ensure that you have set the relevant environment variables before starting the queue manager. Review your database product's documentation, and the advice given in "Checking the DB2 environment variable settings" on page 174, "Checking the Oracle environment variable settings" on page 176, and "Checking the Sybase environment variable settings" on page 180.

With all database products, the queue manager calls `xa_open` once at queue manager startup, as part of the recovery session (as explained in "Administration tasks" on page 183). This `xa_open` call fails if you set your database environment

variables incorrectly, but it does not cause the queue manager to fail to start. This is because the same `xa_open` error code is used by the database client library to indicate that the database server is unavailable. We do not treat this as a serious error, as the queue manager must be able to start to continue processing data outside global units of work involving that database.

Subsequent calls to `xa_open` are made from the queue manager during the first `MQBEGIN` on a WebSphere MQ connection (if dynamic registration is not being used) or during a call by the database client code to the WebSphere MQ-provided `ax_reg` function (if dynamic registration is being used).

The **timing** of any error conditions (or, occasionally, FFST reports) depends on whether you are using dynamic registration:

- If you are using dynamic registration, your `MQBEGIN` call could succeed, but your `SQL UPDATE` (or similar) database call will fail.
- If you are not using dynamic registration, your `MQBEGIN` call will fail.

Ensure that your environment variables are set correctly in your application and queue manager processes.

Summarizing XA calls

Table 13 lists the calls that are made to the XA functions in a database client library as a result of the various MQI calls that control global units of work. This is not a complete description of the protocol described in the XA specification; we have provided it as a brief overview.

Note that `xa_start` and `xa_end` calls are always called by WebSphere MQ code in the application process, whereas `xa_prepare`, `xa_commit`, and `xa_rollback` are always called from the queue manager agent process, `amqzlaa0`.

The `xa_open` and `xa_close` calls shown in this table are all made from the application process. The queue manager agent process calls `xa_open` in the circumstances described in “Error conditions” on page 188.

Table 13. Summary of XA function calls

MQI call	XA calls made with dynamic registration	XA calls made without dynamic registration
First <code>MQBEGIN</code>	<code>xa_open</code>	<code>xa_open</code> <code>xa_start</code>
Subsequent <code>MQBEGIN</code>	No XA calls	<code>xa_start</code>
<code>MQCMIT</code> (without <code>ax_reg</code> being called during the current global unit of work)	No XA calls	<code>xa_end</code> <code>xa_prepare</code> <code>xa_commit</code> <code>xa_rollback</code>
<code>MQCMIT</code> (with <code>ax_reg</code> being called during the current global unit of work)	<code>xa_end</code> <code>xa_prepare</code> <code>xa_commit</code> <code>xa_rollback</code>	Not applicable. No calls are made to <code>ax_reg</code> in non-dynamic mode.
<code>MQBACK</code> (without <code>ax_reg</code> being called during the current global unit of work)	No XA calls	<code>xa_end</code> <code>xa_rollback</code>
<code>MQBACK</code> (with <code>ax_reg</code> being called during the current global unit of work)	<code>xa_end</code> <code>xa_rollback</code>	Not applicable. No calls are made to <code>ax_reg</code> in non-dynamic mode.

Administration tasks

Table 13. Summary of XA function calls (continued)

MQI call	XA calls made with dynamic registration	XA calls made without dynamic registration
MQDISC, where MQCMIT or MQBACK was called first. If they were not, MQCMIT processing is first done during MQDISC.	xa_close	xa_close
Notes: 1. For MQCMIT, xa_commit is called if xa_prepare is successful. Otherwise, xa_rollback is called.		

Scenario 2: Other software provides the coordination

In scenario 2, an external transaction manager coordinates global units of work, starting and committing them under control of the transaction manager's API. The MQBEGIN, MQCMIT, and MQBACK verbs are unavailable.

This section describes this scenario, including:

- "External syncpoint coordination"
- "Using CICS" on page 192
- "Using the Microsoft Transaction Server (COM+)" on page 194

External syncpoint coordination

A global unit of work can also be coordinated by an external X/Open XA-compliant transaction manager. Here the WebSphere MQ queue manager participates in, but does not coordinate, the unit of work.

The flow of control in a global unit of work coordinated by an external transaction manager is as follows:

1. An application tells the external syncpoint coordinator (for example, TXSeries) that it wants to start a transaction.
2. The syncpoint coordinator tells known resource managers, such as WebSphere MQ, about the current transaction.
3. The application issues calls to resource managers associated with the current transaction. For example, the application could issue **MQGET** calls to WebSphere MQ.
4. The application issues a commit or backout request to the external syncpoint coordinator.
5. The syncpoint coordinator completes the transaction by issuing the appropriate calls to each resource manager, typically using two-phase commit protocols.

The supported levels of external syncpoint coordinators that can provide a two-phase commit process for transactions in which WebSphere MQ participates are defined at:

<http://www.ibm.com/software/ts/mqseries/platforms/supported.html>

See the *WebSphere MQ Application Programming Guide* for information about writing and building transactions to be coordinated by an external syncpoint coordinator.

The rest of this chapter describes how to enable external units of work.

The WebSphere MQ XA switch structure

Each resource manager participating in an externally coordinated unit of work must provide an XA switch structure. This structure defines both the capabilities of the resource manager and the functions that are to be called by the syncpoint coordinator.

WebSphere MQ provides two versions of this structure:

- *MQRMIXASwitch* for static XA resource management
- *MQRMIXASwitchDynamic* for dynamic XA resource management

Consult your transaction manager documentation to determine whether to use the static or dynamic resource management interface. Wherever a transaction manager supports it, we recommend that you use dynamic XA resource management.

The following table details the XA switch load file names depending on platform:

Table 14. XA switch load file names

Platform	Switch load file name (server)	Switch load file name (extended transactional client)
Windows	mqmxa.dll	mqcxa.dll
AIX (nonthreaded)	libmqmxa.a	libmqcxa.a
AIX (threaded)	libmqmxa_r.a	libmqcxa_r.a
HP-UX (nonthreaded)	libmqmxa.sl	libmqcxa.sl
HP-UX (threaded)	libmqmxa_r.sl	libmqcxa_r.sl
Linux (nonthreaded)	libmqmxa.so	libmqcxa.so
Linux (thread)	libmqmxa_r.so	libmqcxa_r.so
Solaris	libmqmxa.so	libmqcxa.so

Some external syncpoint coordinators (not CICS) require that each resource manager participating in a unit of work supplies its name in the name field of the XA switch structure. The WebSphere MQ resource manager name is MQSeries_XA_RMI.

The syncpoint coordinator defines how the WebSphere MQ XA switch structure links to it. Information about linking the WebSphere MQ XA switch structure with CICS is provided in “Using CICS” on page 192. For information about linking the WebSphere MQ XA switch structure with other XA-compliant syncpoint coordinators, consult the documentation supplied with those products.

The following considerations apply to using WebSphere MQ with all XA-compliant syncpoint coordinators:

- The `xa_info` structure passed on any `xa_open` call by the syncpoint coordinator includes the name of a WebSphere MQ queue manager. The name takes the same form as the queue-manager name passed to the `MQCONN` call. If the name passed on the `xa_open` call is blank, the default queue manager is used.
- Only one queue manager at a time can participate in a transaction coordinated by an instance of an external syncpoint coordinator. The syncpoint coordinator is effectively connected to the queue manager, and is subject to the rule that only one connection at a time is supported.
- All applications that include calls to an external syncpoint coordinator can connect only to the queue manager that is participating in the transaction managed by the external coordinator (because they are already effectively

External syncpoint coordination

connected to that queue manager). However, such applications must issue an **MQCONN** call to obtain a connection handle, and an **MQDISC** call before they exit.

- A queue manager with resource updates coordinated by an external syncpoint coordinator must start before the external syncpoint coordinator. Similarly, the syncpoint coordinator must end before the queue manager.
- If your external syncpoint coordinator terminates abnormally, stop and restart your queue manager *before* restarting the syncpoint coordinator to ensure that any messaging operations uncommitted at the time of the failure are properly resolved.

Using CICS

CICS is one of the elements of TXSeries. The versions of TXSeries that are XA-compliant (and use a two-phase commit process) are defined at:

<http://www.ibm.com/software/ts/mqseries/platforms/supported.html>

The CICS two-phase commit process

This process applies to those versions of WebSphere MQ that support an XA-compliant external syncpoint coordinator, as defined at:

<http://www.ibm.com/software/ts/mqseries/platforms/supported.html>

Requirements of the two-phase commit process: When you use the CICS two-phase commit process with WebSphere MQ, note the following requirements:

- WebSphere MQ and CICS must reside on the same physical machine.
- WebSphere MQ does not support CICS on a WebSphere MQ client.
- You must start the queue manager, with its name specified in the XAD resource definition stanza, *before* you attempt to start CICS. Failure to do this will prevent you from starting CICS if you have added an XAD resource definition stanza for WebSphere MQ to the CICS region.
- Only one WebSphere MQ queue manager can be accessed at a time from a single CICS region.
- A CICS transaction must issue an **MQCONN** request before it can access WebSphere MQ resources. The **MQCONN** call must specify the name of the WebSphere MQ queue manager specified on the XAOpen entry of the XAD resource definition stanza for the CICS region. If this entry is blank, the **MQCONN** request must specify the default queue manager.
- A CICS transaction that accesses WebSphere MQ resources must issue an **MQDISC** call from the transaction before returning to CICS. Failure to do this might mean that the CICS application server is still connected, leaving queues open.
- You must ensure that the CICS user ID (cics) is a member of the mqm group, so that the CICS code has the authority to call WebSphere MQ.

For transactions running in a CICS environment, the queue manager adapts its methods of authorization and determining context as follows:

- The queue manager queries the user ID under which CICS runs the transaction. This is the user ID checked by the Object Authority Manager, and is used for context information.
- In the message context, the application type is MQAT_CICS.
- The application name in the context is copied from the CICS transaction name.

Enabling the CICS two-phase commit process: To enable CICS to use a two-phase commit process to coordinate transactions that include MQI calls, add a CICS XAD resource definition stanza entry to the CICS region.

Here is an example of adding an XAD stanza entry for WebSphere MQ for Windows, where <Drive> is the drive where WebSphere MQ is installed (for example, D:).

```
cicsadd -cxad -r<cics_region> \
    ResourceDescription="MQM XA Product Description" \
    SwitchLoadFile="<Drive>:\Program Files\IBM\WebSphere MQ\bin\mqmc4swi.dll" \
    XAOpen=<queue_manager_name>
```

For extended transactional clients, use the switch load file mqcc4swi.dll.

Here is an example of adding an XAD stanza entry for WebSphere MQ for UNIX systems:

```
cicsadd -cxad -r<cics_region> \
    ResourceDescription="MQM XA Product Description" \
    SwitchLoadFile="/opt/mqm/lib/amqzsc" \
    XAOpen=<queue_manager_name>
```

For extended transactional clients, use the switch load file amqzsc.

For information about using the **cicsadd** command, see the *CICS Administration Reference*, or *CICS Administration Guide* for your platform.

Calls to WebSphere MQ can be included in a CICS transaction, and the WebSphere MQ resources will be committed or rolled back as directed by CICS. This support is not available to client applications.

You *must* issue an **MQCONN** from your CICS transaction in order to access WebSphere MQ resources, followed by a corresponding **MQDISC** on exit.

Enabling CICS user exits: Before using a CICS user exit, read the *CICS Administration Guide* for your platform.

A CICS user exit *point* (normally referred to as a *user exit*) is a place in a CICS module at which CICS can transfer control to a program that you have written (a user exit *program*), and at which CICS can resume control when your exit program has finished its work.

One of the user exits supplied with CICS is the Task termination user exit (UE014015), invoked at normal and abnormal task termination (after any syncpoint has been taken).

WebSphere MQ supplies a CICS task termination exit in source and executable form:

Table 15. CICS task termination exits

WebSphere MQ for...	Source	Executable
Windows	amqzscgn.c	mqmc1415.dll
UNIX systems	amqzscgx.c	amqzscg

Using CICS

If you are currently using this exit, add the WebSphere MQ calls from the supplied exits to your current exits. Integrate the WebSphere MQ calls into your existing exits at the appropriate place in the program logic. See the comments in the sample source file for help with this.

If you are not currently using this exit, add a CICS PD program definition stanza entry to the CICS region.

Here is an example of adding a PD stanza entry for Windows:

```
cicsadd -cpd -r<cics_region> \  
    PathName="<Drive>:\Program Files\IBM\WebSphere MQ\bin\mqmc1415.dll" \  
    UserExitNumber=15
```

Here is an example of adding a PD stanza entry for UNIX systems:

```
cicsadd -cpd -r<cics_region> \  
    PathName="/opt/mqm/lib/amqzscg" \  
    UserExitNumber=15
```

Using the Microsoft Transaction Server (COM+)

COM+ (Microsoft Transaction Server) is designed to help users run business logic applications in a typical middle tier server. COM+ divides work up into *activities*, which are typically short independent chunks of business logic, such as *transfer funds from account A to account B*. COM+ relies heavily on object orientation and in particular on COM; loosely a COM+ activity is represented by a COM (business) object.

COM+ is an integrated part of the operating system. To use COM+ on Windows 2000 and Windows XP, you need Hotfix Q313582 (also known as COM+ Rollup Package 19.1).

COM+ provides three services to the business object administrator, removing much of the worry from the business object programmer:

- Transaction management
- Security
- Resource pooling

You usually use COM+ with front end code that is a COM client to the objects held within COM+, and back end services such as a database, with WebSphere MQ bridging between the COM+ business object and the back end.

The front end code can be a standalone program, or an Active Server Page (ASP) hosted by the Microsoft Internet Information Server (IIS). The front end code can reside on the same computer as COM+ and its business objects, with connection through COM. Alternatively, the front end code can reside on a different computer, with connection through DCOM. You can use different clients to access the same COM+ business object in different situations.

The back end code can reside on the same computer as COM+ and its business objects, or on a different computer with connection through any of the WebSphere MQ supported protocols.

For detailed information on using COM+, including how to configure it, and how to develop your applications and object code, read the *WebSphere MQ MTS Support* part of the WebSphere MQ Help System.

Chapter 12. 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 196
- “How the rules table is processed” on page 202
- “An example DLQ handler rules table” on page 204

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 `stdin`; you associate the rules table with **runmqdlq** by redirecting `stdin` 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

DLQ handler

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 357.

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.)

In WebSphere MQ for Windows, the source of **amqsdq** is supplied in the directory:

```
c:\Program Files\IBM\WebSphere MQ\tools\c\samples\dlq
```

and the compiled version is supplied in the directory:

```
c:\Program Files\IBM\WebSphere MQ\tools\c\samples\bin
```

In WebSphere MQ for UNIX systems, the source of **amqsdq** is supplied in the directory:

```
/opt/mqm/samp/dlq (/usr/mqm/samp/dlq on AIX)
```

and the compiled version is supplied in the directory:

```
/opt/mqm/samp/bin (/usr/mqm/samp/bin on AIX)
```

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.

DLQ handler

&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

DLQ handler

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.

Notes:

1. Matching rule patterns are sought only for messages on the DLQ that begin with an MQDLH. Messages that do not begin with an MQDLH are reported periodically as being in error, and remain on the DLQ indefinitely.
2. All pattern keywords can be allowed to default, such that a rule 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 *WebSphere MQ Messages*.) 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)
```

DLQ handler

Chapter 13. Supporting the Microsoft Cluster Service (MSCS)

This information applies to WebSphere MQ for Windows only.

The Microsoft Cluster Service (MSCS) enables you to connect servers into a *cluster*, giving higher availability of data and applications, and making it easier to manage the system. MSCS can automatically detect and recover from server or application failures.

MSCS supports *failover* of *virtual servers*, which correspond to applications, Web sites, print queues, or file shares (including their disk spindles, files, IP addresses, and so on).

Failover is the process by which MSCS detects a failure in an application on one computer in the cluster, and shuts down the disrupted application in an orderly manner, transfers its state data to the other computer, and re-initiates the application there.

This chapter introduces MSCS clusters and describes setting up MSCS support in the following sections:

- “Introducing MSCS clusters”
- “Setting up WebSphere MQ for MSCS clustering” on page 209

Then tells you how to configure WebSphere MQ for MSCS clustering, in the following sections:

- “Creating a queue manager for use with MSCS” on page 211
- “Moving a queue manager to MSCS storage” on page 212
- “Putting a queue manager under MSCS control” on page 213
- “Removing a queue manager from MSCS control” on page 215

And then gives some useful hints on using MSCS with WebSphere MQ, and details the WebSphere MQ MSCS support utility programs, in the following sections:

- “Hints and tips on using MSCS” on page 216
- “WebSphere MQ MSCS support utility programs” on page 219

Introducing MSCS clusters

Before we start to look at MSCS clusters, we need to distinguish between them and WebSphere MQ clusters:

WebSphere MQ clusters

are groups of two or more queue managers on one or more computers, providing automatic interconnection, and allowing queues to be shared amongst them for load balancing and redundancy.

MSCS clusters

are groups of two or more computers, connected together and configured in such a way that, if one fails, MSCS performs a *failover*, transferring the state data of applications from the failing computer to another computer in the cluster and reinitiating their operation there.

In the rest of this book, *clusters* means WebSphere MQ clusters. In this chapter, *clusters* **always** means MSCS clusters.

Let us start by looking at a two-machine cluster. A two-machine cluster comprises two computers (for example, A and B) that are jointly connected to a network for client access using a *virtual IP address*. They might also be connected to each other by one or more private networks. A and B share at least one disk for the server applications on each to use. There is also another shared disk, which must be a redundant array of independent disks (*RAID*) Level 1, for the exclusive use of MSCS; this is known as the *quorum* disk. MSCS monitors both computers to check that the hardware and software are running correctly.

In a simple setup such as this, both computers have all the applications installed on them, but only computer A runs with live applications; computer B is just running and waiting. If computer A encounters any one of a range of problems, MSCS shuts down the disrupted application in an orderly manner, transfers its state data to the other computer, and re-initiates the application there. This is known as a *failover*. Applications can be made *cluster-aware* so that they interact fully with MSCS and failover gracefully.

A typical setup for a two-computer cluster is as shown in Figure 20.

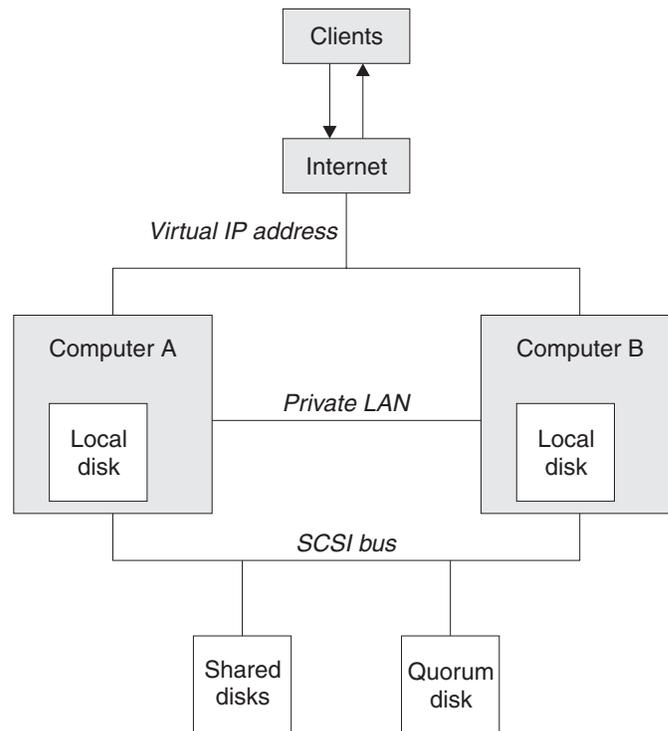


Figure 20. Two-computer MSCS cluster

Each computer can access the shared disk, but only one at a time, under the control of MSCS. In the event of a failover, MSCS switches the access to the other computer. The shared disk itself is usually a RAID, but need not be.

Each computer is connected to the external network for client access, and each has an IP address. However an external client, communicating with this cluster, sees only one *virtual IP address*, and MSCS routes the IP traffic within the cluster appropriately.

MSCS also performs its own communications between the two computers, either over one or more private connections or over the public network, in order to monitor their states using the heartbeat, synchronize their databases, and so on.

Setting up WebSphere MQ for MSCS clustering

You configure WebSphere MQ for clustering by making the queue manager the unit of failover to MSCS. You define a queue manager as a resource to MSCS, which can then monitor it, and transfer it to another computer in the cluster if there is a problem.

To set your system up for this, you start by installing WebSphere MQ on each computer in the cluster. *WebSphere MQ for Windows, V6.0 Quick Beginnings* tells you how to do this.

The queue managers themselves need to exist only on the computer on which you create them. In the event of a failover, the MSCS initiates the queue managers on the other computer. The queue managers, however, must have their log and data files on a cluster shared disk, and not on a local drive. If you have a queue manager already installed on a local drive, you can migrate it using a tool provided with WebSphere MQ; see “Moving a queue manager to MSCS storage” on page 212. If you want to create new queue managers for use with MSCS, see “Creating a queue manager for use with MSCS” on page 211.

After installation and migration, use the MSCS Cluster Administrator to make MSCS aware of your queue managers; see “Putting a queue manager under MSCS control” on page 213.

If you decide to remove a queue manager from MSCS control, use the procedure described in “Removing a queue manager from MSCS control” on page 215.

Setup symmetry

When an application switches from one node to the other it must behave in the same way, regardless of node. The best way of ensuring this is to make the environments identical. If you can, set up a cluster with identical hardware, operating system software, product software, and configuration on each computer. In particular, ensure that all the required software installed on the two computers is identical in terms of version, maintenance level, SupportPacs, paths and exits (as described *WebSphere MQ for Windows, V6.0 Quick Beginnings*), and that there is a common namespace (security environment) as described in “MSCS security.”

MSCS security

Start by making sure you have identical software installations on each computer in the cluster, as described in *WebSphere MQ for Windows, V6.0 Quick Beginnings*.

For successful MSCS security, follow these guidelines:

- Create a common namespace (security environment) across the cluster.
- Make the nodes of the MSCS cluster members of a domain, within which the user account that is the *cluster owner* is a domain account.
- Make the other user accounts on the cluster also domain accounts, so that they are available on both nodes. This is automatically the case if you already have a domain, and the accounts relevant to WebSphere MQ are domain accounts. If you do not currently have a domain, consider setting up a *mini-domain* to cater

for the cluster nodes and relevant accounts. Your aim is to make your cluster of two computers look like a single computing resource.

Remember that an account that is local to one computer does not exist on the other one. Even if you create an account with the same name on the other computer, its security identifier (SID) is different, so, when your application is moved to the other node, the permissions do not exist on that node.

During a failover or move, WebSphere MQ MSCS support ensures that all files that contain queue manager objects have equivalent permissions on the destination node. Explicitly, the code checks that the Administrators and mqm groups, and the SYSTEM account, have full control, and that if Everyone had read access on the old node, that permission is added on the destination node.

You can use a domain account to run your WebSphere MQ Service. Make sure that it exists in the local mqm group on each computer in the cluster.

Using multiple queue managers with MSCS

If you are running more than one queue manager on a computer, you can choose one of the following setups:

- All the queue managers in a single group. In this configuration, if a problem occurs with any queue manager, all the queue managers in the group failover to the other computer as a group.
- A single queue manager in each group. In this configuration, if a problem occurs with the queue manager, it alone fails over to the other computer without affecting the other queue managers.
- A mixture of the first two setups.

Cluster modes

There are two modes in which you might run a cluster system with WebSphere MQ:

- Active/Passive
- Active/Active

Note: If you are using MSCS together with the Microsoft Transaction Server (COM+), you cannot use Active/Active mode.

Active/Passive mode

In Active/Passive mode, computer A has the running application on it, and computer B is backup, only being used when MSCS detects a problem.

You can use this mode with only one shared disk, but, if any application causes a failover, **all** the applications must be transferred as a group (because only one computer can access the shared disk at a time).

You can configure MSCS with A as the *preferred* computer. Then, when computer A has been repaired or replaced and is working properly again, MSCS detects this and automatically switches the application back to computer A.

If you run more than one queue manager, consider having a separate shared disk for each. Then put each queue manager in a separate group in MSCS. In this way, any queue manager can failover to the other computer without affecting the other queue managers.

Active/Active mode

In Active/Active mode, computers A and B both have running applications, and the groups on each computer are set to use the other computer as backup. If a failure is detected on computer A, MSCS transfers the state data to computer B, and reinitiates the application there. Computer B then runs its own application and A's.

For this setup you need at least two shared disks. You can configure MSCS with A as the preferred computer for A's applications, and B as the preferred computer for B's applications. After failover and repair, each application automatically ends up back on its own computer.

For WebSphere MQ this means that you could, for example, run two queue managers, one on each of A and B, with each exploiting the full power of its own computer. After a failure on computer A, both queue managers would run on computer B. This would mean sharing the power of the one computer, with a reduced ability to process large quantities of data at speed. However, your critical applications would still be available while you find and repair the fault on A.

Creating a queue manager for use with MSCS

This procedure ensures that a new queue manager is created in such a way that it is suitable for preparing and placing under MSCS control.

You start by creating the queue manager with all its resources on a local drive, and then migrate the log files and data files to a shared disk. (You can reverse this operation.) Do **not** attempt to create a queue manager with its resources on a shared drive.

You can create a queue manager for use with MSCS in two ways, either from a command prompt, or in the WebSphere MQ Explorer. The advantage of doing using a command prompt is that the queue manager is created *stopped* and set to *manual startup*, which is ready for MSCS. (The WebSphere MQ Explorer automatically starts a new queue manager and sets it to automatic startup after creation. You have to change this.)

Creating a queue manager from a command prompt

1. Ensure that you have the environment variable MQS_PREFIX set to refer to a local drive, for example C:\WebSphere MQ. If you change this, reboot the machine so that the System account picks up the change. If you do not set the variable, the queue manager is created in the WebSphere MQ default directory for queue managers.
2. Create the queue manager using the **crtmqm** command. For example, to create a queue manager called `mcs_test` in the default directory, use:

```
crtmqm mcs_test
```
3. Proceed to "Moving a queue manager to MSCS storage" on page 212.

Creating a queue manager using the WebSphere MQ Explorer

1. Start the WebSphere MQ Explorer from the Start menu.
2. In the Navigator View, expand the tree nodes to find the **Queue Managers** tree node.
3. Right-click the **Queue Managers** tree node, and select **New->Queue Manager**. The Create Queue Manager panel is displayed.

4. Complete the dialog (Step 1), then click **Next>**.
5. Complete the dialog (Step 2), then click **Next>**.
6. Complete the dialog (Step 3), ensuring that **Start Queue Manager** and **Create Server Connection Channel** are not selected, then click **Next>**.
7. Complete the dialog (Step 4), then click **Finish**.
8. Proceed to "Moving a queue manager to MSCS storage."

Moving a queue manager to MSCS storage

This procedure configures an existing queue manager to make it suitable for putting under MSCS control. To achieve this, you move the log files and data files to shared disks to make them available to the other computer in the event of a failure. For example, the existing queue manager might have paths such as C:\WebSphere MQ\log\<QMname> and C:\WebSphere MQ\qmgrs\<QMname>. Do *not* try to move the files by hand; use the utility program supplied as part of WebSphere MQ MSCS Support as described below.

If the queue manager being moved uses SSL connections and the SSL key repository is in the queue manager data directory on the local machine, then the key repository will be moved with the rest of the queue manager to the shared disk. By default, the queue manager attribute that specifies the SSL key repository location, SSLKEYR, is set to <mqmtop>\qmgrs\QMGRNAME\ssl\key, which is under the queue manager data directory. The hamvmqm command does not modify this queue manager attribute. In this situation you must modify the queue manager attribute, SSLKEYR, using the WebSphere MQ Explorer or the MQSC command ALTER QMGR, to point to the new SSL key repository file.

The procedure is:

1. Shut down the queue manager, and check that there are no errors.
2. If the queue manager's log files or queue files are already stored on a shared disk, skip the rest of this procedure and proceed directly to "Putting a queue manager under MSCS control" on page 213.
3. Make a full media backup of the queue files and log files and store the backup in a safe place (see "Queue manager log files" on page 218 for why this is important).
4. If you already have a suitable shared disk resource proceed to step 6. Otherwise, using the MSCS Cluster Administrator to create a resource of type *shared disk* with sufficient capacity to store the queue manager log files and data (queue) files.
5. Test the shared disk by using the MSCS Cluster Administrator to move it from one cluster node to the other and back again.
6. Make sure that the shared disk is online on the cluster node where the queue manager log and data files are stored locally.
7. Run the utility program to move the queue manager as follows:

```
hamvmqm /m qmname /dd "e:\WebSphere MQ" /ld e:\WebSphere MQ\log"
```

substituting your queue manager name for *qmname*, your shared disk drive letter for *e*, and your chosen directory for *WebSphere MQ*. The directories are created if they do not already exist.

8. Test the queue manager to ensure that it works, using the WebSphere MQ Explorer. For example:

- a. Right-click the queue manager tree node, then select **Start**. The queue manager starts.
 - b. Right-click the **Queues** tree node, then select **New->Local Queue...**, and give the queue a name.
 - c. Click **Finish**.
 - d. Right-click the queue, then select **Put Test Message...**. The Put Test Message panel is displayed.
 - e. Type some message text, then click **Put Test Message**, and close the panel.
 - f. Right-click the queue, then select **Browse Messages...**. The Message Browser panel is displayed.
 - g. Ensure your message is on the queue, then click **Close**. The Message Browser panel closes.
 - h. Right-click the queue, then select **Clear Messages...**. The messages on the queue are cleared.
 - i. Right-click the queue, then select **Delete...**. A confirmation panel is displayed, click **OK**. The queue is deleted.
 - j. Right-click the queue manager tree node, then select **Stop...**. The End Queue Manager panel is displayed.
 - k. Click **OK**. The queue manager stops.
9. As WebSphere MQ Administrator ensure that the startup attribute of the queue manager is set to manual. In the WebSphere MQ Explorer, set the Startup field to manual in the queue manager properties panel.
 10. Proceed to "Putting a queue manager under MSCS control."

Putting a queue manager under MSCS control

Before you put a queue manager under MSCS control:

1. Ensure that WebSphere MQ and its MSCS Support is installed on both machines in the cluster and that the software on each computer is identical, as described in "Setting up WebSphere MQ for MSCS clustering" on page 209.
2. If you have not yet created the queue manager, see "Creating a queue manager for use with MSCS" on page 211.
3. If you have created the queue manager, or it already exists, ensure that you have carried out the procedure in "Moving a queue manager to MSCS storage" on page 212.
4. Stop the queue manager, if it is running, using either a command prompt or the WebSphere MQ Explorer.
5. Test MSCS operation of the shared drives before going on to the procedure below.

To place a queue manager under MSCS control:

1. Log in to the cluster node computer hosting the queue manager, or log in to a remote workstation as a user with cluster administration permissions, and connect to the cluster node hosting the queue manager.
2. Start the MSCS Cluster Administrator.
3. Open a connection to the cluster.
4. Create an MSCS group to be used to contain the resources for the queue manager. Name the group in such a way that it is obvious which queue manager it relates to. Each group can contain multiple queue managers, as described in "Using multiple queue managers with MSCS" on page 210. Use the group for all the remaining steps.

5. Create a resource instance for each of the SCSI logical drives that the queue manager uses.

You can use one drive to store both the logs and queue files, or you can split them up across drives. In either case, if each queue manager has its own shared disk, ensure that all drives used by this queue manager are exclusive to this queue manager, that is, that nothing else relies on the drives. Also ensure that you create a resource instance for every drive that the queue manager uses.

The resource type for a drive depends on the SCSI support you are using; refer to your SCSI adapter instructions. There might already be groups and resources for each of the shared drives. If so, you do not need to create the resource instance for each drive. Just move it from its current group to the one created for the queue manager.

For each drive resource, set possible owners to both nodes. Set dependent resources to none.

6. Create a resource instance for the IP address.

Create an IP address resource (resource type *IP Address*). This address should be an unused IP address to be used by clients and other queue managers to connect to the *virtual* queue manager. This IP address is not the normal (static) address of either node; it is an additional address that *floats* between them. Although MSCS handles the routing of this address, it does **not** verify that the address can be reached.

7. Create a resource instance for the queue manager.

Create a resource of type *IBM WebSphere MQ MSCS*; the wizard prompts you for various items, including the following:

- Name; choose a name that makes it easy to identify which queue manager it is for.
- Add to group; use the group that you created
- Run in a separate Resource Monitor; for better isolation
- Possible owners; set both nodes
- Dependencies; add the drive and IP address for this queue manager
- Parameters; as follows:
 - QueueManagerName (required); the name of the queue manager that this resource is to control. This queue manager must exist on the local computer.
 - PostOnlineCommand (optional); you can specify a program to run whenever the queue manager resource changes its state from offline to online. For more details see “PostOnlineCommand and PreOfflineCommand” on page 219.
 - PreOfflineCommand (optional); you can specify a program to run whenever the queue manager resource changes its state from online to offline. For more details see “PostOnlineCommand and PreOfflineCommand” on page 219.

8. Optionally, set a preferred node (but note the comments in “Using preferred nodes” on page 219).
9. The *Failover Policy* (as defined in the properties for the group) is set by default to sensible values, but you can tune the thresholds and periods that control *Resource Failover* and *Group Failover* to match the loads placed on the queue manager.

10. Test the queue manager by bringing it online in the MSCS Cluster Administrator and subjecting it to a test workload. If you are experimenting with a test queue manager, use the WebSphere MQ Explorer. For example:
 - a. Right-click the **Queues** tree node, then select **New->Local Queue...**, and give the queue a name.
 - b. Click **Finish**. The queue is created, and displayed in the content view.
 - c. Right-click the queue, then select **Put Test Message...**. The Put Test Message panel is displayed.
 - d. Type some message text, then click **Put Test Message**, and close the panel.
 - e. Right-click the queue, then select **Browse Messages...**. The Message Browser panel is displayed.
 - f. Ensure your message is on the queue, then click **Close**. The Message Browser panel closes.
 - g. Right-click the queue, then select **Clear Messages...**. The messages on the queue are cleared.
 - h. Right-click the queue, then select **Delete...**. A confirmation panel is displayed, click **OK**. The queue is deleted.
11. Test that the queue manager can be taken offline and back online using the MSCS Cluster Administrator.
12. Simulate a failover.

In the MSCS Cluster Administrator, right-click the group containing the queue manager and select **Move Group**. This can take some minutes to do. (If at other times you just want to move a queue manager to another node quickly, follow the procedure in “Moving a queue manager to MSCS storage” on page 212.) You can also right-click and select **Initiate Failure**; the action (local restart or failover) depends on the current state and the configuration settings.

Removing a queue manager from MSCS control

You can remove queue managers from MSCS control, and return them to manual administration. You do not need to do this for maintenance operations. You can do that by taking a queue manager offline temporarily, using the MSCS Cluster Administrator. Removing a queue manager from MSCS control is a more permanent change; only do it if you decide that you no longer want MSCS to have any further control of the queue manager.

If the queue manager being removed uses SSL connections you must modify the queue manager attribute, `SSLKEYR`, using the WebSphere MQ Explorer or the MQSC command `ALTER QMGR`, to point to the SSL key repository file on the local directory.

The procedure is:

1. Take the queue manager resource offline using the MSCS Cluster Administrator. To do this, see “Taking a queue manager offline from MSCS” on page 216.
2. Destroy the resource instance. This does not destroy the queue manager.
3. Optionally, migrate the queue manager files back from shared drives to local drives. To do this, see “Returning a queue manager from MSCS storage” on page 216.
4. Test the queue manager.

Taking a queue manager offline from MSCS

The procedure is:

1. Start the MSCS Cluster Administrator.
2. Open a connection to the cluster.
3. Select Groups, and open the group containing the queue manager to be moved.
4. Select the queue manager resource.
5. Right-click it and select *Offline*.
6. Wait for completion.

Returning a queue manager from MSCS storage

This procedure configures the queue manager to be back on its computer's local drive, that is, it becomes a *normal* WebSphere MQ queue manager. To achieve this, you move the log files and data files from the shared disks. For example, the existing queue manager might have paths such as E:\WebSphere MQ\log\<QMname> and E:\WebSphere MQ\qmgrs\<QMname>. Do not try to move the files by hand; use the **hamvmqm** utility program supplied as part of WebSphere MQ MSCS Support as described below:

1. Shut down the queue manager, and check that there are no errors.
2. Make a full media backup of the queue files and log files and store the backup in a safe place (see "Queue manager log files" on page 218 for why this is important).
3. Decide which local drive to use and ensure that it has sufficient capacity to store the queue manager log files and data (queue) files.
4. Make sure that the shared disk on which the files currently reside is online on the cluster node to which to move the queue manager log and data files.
5. Run the utility program to move the queue manager as follows:

```
hamvmqm /m qmname /dd "c:\WebSphere MQ" /ld "c:\WebSphere MQ\log"
```

substituting your queue manager name for *qmname*, your local disk drive letter for *c*, and your chosen directory for *WebSphere MQ* (the directories are created if they do not already exist).

6. Test the queue manager to ensure that it works (as described in "Moving a queue manager to MSCS storage" on page 212).

Hints and tips on using MSCS

This section contains some general information to help you use WebSphere MQ support for MSCS effectively.

Verifying that MSCS is working

The task descriptions starting with "Creating a queue manager for use with MSCS" on page 211 assume that you have a running MSCS cluster within which you can create, migrate, and destroy resources. If you want to make sure that you have such a cluster:

1. Using the MSCS Cluster Administrator, create a group.
2. Within that group, create an instance of a generic application resource, specifying the system clock (pathname C:\winnt\system32\clock.exe and working directory of C:\).
3. Make sure that you can bring the resource online, that you can move the group that contains it to the other node, and that you can take the resource offline.

Using the IBM MQSeries Service

Use the IBM MQSeries Service to monitor and control queue managers, running it on both machines in the cluster. The service has its own administration interface. See “MSCS security” on page 209 for information about user account options.

Once you have your queue manager online in MSCS, if you stop the IBM MQSeries Service for any reason, it automatically stops the queue manager. MSCS sees this as a failure condition.

Manual startup

For a queue manager managed by MSCS, you *must* set the startup attribute to manual. This ensures that the WebSphere MQ MSCS support can restart the IBM MQSeries Service without immediately starting the queue manager.

The WebSphere MQ MSCS support needs to be able to restart the service so that it can perform monitoring and control, but must itself remain in control of which queue managers are running, and on which machines. See “Moving a queue manager to MSCS storage” on page 212 for more information.

MSCS and queue managers

This section describes some things to consider about your queue managers when using MSCS, as follows:

- “Creating a matching queue manager on the other node”
- “Default queue managers”
- “Deleting a queue manager”
- “Support for existing queue managers”
- “Telling MSCS which queue managers to manage” on page 218
- “Queue manager log files” on page 218
- “Multiple queue managers” on page 218

Creating a matching queue manager on the other node

For clustering to work with WebSphere MQ, you need an identical queue manager on node B for each one on node A. However, you do not need to explicitly create the second one. You can create or prepare a queue manager on one node, move it to the other node as described in “Moving a queue manager to MSCS storage” on page 212, and it is fully duplicated on that node.

Default queue managers

Do not use a default queue manager under MSCS control. A queue manager does not have a property that makes it the default; WebSphere MQ keeps its own separate record. If you move a queue manager set to be the default to the other computer on failover, it does not become the default there. Make all your applications refer to specific queue managers by name.

Deleting a queue manager

Once a queue manager has moved node, its details exist in the registry on both computers. When you want to delete it, do so as normal on one computer, and then run the utility described in “WebSphere MQ MSCS support utility programs” on page 219 to clean up the registry on the other computer.

Support for existing queue managers

You can put an existing queue manager under MSCS control, provided that you can put your queue manager log files and queue files on a disk that is on the

shared SCSI bus between the two machines (see Figure 20 on page 208). You need to take the queue manager offline briefly while the MSCS Resource is created.

If you want to create a new queue manager, create it independently of MSCS, test it, then put it under MSCS control. See:

- “Creating a queue manager for use with MSCS” on page 211
- “Moving a queue manager to MSCS storage” on page 212
- “Putting a queue manager under MSCS control” on page 213

Telling MSCS which queue managers to manage

You choose which queue managers are placed under MSCS control by using the MSCS Cluster Administrator to create a resource instance for each such queue manager. This process presents you with a list of resources from which to select the queue manager that you want that instance to manage.

Queue manager log files

When you move a queue manager to MSCS storage, you move its log and data files to a shared disk (for an example see “Moving a queue manager to MSCS storage” on page 212).

It is advisable before you move, to shut the queue manager cleanly and take a full backup of the data files and log files.

Multiple queue managers

WebSphere MQ MSCS support allows you to run multiple queue managers on each machine and to place individual queue managers under MSCS control.

Always use MSCS to manage clusters

Do not try to perform start and stop operations directly on any clustered queue manager using either the WebSphere MQ Explorer. Instead, use the MSCS Cluster Administrator to request that MSCS brings the queue manager online or takes it offline. This is partly to prevent possible confusion caused by MSCS reporting that the queue manager is offline, when in fact you have started it outside the control of MSCS. More seriously, stopping a queue manager without using MSCS is detected by MSCS as a failure, initiating failover to the other node.

Working in Active/Active mode

Both computers in the MSCS cluster can run queue managers in Active/Active mode. You do not need to have a completely idle machine acting as standby (but you can, if you want, in Active/Passive Mode). If you plan to use both machines to run workload, provide each with sufficient capacity (processor, memory, secondary storage) to run the entire cluster workload at a satisfactory level of performance.

Note: If you are using MSCS together with Microsoft Transaction Server (COM+), you **cannot** use Active/Active mode. This is because, to use WebSphere MQ with MSCS and COM+:

- Application components that use WebSphere MQ’s COM+ support must run on the same computer as the Distributed Transaction Coordinator (DTC), a part of COM+.
- The queue manager must also run on the same computer.
- The DTC must be configured as an MSCS resource, and can therefore run on only one of the computers in the cluster at any time.

PostOnlineCommand and PreOfflineCommand

Specify these commands in the Parameters to a resource of type IBM WebSphere MQ MSCS. You can use them to integrate WebSphere MQ MSCS support with other systems or procedures. For example, you could specify the name of a program that sends a mail message, activates a pager, or generates some other form of alert to be captured by another monitoring system.

PostOnlineCommand is invoked when the resource changes from offline to online; PreOfflineCommand is invoked for a change from online to offline. Both commands run under the user account used to run the MSCS Cluster Service; and are invoked asynchronously; WebSphere MQ MSCS support does not wait for them to complete before continuing. This eliminates any risk that they might block or delay further cluster operations.

You can also use these commands to issue WebSphere MQ commands, for example to restart Requester channels. However, because they are *edge-triggered*, the commands are not suited to performing long-running functions, and cannot make assumptions about the current state of the queue manager. For example, the commands cannot assume that the queue manager is online; it is quite possible that, immediately after the queue manager was brought online, an administrator issued an offline command.

If you want to run programs that depend on the state of the queue manager, consider creating instances of the MSCS Generic Application resource type, placing them in the same MSCS group as the queue manager resource, and making them dependent on the queue manager resource.

Using preferred nodes

It can be useful when using Active/Active mode to configure a *preferred node* for each queue manager. However, in general it is better not to set a preferred node but to rely on a manual failback. Unlike some other relatively stateless resources, a queue manager can take a while to fail over (or back) from one node to the other. To avoid unnecessary outages, test the recovered node before failing a queue manager back to it. This precludes use of the *immediate* failback setting. You can configure failback to occur between certain times of day.

Probably the safest route is to move the queue manager back manually to the desired node, when you are certain that the node is fully recovered. This precludes use of the preferred node option.

Performance benchmarking

How long does it take to fail a queue manager over from one machine to the other? This depends heavily on the amount of workload on the queue manager and on the mix of traffic, that is, how much of it is persistent, within syncpoint, how much committed before the failure, and so on. In our test we have seen failover and failback times of about a minute. This was on a very lightly loaded queue manager and actual times will vary considerably depending on load.

WebSphere MQ MSCS support utility programs

WebSphere MQ support for MSCS includes the following utility programs that you can run at a command prompt:

Register/unregister the resource type
haregtyp.exe

After you *unregister* the WebSphere MQ MSCS resource type you can no longer create any resources of that type. MSCS does not let you unregister a resource type if you still have instances of that type within the cluster:

1. Using the MSCS Cluster Administrator, stop any queue managers that are running under MSCS control, by taking them offline as described in “Taking a queue manager offline from MSCS” on page 216.
2. Using the MSCS Cluster Administrator, delete the resource instances.
3. At a command prompt, unregister the resource type by entering the following command:

```
haregtyp /u
```

If you want to *register* the type (or re-register it at a later time), enter the following command at a command prompt:

```
haregtyp /r
```

After successfully registering the MSCS libraries, you must reboot the system if you have not done so since installing WebSphere MQ.

Move a queue manager to MSCS storage

```
hamvmqm.exe
```

See “Moving a queue manager to MSCS storage” on page 212.

Delete a queue manager from a node

```
hadltnqm.exe
```

Consider the case where you have had a queue manager in your cluster, it has been moved from one node to another, and now you want to destroy it. Use the WebSphere MQ Explorer to delete it on the node where it currently is. The registry entries for it still exist on the other computer. To delete these, enter the following command at a prompt on that computer:

```
hadltnqm /m qmname
```

where *qmname* is the name of the queue manager to remove.

Check and save setup details

```
amqmsysn.exe
```

This utility presents a dialog showing full details of your WebSphere MQ MSCS Support setup, such as might be requested if you call IBM support. There is an option to save the details to a file.

Part 5. Recovery and problem determination

Chapter 14. Recovery and restart	223
Making sure that messages are not lost (logging)	223
What logs look like	223
The log control file	224
Types of logging	224
Circular logging	224
Linear logging	225
Using checkpointing to ensure complete recovery	226
Checkpointing with long-running transactions	227
Calculating the size of the log	228
Managing logs	229
What happens when a disk gets full.	230
Managing log files.	230
Log file location	232
Using the log for recovery	232
Recovering from power loss or communications failures	232
Recovering damaged objects	232
Media recovery.	232
Recovering from media images	233
Recovering damaged objects during start up	234
Recovering damaged objects at other times	234
Protecting WebSphere MQ log files	234
Backing up and restoring WebSphere MQ	235
Backing up queue manager data	235
Restoring queue manager data	236
Using a backup queue manager	236
Creating a backup queue manager	237
Updating a backup queue manager	237
Starting a backup queue manager	238
Recovery scenarios	239
Disk drive failures.	239
Damaged queue manager object	240
Damaged single object	240
Automatic media recovery failure	240
Dumping the contents of the log using the dmpmqlog command.	240
Chapter 15. Problem determination	245
Preliminary checks	245
Has WebSphere MQ run successfully before?	245
Are there any error messages?.	246
Are there any return codes explaining the problem?.	246
Can you reproduce the problem?.	246
Have any changes been made since the last successful run?.	246
Has the application run successfully before?	246
If the application has not run successfully before	247
Common programming errors.	247
Problems with commands	248
Does the problem affect specific parts of the network?.	248
Does the problem occur at specific times of the day?	248
Is the problem intermittent?	248
Have you applied any service updates?	248
Looking at problems in more detail	249
Have you obtained incorrect output?	249
Messages that do not appear on the queue	249
Messages that contain unexpected or corrupted information	250
Problems with incorrect output when using distributed queues.	251
Have you failed to receive a response from a PCF command?	251
Are some of your queues failing?.	252
Does the problem affect only remote queues?	253
Is your application or system running slowly?	253
Tuning performance for nonpersistent messages on AIX	254
Application design considerations	254
Effect of message length.	254
Effect of message persistence	255
Searching for a particular message	255
Queues that contain messages of different lengths	255
Frequency of syncpoints.	255
Use of the MQPUT1 call.	255
Number of threads in use	255
Error logs	255
Error log files	256
Early errors	257
An example of an error log.	257
Error log access restrictions under UNIX systems	257
Ignoring error codes under UNIX systems.	258
Ignoring error codes under Windows systems	258
Operator messages	258
Dead-letter queues	258
Configuration files and problem determination	259
Tracing	259
Tracing WebSphere MQ for Windows	259
Selective component tracing on WebSphere MQ for Windows	259
Trace files	259
An example of WebSphere MQ for Windows trace data.	260
Tracing WebSphere MQ for UNIX systems.	260
Selective component tracing on WebSphere MQ for UNIX systems	261
Example trace data for WebSphere MQ for UNIX systems	261
Trace files	264
Tracing Secure Sockets Layer (SSL) on UNIX systems	265
Tracing with the AIX system trace	265
Selective component tracing on WebSphere MQ for AIX	266
An example of WebSphere MQ for AIX trace data	267

First-failure support technology (FFST)	267
FFST: WebSphere MQ for Windows	267
FFST: WebSphere MQ for UNIX systems	269
Problem determination with WebSphere MQ clients	271
Terminating clients	271
Java diagnostics	271
Using com.ibm.mq.commonservices	271
Java trace and FFDC files	273

Chapter 14. 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)”
- “Using checkpointing to ensure complete recovery” on page 226
- “Calculating the size of the log” on page 228
- “Managing logs” on page 229
- “Using the log for recovery” on page 232
- “Protecting WebSphere MQ log files” on page 234
- “Backing up and restoring WebSphere MQ” on page 235
- “Recovery scenarios” on page 239
- “Dumping the contents of the log using the `dmpmqlog` command” on page 240

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

Logging

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 9, “Configuring WebSphere MQ,” on page 103), 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 Windows, if you have not changed the log path, log files are created under the directory:

```
C:\Program Files\IBM\WebSphere MQ\log\<QMgrName>
```

In WebSphere MQ for UNIX systems, if you have not changed the log path, log files are created under the directory:

```
/var/mqm/log/<QMgrName>
```

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 9, “Configuring WebSphere MQ,” on page 103.

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 will handle. You will probably 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 228.

Types of logging

In WebSphere MQ, the number of files that are required for logging depends on the file size, the number of messages you have received, and the length of the messages. There are two ways of maintaining records of queue manager activities: circular logging and linear logging.

Circular logging

Use circular logging if all you want is restart recovery, using the log to roll back transactions that were in progress when the system stopped.

Circular logging keeps all restart data in a ring of log files. Logging fills the first file in the ring, then moves on to the next, and so on, until all the files are full. 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.

WebSphere MQ keeps the log entries required to restart the queue manager without loss of data until they are no longer required to ensure queue manager

data recovery. The mechanism for releasing log files for reuse is described in “Using checkpointing to ensure complete recovery” on page 226.

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 228 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 “Using checkpointing to ensure complete recovery” on page 226 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 229 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 will be 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.

Using checkpointing to ensure 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 (units of work) active at the time of the checkpoint.

WebSphere MQ generates checkpoints automatically. They are taken when the queue manager starts, at shutdown, when logging space is running low, and after every 10 000 operations logged.

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

When WebSphere MQ restarts, it finds 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. All the operations that have taken place since the checkpoint are replayed forward. This is known as the replay phase. The replay phase brings the queues back to the logical state they were in before the system failure or shutdown. During the replay phase a list is created of the transactions that were in-flight when the system failure or shutdown occurred. Messages AMQ7229 and AMQ7230 are issued to indicate the progression of the replay phase.

In order to know which operations to back out or commit, WebSphere MQ accesses each active log record associated with an in-flight transaction. This is known as the recovery phase. Messages AMQ7231, AMQ7232 and AMQ7234 are issued to indicate the progression of the recovery phase.

Once all the necessary log records have been accessed during the recovery phase, each active transaction is in turn resolved and each operation associated with the transaction will be either backed out or committed. This is known as the resolution phase. Messages AMQ7233 is issued to indicate the progression of the resolution phase.

WebSphere MQ maintains internal pointers to the head and tail of the log. It moves the head pointer to the most recent checkpoint consistent with recovering message data.

Checkpoints are used to make recovery more efficient, and to control the reuse of primary and secondary log files.

In Figure 21 on page 227, 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.

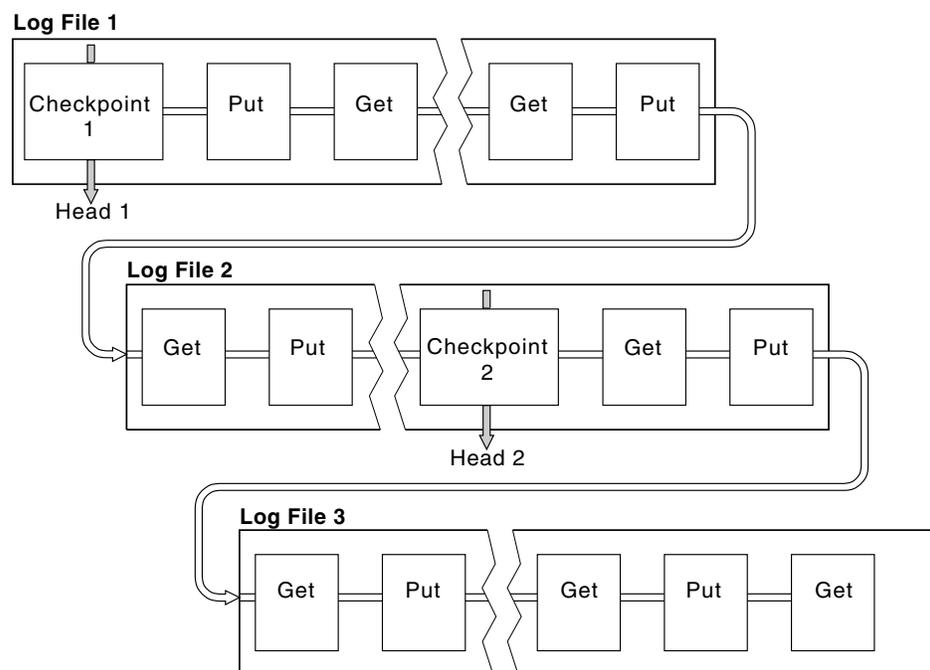


Figure 21. Checkpointing. For simplicity, only the ends of the log files are shown.

Checkpointing with long-running transactions

Figure 22 on page 228 shows how a long-running transaction affects reuse of log files. In the example, a long-running transaction has made an entry to the log, shown as LR 1, after the first checkpoint shown. The transaction does not complete (at point 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 before 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 full before the long-running transaction completes, secondary log files are used to avoid the logs getting full.

When the log head is moved and you are using circular logging, the primary log files might 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.

Log size calculations

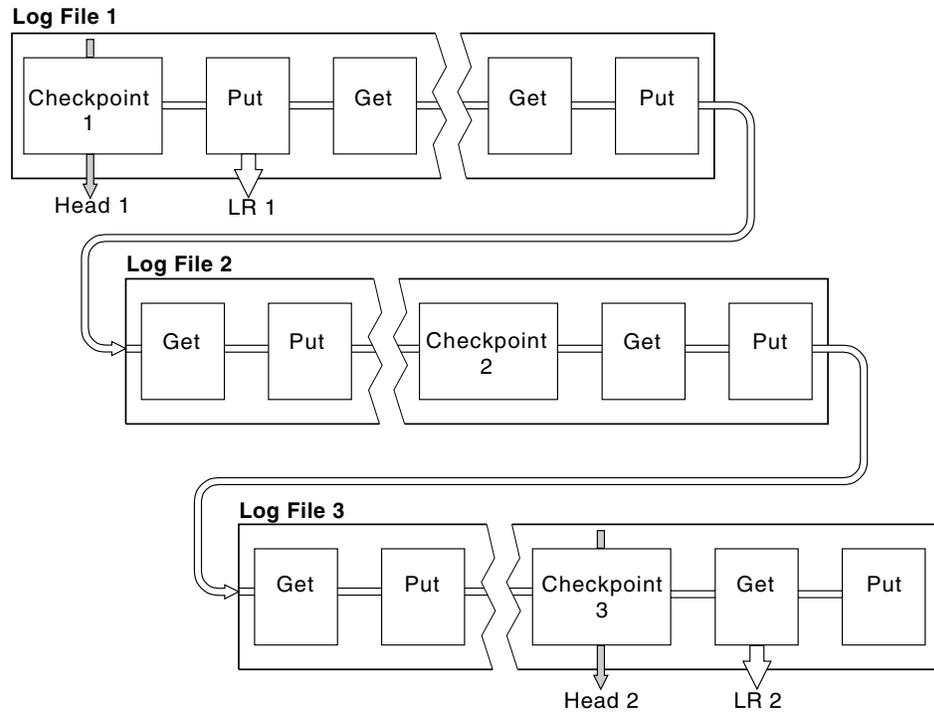


Figure 22. Checkpointing with a long-running transaction. For simplicity, only the ends of the log files are shown.

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 16 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 16. Log overhead sizes (all values are approximate)

Operation	Size
Put persistent message	750 bytes + message length If the message is large, it is divided into segments of 15700 bytes, each with a 300-byte overhead.
Get message	260 bytes
Syncpoint, commit	750 bytes
Syncpoint, 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.

Notes:

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 511 on UNIX systems, or 255 on Windows, 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 will, up to a limit, allocate a secondary log file when a log file becomes full, and the next primary log file in the sequence is not available.
6. Primary log files are made available for reuse during 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.

Managing logs

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 “Log defaults for WebSphere MQ” on page 111 and “Queue manager logs” on page 118). 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 *Monitoring WebSphere MQ*

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 **rccdmqimg** command. For more information about this command, see “rccdmqimg (record media image)” on page 349.

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 **rcdmqing** 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 **rcdmqing** command, described in “rcdmqing (record media image)” on page 349. 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 351 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 351 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 backup queue manager data, the less data you will 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.”
- “Restoring queue manager data” on page 236.

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 237.
- “Starting a backup queue manager” on page 238.

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 9, “Configuring WebSphere MQ,” on page 103.

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.

Backing up and restoring WebSphere MQ

For more information about name transformations, see “Understanding WebSphere MQ file names” on page 20.

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. (If you have queues larger than 2 GB, you cannot use **tar**; for more information, see “Enabling large queues” on page 50.)

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 (Windows systems),” on page 553 and Appendix C, “Directory structure (UNIX systems),” on page 555 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 will now start.

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”
- “Starting a backup queue manager” on page 238

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 235.
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:

Backing up and restoring WebSphere MQ

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.

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 will have 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 (Windows systems)," on page 553 and Appendix C, "Directory structure (UNIX systems)," on page 555 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 will now start.

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 will probably 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 will normally be able to restart. If any damage has occurred to the queue manager log files, then it is likely that the queue manager will not be 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 `rcrmqobj` command. For example:

```
rcrmqobj -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.

Recovery scenarios

- 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 will normally be 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.

Using dmpmqlog

<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 will have the same *XTranid*.

Prepare Transaction

Indicates that the queue manager is prepared to commit the updates associated with the specified *XTranid*. This log record is written as part of a two-phase commit involving other resource managers.

Commit Transaction

Indicates that the queue manager has committed all updates made by a transaction.

Rollback Transaction

This 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 will be instructed to commit. The *RMID* of each prepared resource manager is recorded in the log record. If the queue manager itself is participating in the transaction a *Participant Entry* with an *RMID* of zero will be present.

Transaction Forget

This log record is written by the XA Transaction Manager component of the queue manager. It follows the *Transaction Prepared* log record when the commit decision has been delivered to each participant.

Purge Queue

This logs the fact that all messages on a queue have been purged, for example, using the MQSC command CLEAR QUEUE.

Using dmpmqlog

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 15. 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 249
- “Application design considerations” on page 254
- “Error logs” on page 255
- “Dead-letter queues” on page 258
- “Configuration files and problem determination” on page 259
- “Tracing” on page 259
- “First-failure support technology (FFST)” on page 267
- “Problem determination with WebSphere MQ clients” on page 271
- “Java diagnostics” on page 271

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 is likely that you have not yet set it up correctly. See one of the following publications to check that you have installed the product correctly, and run the verification procedure:

- *WebSphere MQ for AIX, V6.0 Quick Beginnings*
- *WebSphere MQ for HP-UX, V6.0 Quick Beginnings*
- *WebSphere MQ for Linux, V6.0 Quick Beginnings*

Preliminary checks

- *WebSphere MQ for Solaris, V6.0 Quick Beginnings*
- *WebSphere MQ for Windows, V6.0 Quick Beginnings*

Also look at *WebSphere MQ Intercommunication* for information about post-installation configuration of WebSphere MQ.

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 255 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 will also fail 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.

Preliminary checks

- 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 251
- “Are some of your queues failing?” on page 252
- “Does the problem affect only remote queues?” on page 253
- “Is your application or system running slowly?” on page 253

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*)?

What next

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

If your application 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* 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 *WebSphere MQ Intercommunication* 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.

What next

- 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* 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 255 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* 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 *WebSphere MQ Intercommunication* 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.

What next

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.

Tuning performance for nonpersistent messages on AIX

If you are using AIX, consider setting your tuning parameter to exploit full performance for nonpersistent messages. To set the tuning parameter so that it takes effect immediately, issue the following command as a root user:

```
/usr/sbin/iioo -o j2_nPagesPerWriteBehindCluster=0
```

To set the tuning parameter so that it takes effect immediately and persists over reboots, issue the following command as a root user:

```
/usr/sbin/iioo -p -o j2_nPagesPerWriteBehindCluster=0
```

Normally, nonpersistent messages are kept only in memory, but there are circumstances where AIX can schedule nonpersistent messages to be written to disk. Messages scheduled to be written to disk are unavailable for MQGET until the disk write completes. The suggested tuning command varies this threshold; instead of scheduling messages to be written to disk when 16 kilobytes of data are queued, the write-to-disk occurs only when real storage on the machine becomes close to full. This is a global alteration and may effect other software components.

On AIX, when using multithreaded applications and especially when running on machines with multiple CPUs, we strongly recommend setting `AIXTHREADSCOPE=S` in the environment before starting the application, for better performance and more solid scheduling. For example:

```
export AIXTHREADSCOPE=S
```

Setting `AIXTHREAD_SCOPE=S` means that user threads created with default attributes will be placed into system-wide contention scope. If a user thread is created with system-wide contention scope, it is bound to a kernel thread and it is scheduled by the kernel. The underlying kernel thread is not shared with any other user thread.

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.

Number of threads in use

For WebSphere MQ for Windows, an application might require a large number of threads. Each queue manager process is allocated a maximum allowable number of application threads.

Applications might use too many threads. Consider whether the application takes into account this possibility and that it takes actions either to stop or to report this type of occurrence.

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.

Error logs

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, the location of the error log is shown in Table 17.

Table 17. Queue manager error log directory

Platform	Directory
UNIX systems	/var/mqm/qmgrs/ <i>qmname</i> /errors
Windows systems	c:\Program Files\IBM\WebSphere MQ\qmgrs\ <i>qmname</i> \errors

- If the queue manager name is not known, the location of the error log is shown in Table 18.

Table 18. System error log directory

Platform	Directory
UNIX systems	/var/mqm/errors
Windows systems	c:\Program Files\IBM\WebSphere MQ\errors

- If an error has occurred with a client application, the location of the error log on the client is shown in Table 19.

Table 19. Client error log directory

Platform	Directory
UNIX systems	/var/mqm/errors
Windows systems	c:\Program Files\IBM\WebSphere MQ Client\errors

In WebSphere MQ for Windows, an indication of the error is also added to the Application Log, which can be examined with the Event Viewer application provided with Windows systems.

Error log files

At installation time an errors subdirectory is created in the /var/mqm file path under UNIX systems, and in the \IBM\WebSphere MQ\ file path under Windows systems. 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, it creates three error log files when it needs them. These files have the same names as those in the system error log directory, that is AMQERR01, AMQERR02, and AMQERR03, and each has a default capacity of 256 KB. The capacity can be altered in the Extended queue manager properties page from the WebSphere MQ Explorer, or in the QMErrorLog stanza in the qm.ini file. These files are placed in the errors subdirectory in the /var/mqm/qmgrs/*qmname* file path under UNIX systems, or in the \IBM\WebSphere MQ\qmgrs*qmname*\errors file path under Windows systems.

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 queue manager is unavailable, or its name is unknown, in which case channel-related messages are placed in the system error log directory.

To examine the contents of any error log file, use your usual system editor.

Early errors

There are a number of special cases where these 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, because of a corrupt configuration file for example, no location information can be determined, errors are logged to an errors directory that is created at installation time on the root directory (/var/mqm or C:\Program Files\IBM\WebSphere MQ).

If WebSphere MQ can read its configuration information, and can access the value for the Default Prefix, errors are logged in the errors subdirectory of the directory identified by the Default Prefix attribute. For example, if the default prefix is C:\Program Files\IBM\WebSphere MQ, errors are logged in C:\Program Files\IBM\WebSphere MQ\errors.

For further information about configuration files, see Chapter 9, "Configuring WebSphere MQ," on page 103.

Note: Errors in the Windows Registry are notified by messages when a queue manager is started.

An example of an error log

Figure 23 shows an extract from a WebSphere MQ error log:

```
17/11/2004 10:32:29 - Process(2132.1) User(USER_1) Program(runmqchi.exe)
AMQ9542: Queue manager is ending.

EXPLANATION:
The program will end because the queue manager is quiescing.
ACTION:
None.
----- amqrimna.c : 931 -----
```

Figure 23. Sample WebSphere MQ error log

Error log access restrictions under UNIX systems

Certain error log directories and error logs have access restrictions under UNIX systems. To gain the following access permissions, a user or application must be a member of the mqm group:

- Read and write access to all queue manager error log directories.
- Read and write access to all queue manager error logs.
- Write access to the system error logs.

Error logs

If an unauthorized user or application attempts to write a message to a queue manager error log directory, the message is redirected to the system error log directory.

Ignoring error codes under UNIX systems

On UNIX systems, if you do not want certain error messages to be written to a queue manager error log, you can specify the error codes that are to be ignored using the QMErrorLog stanza. For more information, see “Queue manager error logs” on page 127.

Ignoring error codes under Windows systems

On Windows systems, if an error message has a severity of ERROR, the message is written to both the WebSphere MQ error log and the Windows Application Event Log.

If you do not want certain error messages to be written to the Windows Application Event Log, you can specify the error codes that are to be ignored in the Windows registry. Use the registry key:

```
HKEY_LOCAL_MACHINE\Software\IBM\MQSeries\CurrentVersion\IgnoreErrorCodes
```

The value that you set it to is an array of strings delimited by the NULL character, with each string value relating to the error code that you want ignored from the error log. The complete list is terminated with a NULL character, which is of type REG_MULTI_SZ.

For example, if you want WebSphere MQ to exclude error codes AMQ3045, AMQ6055, and AMQ8079 from the Windows Application Event Log, set the value to:

```
AMQ3045\0AMQ6055\0AMQ8079\0\0
```

The list of messages you want to exclude is defined for all queue managers on the machine. Any changes you make to the configuration will not take effect until each queue manager is restarted.

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 48 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 12, “The WebSphere MQ dead-letter queue handler,” on page 195.

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.

Errors in the Windows Registry are notified by messages when a queue manager is started.

Tracing

This section describes how to produce a trace for WebSphere MQ.

Tracing WebSphere MQ for Windows

You enable or modify tracing using the **strmqtrc** control command, described in “strmqtrc (Start trace)” on page 383. To stop tracing, you use the **endmqtrc** control command, described in “endmqtrc (end trace)” on page 337.

In WebSphere MQ for Windows systems, you can also start and stop tracing using the WebSphere MQ Explorer, as follows:

1. Start the WebSphere MQ Explorer from the Start menu.
2. In the Navigator View, right-click the **WebSphere MQ** tree node, and select **Trace...** The Trace Dialog is displayed.
3. Click **Start** or **Stop** as appropriate.

Selective component tracing on WebSphere MQ for Windows

Use the **-t** and **-x** options to control the amount of trace detail to record. By default, **all** trace points are enabled. The **-x** option enables you to specify the points that you do **not** want to trace. So if, for example, you want to trace only data flowing over communications networks, use:

```
strmqtrc -x all -t comms
```

For a full description of the trace command, see “strmqtrc (Start trace)” on page 383.

Trace files

During the installation process, you can choose the drive on which trace files are to be located. The trace files are always placed in the directory \<mqmwork>\trace, where <mqmwork> is the directory selected when WebSphere MQ was installed to hold WebSphere MQ data files.

Tracing

Trace files are named `AMQppppp.qq.TRC`, where:

`ppppp` Is the process identifier (PID) of the process producing the trace.
`qq` Starts at 0. If the full filename already exists, this value is incremented by one until a unique trace filename is found. A trace filename can already exist if a process is reused.

Notes:

1. The process identifier can contain fewer, or more, digits than shown in the example.
2. There is one trace file for each process running as part of the entity being traced.

An example of WebSphere MQ for Windows trace data

Figure 24 shows an extract from a WebSphere MQ for Windows trace:

```
Process : C:\Program Files\IBM\WebSphere MQ\bin\amqxssvn.exe
Version : 6.0.0.0   Level : p000-L050202
Date : 02/07/05   Time : 15:13:09

Counter  TimeStamp          PID.TID  Data
=====
00000D12 15:13:09.961154  10064.1  !! - Thread stack
00000D13 15:13:09.961173  10064.1  !! - -> InitProcessInitialisation
00000D14 15:13:09.961206  10064.1  --{ InitProcessInitialisation
00000D15 15:13:09.961899  10064.1  ---{ xcsReleaseThreadMutexSem
00000D16 15:13:09.961927  10064.1  ---} xcsReleaseThreadMutexSem (rc=OK)
00000D17 15:13:09.961942  10064.1  ---{ xcsGetEnvironmentInteger
00000D18 15:13:09.962017  10064.1  ----{ xcsGetEnvironmentString
00000D19 15:13:09.962045  10064.1  xcsGetEnvironmentString[AMQ_AFFINITY_MASK]#
= NULL
00000D1A 15:13:09.962051  10064.1  ----}! xcsGetEnvironmentString
(rc=xecE_E_ENV_VAR_NOT_FOUND)
00000D1B 15:13:09.962083  10064.1  ---}! xcsGetEnvironmentInteger
(rc=xecE_E_ENV_VAR_NOT_FOUND)
00000D1C 15:13:09.962092  10064.1  --} InitProcessInitialisation (rc=OK)
00000D1D 15:13:09.962097  10064.1  --{ xcsCreateThreadMutexSem
00000D1E 15:13:09.962106  10064.1  ---{ xcsCloseHandle
00000D1F 15:13:09.962113  10064.1  Handle (0x48), Handle Type (9)
00000D20 15:13:09.962121  10064.1  OK
00000D21 15:13:09.962125  10064.1  ---}! xcsCloseHandle (rc=Unknown(1))
00000D22 15:13:09.963830  10064.1  --} xcsCreateThreadMutexSem (rc=OK)
00000D23 15:13:09.963908  10064.1  --{ xcsProgramInit
00000D24 15:13:09.963914  10064.1  ---{ xcsProgramInit
00000D25 15:13:09.964557  10064.1  Adjusted Privilege NewPrivileges.Attribute
= 2 OldPrivileges.Attribute = 1245120
```

Figure 24. Sample WebSphere MQ for Windows trace

Tracing WebSphere MQ for UNIX systems

In WebSphere MQ for UNIX systems you enable or modify tracing using the `strmqtrc` control command, which is described in “`strmqtrc` (Start trace)” on page 383. To stop tracing, you use the `endmqtrc` control command, which is described in “`endmqtrc` (end trace)” on page 337. On WebSphere MQ for Linux (x86 platform) systems you can alternatively use the WebSphere MQ Explorer to start and stop tracing.

Trace output is unformatted; use the **dspmqrtrc** control command to format trace output before viewing. For example, to format all trace files in the current directory use the following command:

```
dspmqrtrc *.TRC
```

For more information on the control command **dspmqrtrc**, see “dspmqrtrc (display formatted trace output)” on page 327.

Selective component tracing on WebSphere MQ for UNIX systems

Use the **-t** and **-x** options to control the amount of trace detail to record. By default, all trace points are enabled. The **-x** option enables you to specify the points you do **not** want to trace. So if, for example, you want to trace, for queue manager QM1, only output data associated with using Secure Sockets Layer (SSL) channel security, use:

```
strmqtrc -m QM1 -t ssl
```

For a full description of the trace command, see “strmqtrc (Start trace)” on page 383.

Example trace data for WebSphere MQ for UNIX systems

Figure 25 shows an extract from a WebSphere MQ for HP-UX trace:

Timestamp	Process.Thread	Trace Data
15:19:01.830759	18153.1	Version : 6.0.0.0 Level : p000-L050228
15:19:01.831571	18153.1	Date : 03/04/05 Time : 15:19:01
15:19:01.831598	18153.1	PID : 18153 Process : strmqm_nd
15:19:01.831607	18153.1	QueueManager : QM1
15:19:01.831615	18153.1	-----
15:19:01.831623	18153.1	Trace Control Memory:
15:19:01.831632	18153.1	StrucId:
15:19:01.831640	18153.1	EarlyTraceOptions: 0
15:19:01.831649	18153.1	EarlyTraceMaxFileSize: 0
15:19:01.831657	18153.1	ActiveEntries: 0
15:19:01.831665	18153.1	Options MaxFileSize FileCount SubPoolName
15:19:01.831674	18153.1	1f4ffff 0 0 elk
15:19:01.831683	18153.1	0 0 0
15:19:01.831691	18153.1	0 0 0
15:19:01.831700	18153.1	0 0 0
15:19:01.831709	18153.1	0 0 0
15:19:01.831717	18153.1	0 0 0
15:19:01.831726	18153.1	0 0 0
15:19:01.831778	18153.1	0 0 0
15:19:01.831787	18153.1	0 0 0
15:19:01.831800	18153.1	Thread stack
15:19:01.838603	18153.1	-> zslWaitEC
15:19:01.838612	18153.1	-> zslCheckIfRunning
15:19:01.841757	18153.1	-> xcsInitialize
15:19:01.841767	18153.1	-> xcsGetEnvironmentString
15:19:01.841774	18153.1	---{ xcsGetEnvironmentString
15:19:01.841785	18153.1	xcsGetEnvironmentString[AMQ_SERVICE_MODULE]
		= NULL
15:19:01.841793	18153.1	---}! xcsGetEnvironmentString
		rc=xecE_E_ENV_VAR_NOT_FOUND
15:19:01.841801	18153.1	---{ xcsReleaseThreadMutexSem

Figure 25. Sample WebSphere MQ for HP-UX trace

Tracing

Figure 26 shows an extract from a WebSphere MQ for Solaris trace:

Timestamp	Process.Thread	Trace Data
15:00:04.324190	12277.1	Version : 6.0.0.0 Level : p000-L050203
15:00:04.325045	12277.1	Date : 07/02/05 Time : 15:00:04
15:00:04.325375	12277.1	PID : 12277 Process : strmqm
15:00:04.325403	12277.1	QueueManager : QM1
15:00:04.325419	12277.1	-----
15:00:04.325446	12277.1	Trace Control Memory:
15:00:04.325471	12277.1	StrucId:
15:00:04.325490	12277.1	EarlyTraceOptions: 0
15:00:04.325507	12277.1	EarlyTraceMaxFileSize: 0
15:00:04.325527	12277.1	ActiveEntries: 0
15:00:04.325544	12277.1	Options MaxFileSize FileCount SubPoolName
15:00:04.325566	12277.1	74ffff 0 0 e1k
15:00:04.325587	12277.1	0 0 0
15:00:04.325609	12277.1	0 0 0
15:00:04.325632	12277.1	0 0 0
15:00:04.325654	12277.1	0 0 0
15:00:04.325677	12277.1	0 0 0
15:00:04.325698	12277.1	0 0 0
15:00:04.325774	12277.1	0 0 0
15:00:04.325798	12277.1	0 0 0
15:00:04.325891	12277.1	Thread stack
15:00:04.325971	12277.1	-> zslWaitEC
15:00:04.326078	12277.1	-> zslCheckIfRunning
15:00:04.326098	12277.1	-> xcsInitialize
15:00:04.326147	12277.1	-> xcsGetEnvironmentString
15:00:04.326186	12277.1	---{ xcsGetEnvironmentString
15:00:04.326241	12277.1	xcsGetEnvironmentString[AMQ_SERVICE_MODULE] = NULL

Figure 26. Sample WebSphere MQ for Solaris trace

Figure 27 shows an extract from a WebSphere MQ for Linux trace:

Timestamp	Process.Thread	Trace Data
15:15:05.931699	1159.1	Version : 6.0.0.0 Level : p000-L050107
15:15:05.931843	1159.1	Date : 02/07/05 Time : 15:15:05
15:15:05.932016	1159.1	PID : 1159 Process : amqzdmaa
15:15:05.932024	1159.1	QueueManager : QM1
15:15:05.932028	1159.1	-----
15:15:05.932037	1159.1	Trace Control Memory:
15:15:05.932044	1159.1	StrucId:
15:15:05.932049	1159.1	EarlyTraceOptions: 0
15:15:05.932054	1159.1	EarlyTraceMaxFileSize: 0
15:15:05.932059	1159.1	ActiveEntries: 0
15:15:05.932064	1159.1	Options MaxFileSize FileCount SubPoolName
15:15:05.932070	1159.1	74ffff 0 0 elk
15:15:05.932075	1159.1	0 0 0
15:15:05.932081	1159.1	0 0 0
15:15:05.932086	1159.1	0 0 0
15:15:05.932091	1159.1	0 0 0
15:15:05.932097	1159.1	0 0 0
15:15:05.932102	1159.1	0 0 0
15:15:05.932107	1159.1	0 0 0
15:15:05.932112	1159.1	0 0 0
15:15:05.932138	1159.1	Thread stack
15:15:05.932149	1159.1	-> xxxInitialize
15:15:05.932158	1159.1	{ xxxInitialize
15:15:05.932165	1159.1	-{ xcsSetlocale
15:15:05.932189	1159.1	category(6) locale() buffer(0xbfffd340) buflen(1285)
15:15:05.932196	1159.1	Doing the first-thread-only locale check
15:15:05.932326	1159.1	-} xcsSetlocale rc=OK
15:15:05.932344	1159.1	-{ xcsGetMem

Figure 27. Sample WebSphere MQ for Linux trace

Tracing

Figure 28 shows an extract from a WebSphere MQ for AIX trace:

Timestamp	Process.Thread	Trace Data
13:12:12.336214	286850.1	Version : 6.0.0.0 Level : p000-L050201
13:12:12.336345	286850.1	Date : 02/15/05 Time : 13:12:12
13:12:12.336419	286850.1	PID : 286850 Process : amqzlaa0_nd
13:12:12.336444	286850.1	QueueManager : QM1
13:12:12.336468	286850.1	-----
13:12:12.336493	286850.1	Trace Control Memory:
13:12:12.336518	286850.1	StrucId:
13:12:12.336542	286850.1	EarlyTraceOptions: 0
13:12:12.336567	286850.1	EarlyTraceMaxFileSize: 0
13:12:12.336591	286850.1	ActiveEntries: 0
13:12:12.336616	286850.1	Options MaxFileSize FileCount SubPoolName
13:12:12.336641	286850.1	74ffff 0 0 elk
13:12:12.336668	286850.1	0 0 0
13:12:12.336692	286850.1	0 0 0
13:12:12.336718	286850.1	0 0 0
13:12:12.336742	286850.1	0 0 0
13:12:12.336768	286850.1	0 0 0
13:12:12.336792	286850.1	0 0 0
13:12:12.336817	286850.1	0 0 0
13:12:12.336842	286850.1	0 0 0
13:12:12.336870	286850.1	Thread stack
13:12:12.336897	286850.1	-> xxxInitialize
13:12:12.336921	286850.1	{ xxxInitialize
13:12:12.336947	286850.1	-{ xcsSetlocale
13:12:12.336977	286850.1	category(-1) locale() buffer(ffffffffffffcf08) buflen(1285)
13:12:12.337005	286850.1	Doing the first-thread-only locale check . . .
13:12:12.338602	286850.1	-} xcsSetlocale rc=0K

Figure 28. Sample WebSphere MQ for AIX trace

Trace files

All trace files are created in the directory `/var/mqm/trace`.

Note: You can accommodate production of large trace files by mounting a temporary file system over this directory.

Trace files are named `AMQppppp.qq.TRC`, where:

<i>ppppp</i>	Is the ID of the process reporting the error.
<i>qq</i>	Starts at 0. If the full filename already exists, this value is incremented by one until a unique trace filename is found. A trace filename can already exist if a process is reused.

Notes:

1. The process identifier can contain fewer, or more, digits than shown in the example.
2. There is one trace file for each process running as part of the entity being traced.

In order to format or view a trace file, you must be either the creator of the trace file, or a member of the `mqm` group.

Tracing Secure Sockets Layer (SSL) on UNIX systems

On UNIX systems, you can request trace information for either iKeyman, the SSL functions, or both:

- To request iKeyman tracing, execute the `gsk7ikm` command with the `-D` flag:
`gsk7ikm -Dkeyman.debug=true`

iKeyman writes three trace files to the directory from which you start it, so consider starting iKeyman from the `/var/mqm/trace` directory. The trace files iKeyman generates are:

ikmgdbg.log Java related trace

ikmjdbg.log JNI related trace

ikmcdbg.log C related trace

SSL trace files have the names `AMQ.SSL.TRC` and `AMQ.SSL.TRC.1`. You cannot format SSL trace files; send them unchanged to IBM support.

Tracing with the AIX system trace

In addition to the WebSphere MQ trace, WebSphere MQ for AIX users can use the standard AIX system trace. AIX system tracing is a two-step process:

1. Gathering the data
2. Formatting the results

WebSphere MQ uses two trace hook identifiers:

X'30D' This event is recorded by WebSphere MQ on entry to or exit from a subroutine.

X'30E' This event is recorded by WebSphere MQ to trace data such as that being sent or received across a communications network.

Trace provides detailed execution tracing to help you to analyze problems. IBM service support personnel might ask for a problem to be re-created with trace enabled. The files produced by trace can be **very** large so it is important to qualify a trace, where possible. For example, you can optionally qualify a trace by time and by component.

There are two ways to run trace:

1. Interactively.

The following sequence of commands runs an interactive trace on the program `myprog` and ends the trace.

```
trace -j30D,30E -o trace.file
->!myprog
->q
```

2. Asynchronously.

The following sequence of commands runs an asynchronous trace on the program `myprog` and ends the trace.

```
trace -a -j30D,30E -o trace.file
myprog
trcstop
```

You can format the trace file with the command:

```
trcrpt -t /usr/mqm/lib/amqtrc.fmt trace.file > report.file
```

Tracing

report.file is the name of the file where you want to put the formatted trace output.

Note: All WebSphere MQ activity on the machine is traced while the trace is active.

Selective component tracing on WebSphere MQ for AIX

Use the environment variable MQS_TRACE_OPTIONS to activate the high detail and parameter tracing functions individually. Because it enables tracing to be active without these functions, you can use it to reduce the overhead on execution speed when you are trying to reproduce a problem with tracing switched on.

Only set the environment variable MQS_TRACE_OPTIONS if you have been instructed to do so by your service personnel.

Typically MQS_TRACE_OPTIONS must be set in the process that starts the queue manager, and before the queue manager is started, or it is not recognized. Set MQS_TRACE_OPTIONS before tracing starts. If it is set after tracing starts it is not recognized.

SSL trace: If you request SSL trace, note the following:

- SSL trace is written to the directory /var/mqm/trace.
- The SSL trace files are AMQ.SSL.TRC and AMQ.SSL.TRC.1.
- You cannot format SSL trace files; send them unchanged to IBM support.

An example of WebSphere MQ for AIX trace data

The following example is an extract of an AIX trace:

ID	ELAPSED_SEC	DELTA_MSEC	APPL	SYSCALL	KERNEL	INTERRUPT
30D	0.000000000	0.000000	MQS FNC	Exit!.....	23298.1	ziiSendReceiveAgent rc=00000814
30D	0.000009512	0.009512	MQS FNC	Entry.....	23298.1	zcpDeleteMessage
30D	0.000011869	0.002357	MQS FNC	Exit!.....	23298.1	zcpDeleteMessage rc=00000000
30D	0.000014196	0.002327	MQS FNC	Exit!.....	23298.1	ziiSPIInq1 rc=00000814
30D	0.000016727	0.002531	MQS FNC	Exit!.....	23298.1	lpiSPIInq1 rc=00000814
30D	0.000019847	0.003120	MQS FNC	Entry....	23298.1	lpiSPIInq1
30D	0.000022465	0.002618	MQS FNC	Entry.....	23298.1	zstVerifyPCD
30D	0.000024792	0.002327	MQS FNC	Exit!.....	23298.1	zstVerifyPCD rc=00000000
30D	0.000027505	0.002713	MQS FNC	Entry.....	23298.1	xcsCheckPointer
30D	0.000032436	0.004931	MQS FNC	Exit!.....	23298.1	xcsCheckPointer rc=00000000
30D	0.000034923	0.002487	MQS FNC	Entry.....	23298.1	xcsCheckPointer
30D	0.000039716	0.004793	MQS FNC	Exit!.....	23298.1	xcsCheckPointer rc=00000000
30D	0.000042218	0.002502	MQS FNC	Entry.....	23298.1	xcsCheckPointer
30D	0.000046982	0.004764	MQS FNC	Exit!.....	23298.1	xcsCheckPointer rc=00000000
30D	0.000049593	0.002611	MQS FNC	Entry.....	23298.1	ziiSPIInq1
30D	0.000052116	0.002523	MQS FNC	Entry....	23298.1	ziiCreateIPCCMessage
30D	0.000054611	0.002495	MQS FNC	Entry.....	23298.1	zcpCreateMessage
30E	0.000059062	0.004451	Terminus(0)	RequestedSize(236)		
30D	0.000061549	0.002487	MQS FNC	Exit!.....	23298.1	zcpCreateMessage rc=00000000
30D	0.000063884	0.002335	MQS FNC	Exit!.....	23298.1	ziiCreateIPCCMessage rc=00000000

Figure 29. Sample WebSphere MQ for AIX trace

First-failure support technology (FFST)

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

FFST: WebSphere MQ for Windows

In WebSphere MQ for Windows, FFST information is recorded in a file in the `c:\Program Files\IBM\WebSphere MQ\errors` directory.

An FFST file contains one or more records. Each FFST record contains information about an error that is normally severe, and possibly unrecoverable. These records typically indicate either a configuration problem with the system or a WebSphere MQ internal error.

FFST files are named `AMQnnnnn.mm.FDC`, where:

<i>nnnnn</i>	Is the ID of the process reporting the error
<i>mm</i>	Starts at 0. If the full filename already exists, this value is incremented by one until a unique FFST filename is found. An FFST filename can already exist if a process is reused.

An instance of a process will write all FFST information to the same FFST file. If multiple errors occur during a single execution of the process, an FFST file can contain many records.

When a process writes an FFST record it also sends a record to the Event Log. The record contains the name of the FFST file to assist in automatic problem tracking. The Event log entry is made at the application level.

A typical FFST log is shown in Figure 30.

```

+-----+
| WebSphere MQ First Failure Symptom Report |
|-----|
| Date/Time      :- Sun February 06 21:59:06 GMT Standard Time 2005 |
| Host Name     :- 99VXY09 (Windows XP Build 2600: Service Pack 1) |
| PIDS          :- 5724H7200 |
| LVLS          :- 6.0.0.0 |
| Product Long Name :- WebSphere MQ for Windows |
| Vendor        :- IBM |
| Probe Id      :- HL010004 |
| Application Name :- MQM |
| Component     :- hlgReserveLogSpace |
| SCCS Info     :- lib/logger/amqhlge0.c, 1.26 |
| Line Number   :- 246 |
| Build Date    :- Feb 2 2005 |
| CMVC level    :- p000-L050202 |
| Build Type    :- IKAP - (Production) |
| UserID       :- IBM_User |
| Process Name  :- C:\Program Files\IBM\WebSphere MQ\bin\amqz1aa0.exe |
| Process      :- 00003456 |
| Thread       :- 00000030 |
| QueueManager :- qmgr2 |
| ConnId(1) IPCC :- 162 |
| ConnId(2) QM  :- 45 |
| Major Errorcode :- hrcE_LOG_FULL |
| Minor Errorcode :- OK |
| Probe Type     :- MSGAMQ6709 |
| Probe Severity :- 2 |
| Probe Description :- AMQ6709: The log for the Queue manager is full. |
| FDCSequenceNumber :- 0 |
+-----+

MQM Function Stack
z1aMainThread
z1aProcessMessage
z1aProcessMQIRequest
z1aMQPUT
zsQMMPUT
kpiMQPUT
kqiPutIt
kqiPutMsgSegments
apiPutMessage
aqmPutMessage
aqhPutMessage
aqqWriteMsg
aqqWriteMsgData
aqlReservePutSpace
a1mReserveSpace
hlgReserveLogSpace
xcsFFST

MQM Trace History
-----} hlgReserveLogSpace rc=hrcW_LOG_GETTING_VERY_FULL
-----{ x1lLongLockRequest
-----} x1lLongLockRequest rc=OK

...

```

Figure 30. Sample WebSphere MQ for Windows First Failure Symptom Report

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

In certain circumstances a small dump file can be generated in addition to an FFST file and placed in the `c:\Program Files\IBM\WebSphere MQ\errors` directory. A dump file will have the same name as the FFST file, in the form `AMQnnnnn.mm.dmp`. These files can be used by IBM to assist in problem determination.

FFST: WebSphere MQ for UNIX systems

For WebSphere MQ for UNIX systems, FFST information is recorded in a file in the `/var/mqm/errors` directory.

An FFST file contains one or more records. Each FFST record contains information about an error that is normally severe, and possibly unrecoverable. These records indicate either a configuration problem with the system or a WebSphere MQ internal error.

FFST files are named `AMQnnnnn.mm.FDC`, where:

<i>nnnnn</i>	Is the ID of the process reporting the error
<i>mm</i>	Starts at 0. If the full filename already exists, this value is incremented by one until a unique FFST filename is found. An FFST filename can already exist if a process is reused.

An instance of a process will write all FFST information to the same FFST file. If multiple errors occur during a single execution of the process, an FFST file can contain many records.

In order to read the contents of a FFST file, you must be either the creator of the file, or a member of the `mqm` group.

When a process writes an FFST record, it also sends a record to `syslog`. The record contains the name of the FFST file to assist in automatic problem tracking. The `syslog` entry is made at the `user.error` level. See the operating-system documentation about `syslog.conf` for information about configuring this.

Some typical FFST data is shown in Figure 31 on page 270.

```

+-----+
| WebSphere MQ First Failure Symptom Report |
|=====|
| Date/Time      :- Friday February 04 10:39:24 MST 2005 |
| Host Name     :- mqperfh2 (HP-UX B.11.23) |
| PIDS          :- 5724H7202 |
| LVLS          :- 6.0.0.0 |
| Product Long Name :- WebSphere MQ for HP-UX |
| Vendor        :- IBM |
| Probe Id      :- XC034255 |
| Application Name :- MQM |
| Component     :- xcsWaitEventSem |
| SCCS Info     :- lib/cs/unix/amqxerrx.c, 1.204 |
| Line Number   :- 6262 |
| Build Date    :- Feb 3 2005 |
| CMVC level    :- p000-L050203 |
| Build Type    :- IKAP - (Production) |
| UserID        :- 00000106 (mqperfh) |
| Program Name  :- amqzmuc0 |
| Addressing mode :- 64-bit |
| Process       :- 15497 |
| Thread        :- 1 |
| QueueManager  :- CSIM |
| ConnId(2) QM  :- 4 |
| Major Errorcode :- OK |
| Minor Errorcode :- OK |
| Probe Type    :- INCORROUT |
| Probe Severity :- 4 |
| Probe Description :- AMQ6109: An internal WebSphere MQ error has occurred. |
| FDCSequenceNumber :- 0 |
|-----+
| MQM Function Stack |
| amqzmuc0 |
| xcsWaitEventSem |
| xcsFFST |
|
| MQM Trace History |
| Data: 0x00003c87 |
| --} xcsCheckProcess rc=OK |
| --{ xcsRequestMutexSem |
| --} xcsRequestMutexSem rc=OK |
|
| ... |

```

Figure 31. FFST report for WebSphere MQ for UNIX systems

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

However, there are some problems that the system administrator might be able to solve. If the FFST shows *out of resource* or *out of space on device* descriptions when calling one of the IPC functions (for example, **semop** or **shmget**), it is likely that the relevant kernel parameter limit has been exceeded.

If the FFST report shows a problem with **setitimer**, it is likely that a change to the kernel timer parameters is needed.

To resolve these problems, increase the IPC limits, rebuild the kernel, and restart the machine. See one of the following for further information:

- *WebSphere MQ for AIX, V6.0 Quick Beginnings*
- *WebSphere MQ for HP-UX, V6.0 Quick Beginnings*

- *WebSphere MQ for Linux, V6.0 Quick Beginnings*
- *WebSphere MQ for Solaris, V6.0 Quick Beginnings*

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.

Java diagnostics

For Java components of WebSphere MQ, for example the WebSphere MQ Explorer and the Java implementation of WebSphere MQ Transport for SOAP, diagnostic information is output using the standard WebSphere MQ diagnostic facilities or by Java diagnostic classes. Diagnostic information in this context consists of trace, first-failure data capture (FFDC) and error messages.

You can choose to have this information produced using WebSphere MQ facilities or the Java classes. You should generally use the WebSphere MQ diagnostic facilities if they are available on the local system.

You might want to use the Java diagnostics in the following circumstances:

- On a system on which queue managers are available, if the queue manager is managed separately from the software you are running.
- To reduce performance overhead of WebSphere MQ trace.

To request and configure diagnostic output, two system properties are used when starting a WebSphere MQ Java process:

- System property `com.ibm.mq.commonservices` specifies a standard Java property file, which contains a number of lines which are used to configure the diagnostic outputs. Each line of code in the file is free-format, and is terminated by a new line character.
- System property `com.ibm.mq.commonservices.diagid` associates trace and FFDC files with the process which created them.

Using `com.ibm.mq.commonservices`

The `com.ibm.mq.commonservices` properties file contains the following entries relating to the output of diagnostics from the Java components of WebSphere MQ. Note that case is significant in all these entries.

Java diagnostics

Diagnostics.MQ=*enabled | disabled*

Are WebSphere MQ diagnostics to be used? If **Diagnostics.MQ** is enabled, diagnostic output is as for other WebSphere MQ components; trace output is controlled by the parameters in the **strmqtrc** and **endmqtrc** control commands, or the equivalent. The default is *enabled*.

Diagnostics.Java=*options*

Which components are traced using Java trace. Options are one or more of *explorer*, *soap*, and *wmqjavaclasses*, separated by commas, where "explorer" refers to the diagnostics from the WebSphere MQ Explorer, "soap" refers to the diagnostics from the running process within WebSphere MQ Transport for SOAP, and "wmqjavaclasses" refers to the diagnostics from the underlying WebSphere MQ Java classes. By default no components are traced.

Diagnostics.Java.Trace.Detail=*high | medium | low*

Detail level for Java trace. The *high* and *medium* detail levels match those used in WebSphere MQ tracing but *low* is unique to Java trace. This property is ignored if **Diagnostics.Java** is not set. The default is *medium*.

Diagnostics.Java.Trace.Destination.File=*enabled | disabled*

Whether Java trace is written to a file. This property is ignored if **Diagnostics.Java** is not set. The default is *disabled*.

Diagnostics.Java.Trace.Destination.Console=*enabled | disabled*

Whether Java trace is written to the system console. This property is ignored if **Diagnostics.Java** is not set. The default is *disabled*.

Diagnostics.Java.Trace.Destination.Pathname=*dirname*

The directory to which Java trace is written. This property is ignored if **Diagnostics.Java** is not set or **Diagnostics.Java.Trace.Destination.File**=*disabled*. On UNIX systems, the default is `/var/mqm/trace` if it is present, otherwise the Java console (`System.err`). On Windows, the default is the system console.

Diagnostics.Java.FFDC.Destination.Pathname=*dirname*

The directory to which Java FFDC output is written. The default is the current working directory.

Diagnostics.Java.Errors.Destination.Filename=*filename*

The fully qualified filename to which Java error messages are written. The default is `AMQJAVA.LOG` in the current working directory.

An example of a `com.ibm.mq.commonservices` properties file is given in Figure 32 on page 273. Lines beginning with the number sign (#) are treated as comments.

```

#
# Base WebSphere MQ diagnostics are disabled
#
Diagnostics.MQ=disabled
#
# Java diagnostics for WebSphere MQ Transport for SOAP
# and the WebSphere MQ Java Classes are both enabled
#
Diagnostics.Java=soap,wmqjavaclasses
#
# High detail Java trace
#
Diagnostics.Java.Trace.Detail=high
#
# Java trace is written to a file and not to the console.
#
Diagnostics.Java.Trace.Destination.File=enabled
Diagnostics.Java.Trace.Destination.Console=disabled
#
# Directory for Java trace file
#
Diagnostics.Java.Trace.Destination.Pathname=c:\\tracedir
#
# Directory for First Failure Data Capture
#
Diagnostics.Java.FFDC.Destination.Pathname=c:\\ffdcdir
#
# Directory for error logging
#
Diagnostics.Java.Errors.Destination.FileName=c:\\errorsdir\\SOAPERRORS.LOG
#

```

Figure 32. Sample `com.ibm.mq.commonservices` properties file

A sample properties file, `WMQSoap_RAS.properties`, is also supplied as part of the "Java messaging and SOAP transport" install option.

Java trace and FFDC files

When Java trace is generated for the WebSphere MQ Explorer or for WebSphere MQ Transport for SOAP it is written to a file with a name of the format `AMQ.diagid.counter.TRC` where *diagid* is the value of the system property `com.ibm.mq.commonservices.diagid` associated with this Java process, as described earlier in this section, and *counter* is an integer greater than or equal to 0. All letters in the name are in upper case. This matches the naming convention used for normal WebSphere MQ trace.

If `com.ibm.mq.commonservices.diagid` is not specified, the value of *diagid* is the current time, in the format `YYYYMMDDhhmssmm`.

The WebSphere MQ Java classes trace file has a name based on the equivalent WebSphere MQ Explorer or SOAP Java trace file. The name differs in that it has the string `.JC` added before the `.TRC` string, giving a format of `AMQ.diagid.counter.JC.TRC`.

When Java FFDC is generated for the WebSphere MQ Explorer or for WebSphere MQ Transport for SOAP it is written to a file with a name of the format `AMQ.diagid.counter.FDC` where *diagid* and *counter* are as described for Java trace files.

Java diagnostics

Java error message output for the WebSphere MQ Explorer and for WebSphere MQ Transport for SOAP is written to the file specified by *Diagnostics.Java.Errors.Destination.FileName* for the appropriate Java process. The format of these files matches closely the format of the standard WebSphere MQ error logs.

When a process is writing trace information to a file, it appends to a single trace output file for the lifetime of the process. Similarly, a single FFDC output file is used for the lifetime of a process.

All trace output is in the UTF-8 character set.

Note that none of the above applies to output of FFDC data or error messages for the WebSphere MQ Java classes. This occurs as part of exception logging as detailed in *WebSphere MQ Using Java*. The WebSphere MQ Java class trace is also detailed in *WebSphere MQ Using Java*.

Part 6. WebSphere MQ control commands

Chapter 16. How to use WebSphere MQ control commands 277

Names of WebSphere MQ objects.	277
How to read syntax diagrams	278
Example syntax diagram	279
Syntax help	280
Examples.	280

Chapter 17. The control commands 281

amqccert (check certificate chains)	283
amqmdain (WebSphere MQ services control)	285
amqtcert (transfer certificates)	291
crtmqcvx (data conversion)	297
crtmqm (create queue manager)	299
dltmqm (delete queue manager)	303
dmpmqaut (dump authority)	305
dmpmqlog (dump log)	309
dspm (display queue managers)	311
dspmqa (display authority)	312
dspmqs (display command server)	316
dspmql (display files)	317
dspmqr (WebSphere MQ display route application)	319
dspmqr (display formatted trace output)	327
dspmqr (display transactions)	328
dspmqr (display version information)	329
endmqsv (end command server)	331
endmqlsr (end listener)	333
endmqdnm (stop .NET monitor)	334
endmqm (end queue manager)	335
endmqtr (end trace)	337
mqftapp (run File Transfer Application GUI)	338
mqftrcv (receive file on server)	339
mqftrvc (receive file on client)	342
mqftsnd (send file from server)	345
mqftsnc (send file from client)	347
rcdmqimg (record media image)	349
rcrmqobj (recreate object)	351
rsvmqtrn (resolve transactions)	353
runmqchi (run channel initiator)	355
runmqchl (run channel)	356
runmqdlq (run dead-letter queue handler)	357
runmqdnm (run .NET monitor)	358
runmqlsr (run listener)	361
runmqsc (run MQSC commands)	363
runmqtm (start client trigger monitor)	366
runmqtrm (start trigger monitor)	367
setmqaut (set or reset authority)	368
setmqcrl (set certificate revocation list (CRL) LDAP server definitions)	375
setmqscp (set service connection points)	377
strmqcfg (start WebSphere MQ Explorer)	379
strmqcsv (start command server)	380
strmqm (start queue manager)	381
strmqtr (Start trace)	383

Chapter 18. Managing keys and certificates 387

Preparing to use the gsk7cmd command	387
gsk7cmd and runmqckm commands	387
Commands for a CMS key database only	388
Commands for CMS or PKCS #12 key databases	388
Commands for cryptographic device operations	390
gsk7cmd and runmqckm options	392

Chapter 16. 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. For more information about this, see “Authority to administer WebSphere MQ” on page 129. In addition, note the following environment-specific information:

WebSphere MQ for Windows

All control commands can be issued from a command line. Command names and their flags are not case sensitive: you can enter them in uppercase, lowercase, or a combination of uppercase and lowercase. However, arguments to control commands (such as queue names) are case sensitive.

In the syntax descriptions, the hyphen (-) is used as a flag indicator. You can use the forward slash (/) instead of the hyphen.

WebSphere MQ for UNIX systems

All WebSphere MQ control commands can be issued from a shell. All commands are case-sensitive.

A subset of the control commands can be issued using the WebSphere MQ Explorer.

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)

Notes:

1. Forward slash and percent are special characters. If you use either of these characters in a name, the name must be enclosed in double quotation marks whenever it is used.

Names

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 book contains syntax diagrams (sometimes referred to as *railroad* diagrams).

Each syntax diagram begins with a double right arrow and ends with a right and left arrow pair. Lines beginning with a single right arrow are continuation lines. You read a syntax diagram from left to right and from top to bottom, following the direction of the arrows.

Other conventions used in syntax diagrams are:

Table 20. How to read syntax diagrams

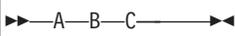
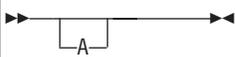
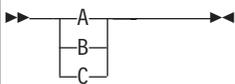
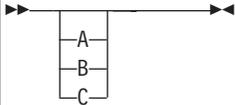
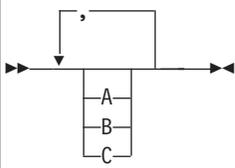
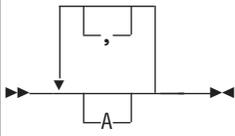
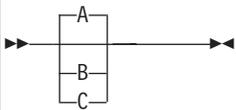
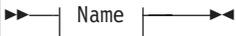
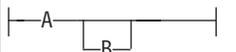
Convention	Meaning
	You must specify values A, B, and C. Required values are shown on the main line of a syntax diagram.
	You can specify value A. Optional values are shown below the main line of a syntax diagram.
	Values A, B, and C are alternatives, one of which you must specify.
	Values A, B, and C are alternatives, one of which you might specify.
	You can specify one or more of the values A, B, and C. Any required separator for multiple or repeated values (in this example, the comma (,)) is shown on the arrow.
	You can specify value A multiple times. The separator in this example is optional.

Table 20. How to read syntax diagrams (continued)

Convention	Meaning
	<p>Values A, B, and C are alternatives, one of which you can specify. If you specify none of the values shown, the default A (the value shown above the main line) is used.</p>
<p>  </p> <p>Name:</p> 	<p>The syntax fragment Name is shown separately from the main syntax diagram.</p>
<p>Punctuation and uppercase values</p>	<p>Specify exactly as shown.</p>

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?
```

Syntax diagrams

The space before the *name* value is significant, and that if you do not code *name* at all, you must still code the comma before how are you?.

Syntax help

You can obtain help for the syntax of any 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 17. The control commands

This section provides reference information for each of the following WebSphere MQ control commands:

Command name	Purpose
amqccert	Check certificate chains
amqmdain	Configure or control WebSphere MQ services (Windows systems only)
amqtcert	Transfer certificates
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
endmqdnm	Stop .NET monitor
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
mqftrcv	Receive file using the File Transfer Application (server)
mqftrvc	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
runmqdnm	Run .NET monitor
runmqlsr	Start a listener process
runmqsc	Issue MQSC commands to a queue manager

amqccert

runmqtmc	Invoke a trigger monitor for a client (AIX clients only)
runmqtrm	Invoke a trigger monitor for a server
setmqaut	Change authorizations to an object
setmqcrl	Set certificate revocation list (CRL) LDAP server definitions (Windows systems only)
setmqscp	Set service connection points (Windows systems only)
strmqcsv	Start the command server for a queue manager
strmqm	Start a local queue manager
strmqtrc	Enable tracing

amqccert (check certificate chains)

Purpose

The **amqccert** command applies to WebSphere MQ for Windows only.

The **amqccert** command is used during SSL Certificate Migration from WebSphere MQ for Windows Version 5.3, or Version 5.3.1. SSL Certificate Migration instructions are detailed in the *WebSphere MQ Migration Information*.

In this section when referring to a WebSphere MQ Certificate Store file, we are specifically referring to a WebSphere MQ for Windows Version 5.3, or Version 5.3.1, Certificate Store file.

To use **amqccert** you must be either an administrator or a member of the mqm group.

The **amqccert** control command is used to determine whether there are any incomplete certificate chains in a WebSphere MQ Certificate Store file. A report is generated that lists each incomplete certificate chain accompanied by information relating to the certificate chain.

Incomplete certificate chains must be completed before the SSL Certificate Migration process can continue. The following are available with WebSphere MQ for Windows Version 5.3, and Version 5.3.1, to help complete certificate chains:

- The control command **amqmcert** (manage certificates).
- The Services snap-in.

Syntax

►►—amqccert—*FileName*—————►►

Required parameters

FileName

Specifies is the absolute (rather than relative) directory path name and filename (excluding the .sto suffix) of a WebSphere MQ Certificate Store.

Examples

In the following example reports the term, Microsoft Certificate Store, refers to a WebSphere MQ Certificate Store file.

amqccert C:\SSL\C1ient

Generates a report that details whether there are any incomplete certificate chains.

The following is an example of a report that details no incomplete certificate chains:

```
C:\ssl\client
5724-B41 (C) Copyright IBM Corp. 1994, 2005. ALL RIGHTS RESERVED.
The number of certificates in the Microsoft Certificate Store
'c:\ssl\client' is '13'.
```

Certificate chain checking has completed with no failures.
The Check Certificate Chains (amqccert) command has completed.

amqccert

The following is an example of a report the details two incomplete certificate chains:

```
C:\ssl\client
5724-B41 (C) Copyright IBM Corp. 1994, 2005. ALL RIGHTS RESERVED.
The number of certificates in the Microsoft Certificate Store
                                     'c:\ssl\client' is '13'.
```

The signer certificate 'GlobalSign Primary Class 1 CA' is missing for the following certificate.

```
Microsoft Certificate Store: 'c:\ssl\client'.
Certificate Subject:         'GlobalSign PersonalSign Class 1 CA'.
Certificate Issuer:          'GlobalSign Primary Class 1 CA'.
Certificate Serial Number:   '0400 0000 0000 FA3D EEE9 D9'.
Certificate Valid From:     '22/01/2004' to '28/01/2009'.
```

The signer certificate 'GlobalSign PersonalSign Class 1 CA' is missing for the following certificate.

```
Microsoft Certificate Store: 'c:\ssl\client'.
Certificate Subject:         'wm.shakespeare@hamlet.com'.
Certificate Issuer:          'GlobalSign PersonalSign Class 1 CA'.
Certificate Serial Number:   '0100 0000 0001 0170 978B 1E'.
Certificate Valid From:     '14/01/2005' to '14/02/2005'.
```

Certificate chain checking has completed with some failures.
The Check Certificate Chains (amqccert) command has completed.

Return codes

- | | |
|---|---|
| 1 | amqccert command usage error |
| 2 | User not authorized to run amqccert command |
| 3 | WebSphere MQ Certificate Store file not found |
| 4 | WebSphere MQ Certificate Store file is empty |
| 5 | WebSphere MQ Certificate Store file cannot be opened |
| 6 | No memory to allocate tables for storing root/intermediate certificates |
| 7 | Certificate is either an orphan or has expired |
| 8 | Windows operation failed |

Related commands

amqtcert	Transfer certificates
----------	-----------------------

amqmdain (WebSphere MQ services control)

Purpose

The **amqmdain** command applies to WebSphere MQ for Windows only.

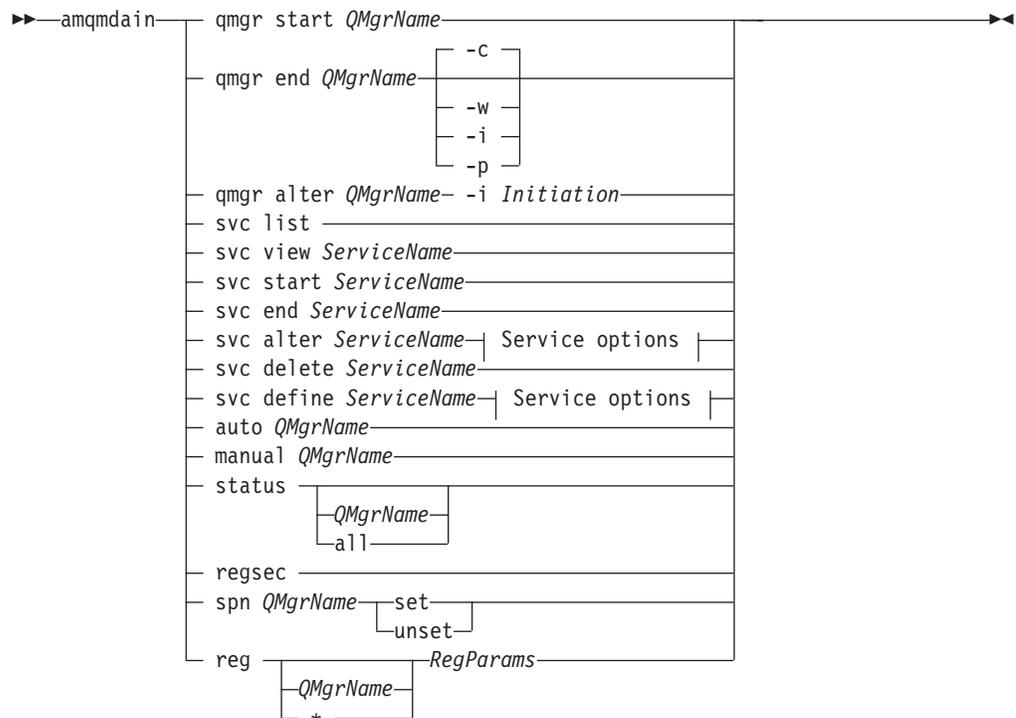
In WebSphere MQ Version 6.0 all WebSphere MQ services were migrated to WebSphere MQ service or listener objects, with the exception of ROOT custom services and queue manager services.

Use **amqmdain** to define and administer ROOT custom services, define and administer queue manager services, and perform other Windows specific administrative tasks.

Starting a queue manager service with **amqmdain** is not the same as using **strmqm** from the command line, because the WebSphere MQ service executes in a non-interactive session, running under a different user account.

To administer and define WebSphere MQ service and listener objects, use MQSC commands, PCF commands, or the WebSphere MQ Explorer.

Syntax



Service options:





Keywords and parameters

All parameters are required unless the description states they are optional.

In every case, *QMgrName* is the name of the queue manager service to which the command applies, and *ServiceName* is the name of the ROOT custom services to which the command applies.

qmgr start *QMgrName*

Starts a queue manager service.

This parameter can also be written in the form `start QMgrName`.

qmgr end *QMgrName*

Ends a queue manager service.

This parameter can also be written in the form `end QMgrName`.

-c Controlled (or quiesced) shutdown.

-w Wait shutdown.

-i Immediate shutdown.

-p Preemptive shutdown.

qmgr alter *QMgrName*

Alters a queue manager service.

-i *Initiation*

Specifies the initiation type. Possible values are:

auto Sets a queue manager service to automatic startup.

manual Sets a queue manager service to manual startup.

svc list

Displays a list of currently defined ROOT custom services.

svc view *ServiceName*

Displays detailed information for a ROOT custom service.

svc start *ServiceName*

Starts a ROOT custom service.

svc end *ServiceName*

Ends a ROOT custom service.

svc delete *ServiceName*

Deletes a ROOT custom service.

svc alter *ServiceName ServiceOptions*

Alter a ROOT custom service with the options specified in *ServiceOptions*.

svc define *ServiceName ServiceOptions*

Define a ROOT custom service with the options specified in *ServiceOptions*.

auto *QMgrName*

Sets a queue manager service to automatic startup.

manual *QMgrName*

Sets a queue manager service to manual startup.

status *QMgrName* | **all**

These parameters are optional.

- | | |
|--|---|
| If no parameter is supplied: | Displays the status of the WebSphere MQ services. |
| If a <i>QMgrName</i> is supplied: | Displays the status of the named queue manager service. |
| If the parameter <i>all</i> is supplied: | Displays the status of all ROOT custom services. |

regsec

Ensures that the security permissions assigned to the Registry keys are correct.

spn *QMgrName* **set** | **unset**

Allows you to set or unset the service principal name for a queue manager.

reg *QMgrName* | * *RegParams*

Parameters *QMgrName*, and * are optional.

- | | |
|--|---|
| If <i>RegParams</i> is specified alone: | Modifies queue manager configuration information in the Windows Registry related to the default queue manager. |
| If <i>QMgrName</i> and <i>RegParams</i> are specified: | Modifies queue manager configuration information in the Windows Registry related to the queue manager specified by <i>QMgrName</i> . |
| If * and <i>RegParams</i> are specified: | Modifies WebSphere MQ configuration information in the Windows Registry. |

The parameter, *RegParams*, specifies the Registry stanzas to change, and the changes that are to be made. *RegParams* takes one of the following forms:

- -c add -s *stanza* -v attribute=*value*
- -c remove -s *stanza* -v [attribute|*]
- -c display -s *stanza* -v [attribute|*]

If you are specifying queue manager configuration information, the valid values for *stanza* are:

```
XAResourceManager\name
ApiExitLocal\name
CHANNELS
ExitPath
Log
QueueManagerStartup
TCP
LU62
SPX
NetBios
Connection
QMErrorLog
Broker
```

If you are modifying WebSphere MQ configuration information, the valid values for *stanza* are:

```
ApiExitCommon\name
ApiExitTemplate\name
ACPI
AllQueueManagers
CHANNELS
DefaultQueueManager
LogDefaults
ExitProperties
```

amqmdain

The following are usage considerations:

- **amqmdain** does not validate the values you specify for *name*, *attribute*, or *value*.
- When you specify *add*, and an attribute already exists, it is modified.
- If a stanza does not exist, **amqmdain** creates it.
- When you specify *remove* or *display*, you can use the value *** to remove or display all attributes.
- If you use *remove* to delete the only attribute in a stanza, the stanza itself is deleted.
- Any modification you make to the Registry re-secures all WebSphere MQ Registry entries.

ServiceOptions

The options available when defining, or altering, a ROOT custom service.

-m *QMgrName*

The name of the associated queue manager.

If no parameter is supplied: The service is defined as a ROOT custom service with no associated queue manager.

If *QMgrName* is supplied: The queue manager specified by *QMgrName* is used.

If you omit this parameter, the service is defined as a ROOT custom service with no associated queue manager.

-i *Initiation*

Specifies the initiation type. Possible values are:

auto Sets the ROOT custom service to automatic startup.

manual Sets the ROOT custom service to manual startup.

If you omit this parameter, automatic startup is set.

-t *Service*

Specifies the ROOT custom service type. Possible values are:

process The ROOT custom service is not expected to run to completion. To end the ROOT custom service, issue the *svc end ServiceName* command.

command The ROOT custom service is expected to run to completion.

If you omit this parameter, *Service* is specified as **process**.

-s *command*

The command to execute when the ROOT custom service starts.

-e *command*

The command to execute when the ROOT custom service ends.

-x *Execution*

Specifies the execution type. Possible values are:

prefix The ROOT custom service starts before the associated queue manager starts.

suffix The ROOT custom service starts after the associated queue manager starts.

If you omit this parameter, *Execution* is specified as **suffix**.

Examples

The following example adds an XAResourceManager to queue manager TEST. The commands issued are:

```
amqmdain reg TEST -c add -s XAResourceManager\Sample -v SwitchFile=sf1
amqmdain reg TEST -c add -s XAResourceManager\Sample -v ThreadOfControl=THREAD
amqmdain reg TEST -c add -s XAResourceManager\Sample -v XAOpenString=openit
amqmdain reg TEST -c add -s XAResourceManager\Sample -v XACloseString=closeit
```

To display the values set by the commands above, use:

```
amqmdain reg TEST -c display -s XAResourceManager\Sample -v *
```

The display would look something like this:

```
0784726, 5639-B43 (C) Copyright IBM Corp. 1994, 2002. ALL RIGHTS RESERVED.
Displaying registry value for Queue Manager 'TEST'
Attribute = Name, Value = Sample
Attribute = SwitchFile, Value = sf1
Attribute = ThreadOfControl, Value = THREAD
Attribute = XAOpenString, Value = openit
Attribute = XACloseString, Value = closeit
```

To remove the XAResourceManager from queue manager TEST, use:

```
amqmdain reg TEST -c remove -s XAResourceManager\Sample -v *
```

Return codes

0	Command completed normally
-2	Syntax error
-3	Failed to initialize COM library
-4	Failed to initialize COM components
-7	Failed to configure service
-9	Unexpected Registry error
-10	Unable to access required service interface (IMQDService)
-11	Unable to access required service interface (ICustomService)
-12	Unable to access required service interface (ICustomServices)
-13	Unable to access required service interface (IUnknown)
-14	Specified service not found
-15	Specified service name already exists
-16	Failed to configure service principal name
-17	Failed to start service
-18	Failed to end service
-19	Failed to delete service
-20	Failed to store service definition
-21	Service initiation type could not be configured
-22	Service flags could not be configured
-23	Service flags could not be read
-24	Service dependency could not be configured
-25	Service start command could not be configured
-26	Service end command could not be configured
-27	Service name could not be configured

Notes:

1. If the `qmgr start QMgrName` command is issued, all return codes that can be returned with **strmqm**, can be returned here also. For a list of these return codes, see “`strmqm (start queue manager)`” on page 381.

amqmdain

-
2. If the `qmgr end QMgrName` command is issued, all return codes that can be returned with **endmqm**, can be returned here also. For a list of these return codes, see “endmqm (end queue manager)” on page 335.

Manual migration options:

```

|----- -p —Password—-----|----- -e —ExpTime—-----|----- -u —ClntLogonID— -i —ListNumber—-----|
|----- -m —QMgrName—-----|

```

Keywords and parameters

-a Specifies automatic migration.

When used in conjunction with the **-m** or **-c** parameters, it prepares the specified queue manager or client to automatically migrate the WebSphere MQ Certificate Store.

When used in conjunction with the **-l** parameter, it lists the contents of the registry entries for automatic migration.

-c *FileName* | *

FileName specifies the absolute (rather than relative) directory path name and filename (excluding the .sto suffix) of the client's WebSphere MQ Certificate Store. In manual migration, the **-c** parameter is not required.

FileName is used to identify a specific client WebSphere MQ Certificate Store. For automatic migration, the filename is stored in the registry and flagged as requiring automatic migration.

When the client connects to the queue manager, the key repository value (either MQSSLKEYR or the KeyRepository field of the MQSCO) being used by the client is compared against the list of stored filenames flagged as requiring automatic migration; if the values match then migration takes place. The filename is cleared from the registry list once successful migration has taken place.

-c * is used only in combination with the **-r** flag and specifies all client entries in the registry.

-e *ExpTime*

The expiration time (in days) of the GSKit key database password. The default is 60 days.

-g *Filename*

Use manual migration. The absolute (rather than relative) directory path name and filename (excluding the .kdb suffix) of a GSKit key database. The **-w** parameter must also be specified.

-l In combination with the **-c** *FileName* or **-m** *QMgrName* parameters, it lists the certificates in a WebSphere MQ Certificate Store.

In combination with the **-a** parameter, it lists the contents of the registry entries for automatic migration.

-m *QMgrName* | *

QMgrName specifies the name of an individual queue manager. * represents all queue managers.

When specifying manual migration of a queue manager certificate store, the **-m** *QMgrName* parameter is mandatory. This allows the correct label to be given to the assigned personal certificate when it is written to the GSKit key database file (see the description of the **-u** parameter for more details). The * value is not valid for manual migration.

When specifying automatic migration, the names of the source certificate store and the target key database file are derived from the queue manager's SSLKeyRepository attribute.

-p *Password*

The password for the GSKit key database. This must be specified for automatic or manual migration. The maximum password length is 255 bytes.

-r Remove the registry state information relating to automatic migration.

-u *ClntLogonID*

This parameter is only applicable when the command is used for manual migration of clients. The *-i ListNumber* parameter must also be specified.

In the WebSphere MQ Certificate Store there is usually one certificate assigned to the client. During migration, the copy of this certificate is modified before it is stored in the GSKit database.

The modification sets the certificate's Friendly Name attribute to the string `ibmwebspheremq`, followed in lower case by the client logon ID. The previous Friendly Name value, if any, is lost. This Friendly Name value becomes the label in the GSKit key database.

If neither **-u** nor **-m** are specified on manual migration, it is assumed to be a client migration. The *ClntLogonID* used is the userid used by the current **amqtcert** user to logon.

-i *ListNumber*

This parameter is only applicable when the command is used for manual migration of clients. The *-u ClntLogonID* parameter must also be specified.

This parameter is used to identify a specific personal certificate which is to have its GSKit label set to the value specified by the *-u ClntLogonID* parameter.

Prior to using **amqtcert** with *-i ListNumber* specified, you must execute **amqtcert** with *-l* specified to list the certificates in a WebSphere MQ Certificate Store. You must identify the required personal certificate from the list, then execute **amqtcert** again, specifying *-i ListNumber* with the required certificate number.

For example, after executing `amqtcert -l -c C:\SSL\Client\key` you might identify the following personal certificate from the list displayed as the required certificate:

```
Certificate 14
Certificate Type: Personal
Subject:          personalcert@ibm.com, personalcert@ibm.com
Issuer:          BE, GlobalSign nv-sa, PersonalSign Class 1 CA, GlobalSign
                  PersonalSign Class 1 CA
Valid From:      14/10/2004 to 14/11/2004
Certificate Usage: <All>
```

You will then execute **amqtcert** and specify *-i ListNumber* as *-i 14*.

ListNumber must be a number greater than 0.

If *ListNumber* references a valid personal certificate, which is not the currently assigned certificate, then:

- The assigned certificate is not modified.
- The assigned certificate is not given a label of the form `ibmwebspheremq<xxxxx>` in the GSKit key database file, and ceases to be assigned.

amqtcert

- The certificate referenced by *ListNumber* becomes the assigned certificate in the GSKit key database.

If *ListNumber* does not reference a valid personal certificate, then the command fails and no migration occurs for any certificates (personal or otherwise).

-w *FileName*

Use manual migration. *FileName* is the absolute (rather than relative) directory path name and filename (excluding the .sto suffix) of a WebSphere MQ Certificate Store. The **-g** parameter must also be specified.

Examples

Listing the contents of certificate stores

```
amqtcert -l -c C:\SSL\Client\key
```

Lists the contents of the client's WebSphere MQ Certificate Store.

```
amqtcert -l -m QM1
```

Lists the contents of the QM1 queue manager's WebSphere MQ Certificate Store.

Manually migrating certificate stores

```
amqtcert -g C:\SSL\Client\key -w C:\SSL\Client\key -p MyPassword
```

Manually migrates the client WebSphere MQ Certificate Store, specified by the **-w** parameter value, to the GSKit key database specified by the **-g** parameter value. It sets the password of the GSKit key database (called key.kdb) to MyPassword. It also sets the GSKit label (of the certificate which was assigned in the WebSphere MQ client store) to `ibmwebspheremq<mylogonid>`, where `<mylogonid>` stands for the logon id of the current user in lower case.

```
amqtcert -g C:\GSKitSSL\Client\key -w C:\SSL\Client\key -p MyPassword -u MyClientID -i 14
```

Manually migrates a client's WebSphere MQ Certificate Store, specified by the **-w** parameter value, to the GSKit key database specified by the **-g** parameter value. This command sets the password of the GSKit key database (named key.kdb) to MyPassword. It also sets the GSKit label of the certificate specified by **-i** to `ibmwebspheremq<mylogonid>`, where `<mylogonid>` stands for the logon id of the current user in lower case.

```
amqtcert -g "C:\Program Files\IBM\WebSphere MQ\Qmgrs\QM1\SSL\key" -w "C:\Program Files\IBM\WebSphere MQ\Qmgrs\QM1\SSL\key" -p MyPassword -m QM1
```

Manually migrates the WebSphere MQ Certificate Store for queue manager QM1, specified by the **-w** parameter value, to the GSKit key database specified by the **-g** parameter value. It sets the password of the GSKit key database (named key.kdb) to MyPassword. It also sets the GSKit label (of the certificate which was assigned to queue manager QM1) to `ibmwebspheremqmqm1`.

Automatically migrating certificate stores

```
amqtcert -a -p MyPassword -m QM1
```

Automatically migrates the WebSphere MQ Certificate Store for queue manager QM1, and sets the GSKit key database password to MyPassword

```
amqtcert -a -p MyPassword -c C:\SSL\Client\key
```

Automatically migrates the specified client WebSphere MQ Certificate Store, and sets the GSKit key database password to "MyPassword".

amqtcert -a -p MyPassword -m *

Automatically migrates the WebSphere MQ Certificate Stores for all queue managers, and sets the GSKit key database password to MyPassword.

Listing the contents of registry entries

amqtcert -l -a

Lists the contents of the registry entries for automatic migration.

Removing state information

amqtcert -r -c C:\SSL\Client\key

Removes the registry state information relating to automatic migration for the specified client WebSphere MQ Certificate Store.

amqtcert -r -c *

Removes the registry state information relating to automatic migration for all clients.

amqtcert -r -m QM1

Removes the registry state information relating to automatic migration for queue manager QM1.

amqtcert -r -m *

Removes the registry state information relating to automatic migration for all queue managers.

Return codes

1	Error accessing certificate store
2	Auto migration failed
3	Invalid argument combination
4	Certificate expired
5	Certificate import failed
6	Certificate is an orphan
7	Create file failed
8	Duplicate registry entry
9	WebSphere MQ Certificate Store file is empty
16	WebSphere MQ Certificate Store file found
17	WebSphere MQ Certificate Store file not found
18	GSKit add certificate failed
19	GSKit error
20	GSKit initialization error
21	GSKit add CA certificate error
22	Load library failed
23	No memory to allocate tables for migrating root/intermediate certificates
24	No memory
25	WebSphere MQ Certificate Store file cannot be opened
32	User not authorized to run amqtcert command
33	Windows operation failed
34	Windows export of personal certificate failed
35	GSKit create new key database error
36	Windows registry error
37	amqtcert command usage error
38	Queue manager name error
39	Unexpected system return code
40	Local mqm group not found
41	Invalid arguments
48	Bad argument

amqtcert

49 Invalid -i *ListNumber* parameter

Related commands

amqccert Check certificate chains

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 549. When a queue manager has been created, use the **strmqm** command to start it.

Syntax

```

▶▶crtmqm [ -c Text ] [ -d DefaultTransmissionQueue ]
      [ -h MaximumHandleLimit ] [ -lc ] [ -ll ] [ -ld LogPath ]
      [ -lf LogFilePages ] [ -lp LogPrimaryFiles ] [ -ls LogSecondaryFiles ]
      [ -q ] [ -g ApplicationGroup ] [ -t IntervalValue ]
      [ -u DeadLetterQueue ] [ -x MaximumUncommittedMessages ] [ -z ]
▶▶QMgrName

```

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 232.

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 223.

To calculate the size of the primary log files, see “Calculating the size of the log” on page 228.

-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 223.

To calculate the size of the secondary log files, see “Calculating the size of the log” on page 228.

- 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 30.

-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 **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.

crtmqm

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

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.

Examples

- The following command deletes the queue manager saturn.queue.manager.
dltmqm saturn.queue.manager
- The following command deletes the queue manager travel and also suppresses any messages caused by the command.
dltmqm -z travel

Related commands

crtmqm Create queue manager

dltmqm

strmqm
endmqm

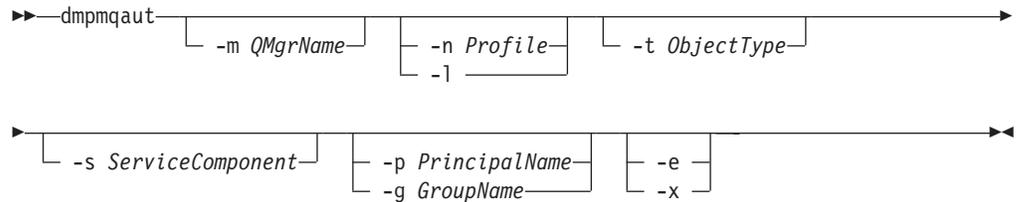
Start queue manager
End queue manager

dmpmqaut (dump authority)

Purpose

Use the **dmpmqaut** command to dump the current authorizations to a specified object.

Syntax



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. The profile name can be generic, using wildcard characters to specify a range of names as explained in “Using OAM generic profiles” on page 142.

-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.

dmpmqaut

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 “Principals and groups” on page 132.

-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*.

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:** 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

```

dmpmqaut

Related commands

dspmqaut	Display authority
setmqaut	Set or reset authority

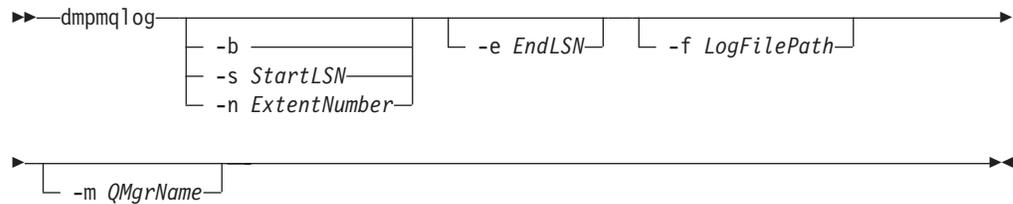
dmpmqlog (dump log)

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

dmpmqlog

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.

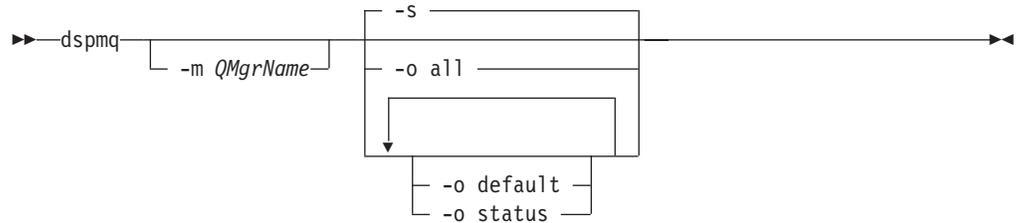
The queue manager must not be running when the **dmpmqlog** command is issued. Similarly, the queue manager must not be started while **dmpmqlog** is running.

dspmqr (display queue managers)

Purpose

Use the **dspmqr** 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.

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 “Installable services” on page 116, “Service components” on page 117, and Chapter 20, “Authorization service,” on page 409.

Syntax

```

▶▶ dspmqaout [-m QMgrName] -n ObjectName -t ObjectType
▶ [-g GroupName] [-p PrincipalName] [-s ServiceComponent]

```

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 setting 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 “Principals and groups” on page 132.

-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 21 shows the authorities that can be given to the different object types.

Table 21. 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	Yes	Yes	Yes	Yes
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

dspmqaout

Table 21. Specifying authorities for different object types (continued)

Authority	Queue	Process	Queue manager	Namelist	Auth info	Clntconn	Channel	Listener	Service
passid	Yes	No	No	No	No	No	No	No	No
set	Yes	No	No	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.
dlt	Delete the specified object using the appropriate command set.
dsp	Display the attributes of the specified object using the appropriate command set.
get	Retrieve a message from a queue by issuing an MQGET call.
inq	Make an inquiry on a specific queue by issuing an MQINQ call.
passall	Pass all context.
passid	Pass the identity context.
put	Put a message on a specific queue by issuing an MQPUT call.
set	Set attributes on a queue from the MQI by issuing an MQSET call.
setall	Set all context on a queue.
setid	Set the identity context on a queue.

The authorizations for administration operations, where supported, apply to these command sets:

- Control commands
- MQSC commands
- PCF commands

Return codes

0	Successful operation
36	Invalid arguments supplied
40	Queue manager not available
49	Queue manager stopping
69	Storage not available
71	Unexpected error
72	Queue manager name error
133	Unknown object name

```

145      Unexpected object name
146      Object name missing
147      Object type missing
148      Invalid object type
149      Entity name missing

```

Examples

- The following example shows a command to display the authorizations on queue manager saturn.queue.manager associated with user group staff:

```
dspmqaout -m saturn.queue.manager -t qmgr -g staff
```

The results from this command are:

Entity staff has the following authorizations for object:

```

get
browse
put
inq
set
connect
altusr
passid
passall
setid

```

- 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 20 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

dspmqfls

Notes:

1. The **dspmqfls** 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.

```
dspmqfls SYSTEM.ADMIN*
```
2. The following command displays file details for all processes with names beginning PROC defined on queue manager RADIUS.

```
dspmqfls -m RADIUS -t prcs PROC*
```

dspmqrte (WebSphere MQ display route application)

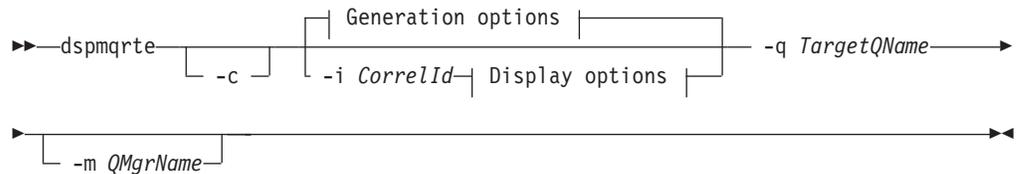
Purpose

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.

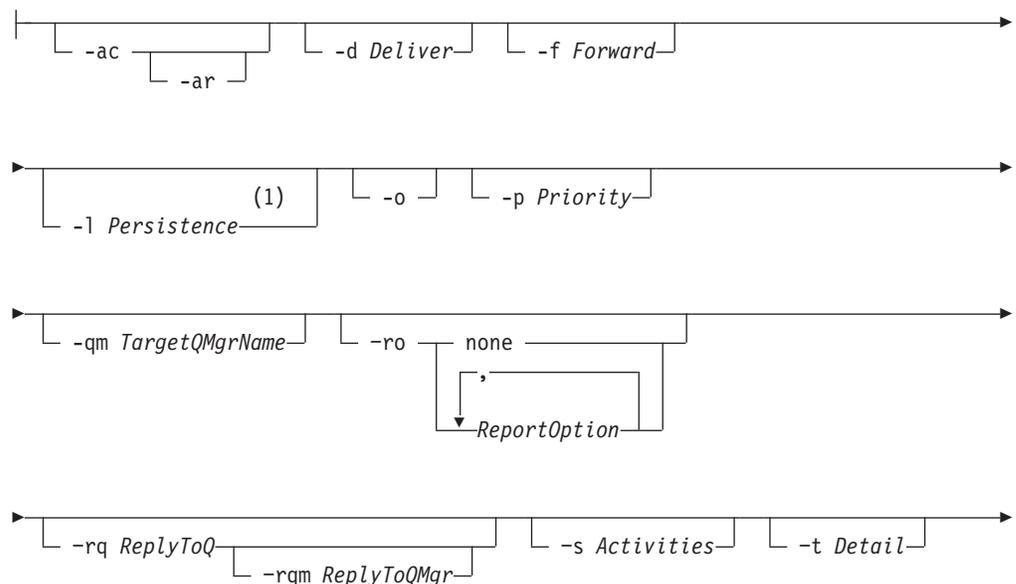
Note: To run a client application against a WebSphere MQ for z/OS queue manager, the client attachment feature must be installed.

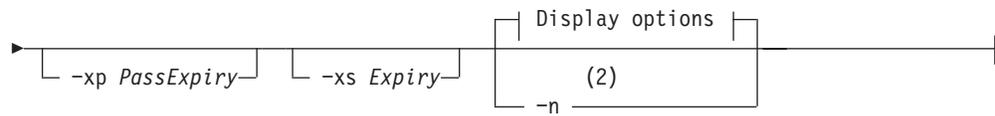
Use **dspmqrte** to help determine the route a message has taken through a queue manager network. The WebSphere MQ display route application generates and puts a trace-route message into a queue manager network. As the trace-route message travels through the queue manager network, activity information is recorded. When the trace-route message reaches its 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* book.

Syntax

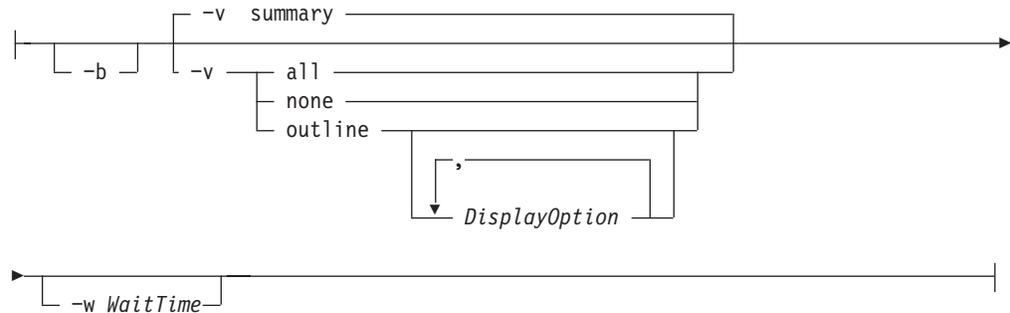


Generation options:





Display options:



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* 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* book.

If you do not specify this parameter, a trace-route reply message will **not** be 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*. 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 will not be 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 only forwarded to a queue manager that will honor the *Deliver* parameter from the *TraceRoute* PCF group.

If you do not specify this parameter, the trace-route message will only be forwarded to a queue manager that will honor 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 <i>-q TargetQName</i> . (MQPER_PERSISTENCE_AS_Q_DEF).

A trace-route reply message, or any report messages, returned will 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 will then apply. 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 The report option MQRO_PASS_DISCARD_AND_EXPIRY is specified in the message descriptor of the trace-route message.

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.

This is the default value.

no The report option MQRO_PASS_DISCARD_AND_EXPIRY is **not** specified.

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.

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 will only browse 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 will destructively get 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 <i>DisplayOption</i>	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:
	<ul style="list-style-type: none"> • The application name • The type of each operation • Any operation specific parameters
	Possible values for <i>DisplayOption</i> are:
activity	All non-PCF group parameters in <i>Activity</i> 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 <i>Message</i> PCF groups are displayed. When this value is specified, you cannot specify msgdelta.
msgdelta	All non-PCF group parameters in <i>Message</i> 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 <i>Operation</i> PCF groups are displayed.
traceroute	All non-PCF group parameters in <i>TraceRoute</i> 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 will wait for activity reports, or a trace-route reply message, to return to the specified reply-to queue.

dspmqrte

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* book.

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

dspmqr (display version information)

Purpose

Use the **dspmqr** command to display WebSphere MQ version and build information.

Syntax

```

>> dspmqr [-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.

dspmqver

Examples

The following command displays WebSphere MQ version and build information, using the default settings for *-p Components* and *-f Fields*:

```
dspmqver
```

The following command displays version and build information for the WebSphere MQ classes for Java:

```
dspmqver -p 2
```

The following command displays the name and version of the WebSphere MQ classes for Java Message Service:

```
dspmqver -p 4 -f 3
```

The following command displays the build level of the WebScale Distribution Hub:

```
dspmqver -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
```

endmqcsv

Related commands

strmqcsv

Start a command server

dspmqcsv

Display the status of a command server

endmq1sr (end listener)

Purpose

The **endmq1sr** command ends all listener processes for the specified queue manager.

Stop the queue manager before issuing the **endmq1sr** command.

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

endmqdnm (stop .NET monitor)

Purpose

The **endmqdnm** command applies to WebSphere MQ for Windows only.

Use the **endmqdnm** control command to stop a .NET monitor.

Syntax

```
►►—endmqdnm— -q QueueName —————►►  
└── -m QMgrName ─┘
```

Required parameters

-q *QueueName*

The name of the application queue that the .NET monitor is monitoring.

Optional parameters

-m *QMgrName*

The name of the queue manager that hosts the application queue.

If omitted, the default queue manager is used.

Return codes

0	Successful operation
36	Invalid arguments supplied
40	Queue manager not available
71	Unexpected error
72	Queue manager name error
133	Unknown object name error

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 will effect 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 will be 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.

endmqm

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

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>dltmqm</code>	Delete a queue manager

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).

When run for the first time, the graphical user interface must be configured. For instructions of how to do this, see “Configuring the GUI” on page 570.

Syntax

The syntax of this command follows:

►►—mqftapp—◄◄

Related commands

mqftrev	Receive file on server
mqftrevc	Receive file on client
mqftsnd	Send file from server
mqftsndc	Send file from client

mqftrcv (receive file on server)

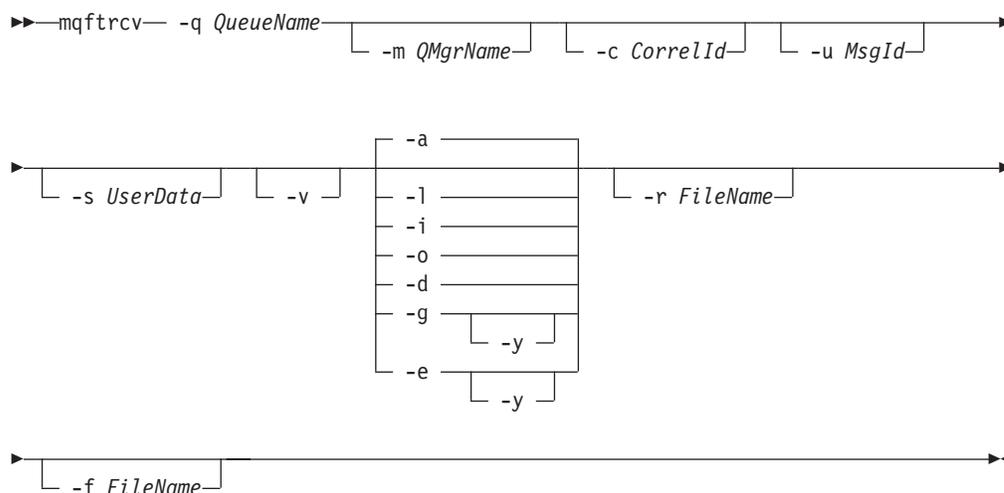
Purpose

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 "File status" on page 572.

- 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
mqftrcvc	Receive file on client
mqftsnd	Send file from server
mqftsndc	Send file from client

mqftrcvc (receive file on client)

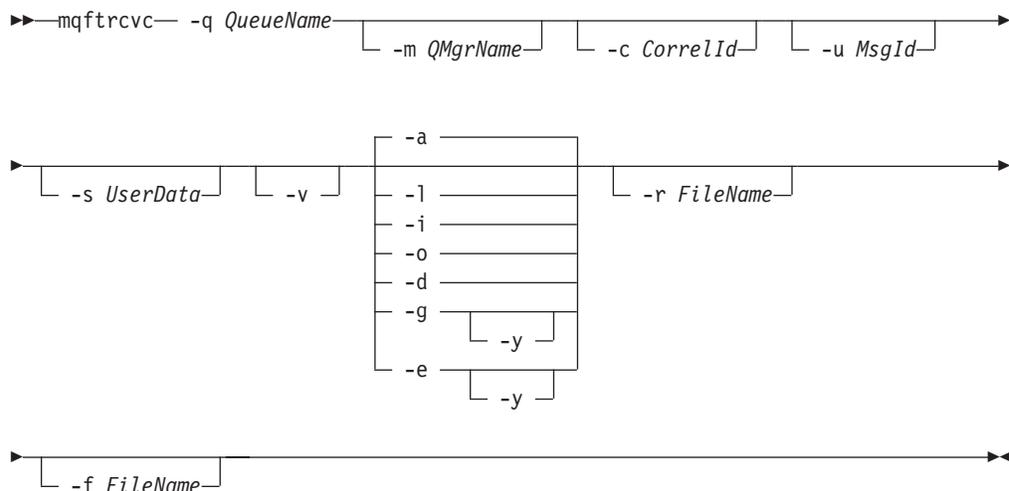
Purpose

The **mqftrcvc** command is available with the File Transfer Application on WebSphere MQ for Windows, and WebSphere MQ for Linux (x86 platform) clients only.

Use the **mqftrcvc** 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 "File status" on page 572.

-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

mqftrcvc

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:

```
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

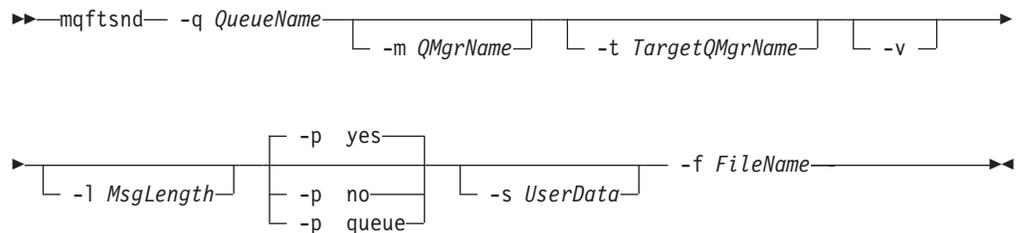
mqftsnd (send file from server)

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



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.

mqftsnd

The default value is 100000.

-p *yes*

Messages are persistent. This is the default value.

-p *no*

Messages are not persistent.

-p *yes*

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
mqftrcv	Receive file on server
mqftrcvc	Receive file on client
mqftsndc	Send file from client

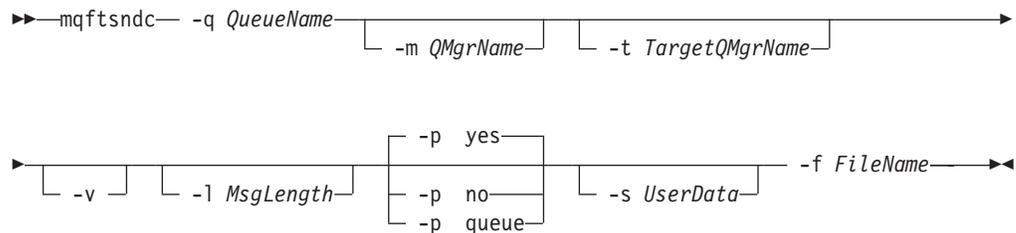
mqftsndc (send file from client)

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



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.

mqftsndc

The default value is 100000.

-p *yes*

Messages are persistent. This is the default value.

-p *no*

Messages are not persistent.

-p *yes*

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
▶—GenericObjName

```

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

`rccmqobj` 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 re-create an object, replay the log to re-create 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

Note: When using WebSphere MQ for UNIX systems, you need to prevent the shell from interpreting the meaning of special characters, for example, *****.

rcrmqobj

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 re-create 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 re-creates all local queues for the default queue manager:
`rcrmqobj -t ql *`
2. The following command re-creates all remote queues associated with queue manager store:
`rcrmqobj -m store -t qr *`

Related commands

`rcdmqing` Record an object in the log

rsvmqtrn

49	Queue manager stopping
69	Storage not available
71	Unexpected error
72	Queue manager name error
85	Transactions not known

Related commands

dspmqtrn	Display list of prepared transactions
----------	---------------------------------------

runmqchi (run channel initiator)

Purpose

Use the **runmqchi** command to run a channel initiator process. For more information about the use of this command, refer to *WebSphere MQ Intercommunication*.

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.

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 255.

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.

Return codes

0	Command completed normally
10	Command completed with unexpected results
20	An error occurred during processing

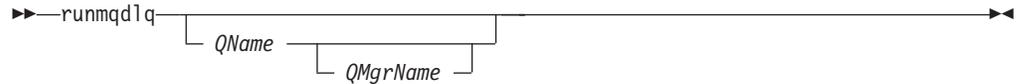
If return codes 10 or 20 are generated, review the error log of the associated queue manager for the error messages, and the system error log for records of problems that occur before the channel is associated with the queue manager.

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 196.

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.

runmqdnm (run .NET monitor)

Purpose

The **runmqdnm** command applies to WebSphere MQ for Windows only.

runmqdnm can be run from the command line, or as a triggered application.

Use the **runmqdnm** control command to start processing messages on an application queue with a .NET monitor.

Syntax

```

runmqdnm -q QueueName -a AssemblyName [-m QMgrName]
[-c ClassName] [-u UserParameter] [-s Syncpoint]
[-d Conversion] [-n MaxThreads] [-t Timeout]
[-b BackoutThreshold] [-r QueueName] [-p ContextOption]

```

Required parameters

- q *QueueName*
The name of the application queue to monitor.
- a *AssemblyName*
The name of the .NET assembly.

Optional parameters

- m *QMgrName*
The name of the queue manager that hosts the application queue.
If omitted, the default queue manager is used.
- c *ClassName*
The name of the .NET class that implements the IMQObjectTrigger interface.
This class must reside in the specified assembly.
If omitted, the specified assembly is searched to identify classes that implement the IMQObjectTrigger interface:
 - If one class is found, then *ClassName* takes the name of this class.
 - If no classes or multiple classes are found, then the .NET monitor is not started and a message is written to the console.
- u *UserData*
User defined data. This data is passed to the Execute method when the .NET monitor calls it. User data must be comprised of ASCII characters only, with no double-quotes, NULLs, or carriage returns.
If omitted, null is passed to the Execute method.
- s *Syncpoint*
Specifies whether syncpoint control is required when messages are retrieved

from the application queue. Possible values are:

YES	Messages are retrieved under syncpoint control (MQGMO_SYNCPOINT).
NO	Messages are not retrieved under syncpoint control (MQGMO_NO_SYNCPOINT).
PERSISTENT	Persistent messages are retrieved under syncpoint control (MQGMO_SYNCPOINT_IF_PERSISTENT).

If omitted, the value of *Syncpoint* is dependent on your transactional model:

- If distributed transaction coordination (DTC) is being used, then *Syncpoint* is specified as YES.
- If distributed transaction coordination (DTC) is not being used, then *Syncpoint* is specified as PERSISTENT.

-d *Conversion*

Specifies whether data conversion is required when messages are retrieved from the application queue. Possible values are:

YES	Data conversion is required (MQGMO_CONVERT).
NO	Data conversion is not required (no get message option specified).

If omitted, *Conversion* is specified as NO.

-n *MaxThreads*

The maximum number of active worker threads.

If omitted, *MaxThreads* is specified as 20.

-t *Timeout*

The time, in seconds, that the .NET monitor will wait for further messages to arrive on the application queue. If you specify -1, the .NET monitor will wait indefinitely.

If omitted when run from the command line, the .NET monitor will wait indefinitely.

If omitted when run as a triggered application, the .NET monitor will wait for 10 seconds.

-b *BackoutThreshold*

Specifies the backout threshold for messages retrieved from the application queue. Possible values are:

-1	The backout threshold is taken from the application queue attribute, BOTHRESH.
0	The backout threshold is not set.
1 or more	Explicitly sets the backout threshold.

If omitted, *BackoutThreshold* is specified as -1.

-r *QueueName*

The queue to which messages, whose backout count exceeds the backout threshold, are put.

If omitted, the value of *QueueName* is dependent on the value of the BOQNAME attribute from the application queue:

- If BOQNAME is non-blank, then *QueueName* takes the value of BOQNAME.

runmqdmn

- If BOQNAME is blank, then *QueueName* is specified as the queue manager dead letter queue. If a dead letter queue has not been assigned to the queue manager, then backout processing is not available.

-p *ContextOption*

Specifies whether context information from a message that is being backed out is passed to the backed out message. Possible values are:

NONE	No context information is passed.
IDENTITY	Identity context information is passed only.
ALL	All context information is passed.

If omitted, *ContextOption* is specified as ALL.

Return codes

0	Successful operation
36	Invalid arguments supplied
40	Queue manager not available
49	Queue manager stopping
71	Unexpected error
72	Queue manager name error
133	Unknown object name error

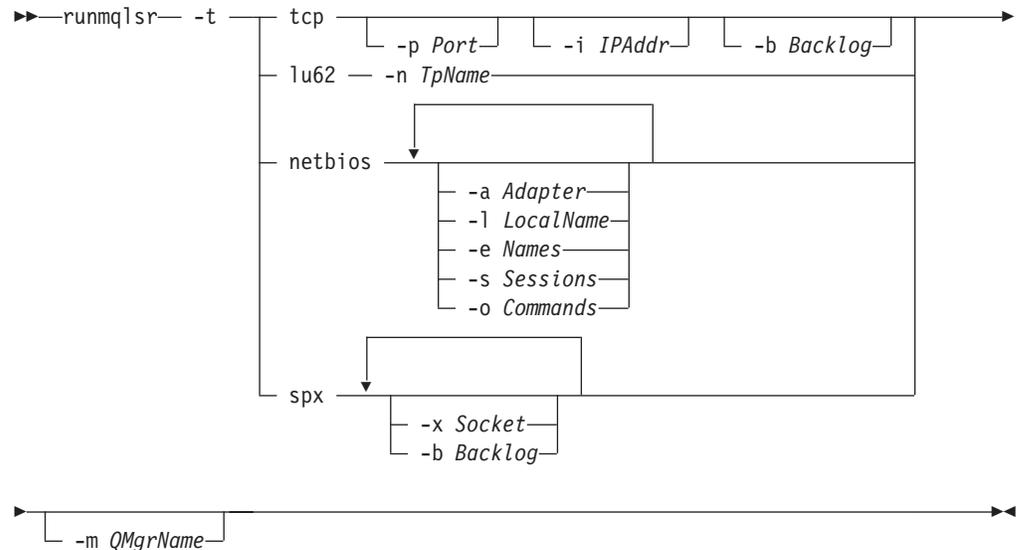
runmqtsr (run listener)

Purpose

Use the **runmqtsr** command to start a listener process.

This command is run synchronously and will wait 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.

runmq1sr

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 "LU62, NETBIOS, TCP, and SPX" on page 124 for a list of default values and further information.

Return codes

0	Command completed normally
10	Command completed with unexpected results
20	An error occurred during processing

Examples

The following command runs a listener on the default queue manager using the NetBIOS protocol. 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.

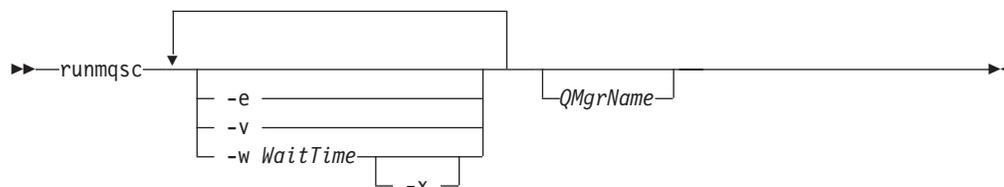
```
runmq1sr -t netbios -e 5 -s 5 -o 5
```

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*.

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

runmqsc

required channel and transmission queues set up for this. See "Preparing channels and transmission queues for remote administration" on page 68 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

The `runmqtmc` command is available on AIX clients only.

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*.

Once a trigger monitor has started, it will continuously monitor the specified initiation queue. The trigger monitor will not stop until the queue manager ends, see “`endmqm` (end queue manager)” on page 335. 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.

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*.

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*.

Once a trigger monitor has started, it will continuously monitor the specified initiation queue. The trigger monitor will not stop until the queue manager ends, see “endmqm (end queue manager)” on page 335. 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.

Return codes

0	Not used. The trigger monitor is designed to run continuously and therefore not to end. Hence a value of 0 would not be seen. The value is reserved.
10	Trigger monitor interrupted by an error.
20	Error; trigger monitor not run.

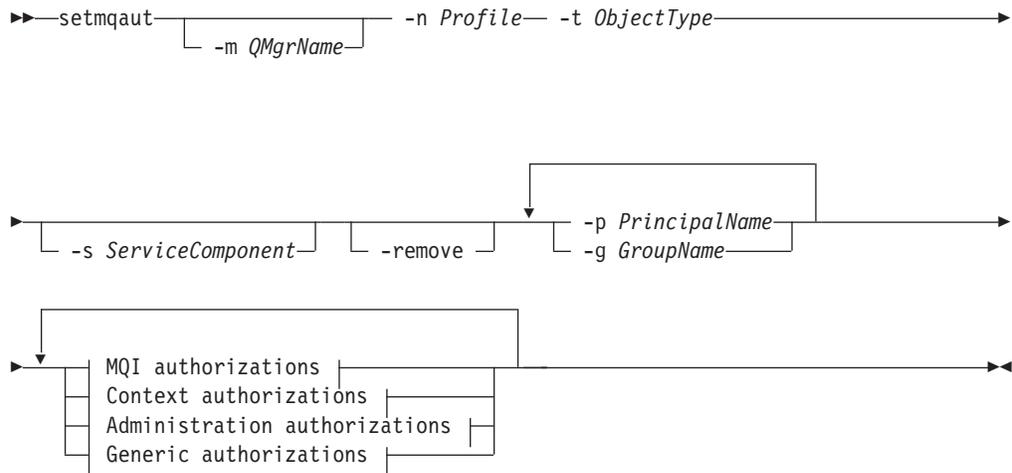
setmqaut (set or reset authority)

Purpose

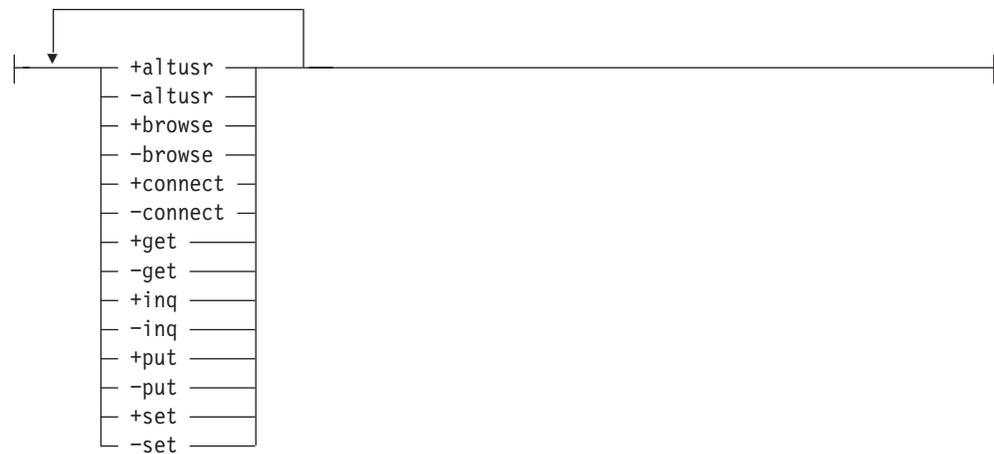
Use the **setmqaut** command to change the authorizations to a profile, object or class of objects. Authorizations can be granted to, or revoked from, any number of principals or groups.

For more information about authorization service components, see “Installable services” on page 116, “Service components” on page 117, and Chapter 20, “Authorization service,” on page 409.

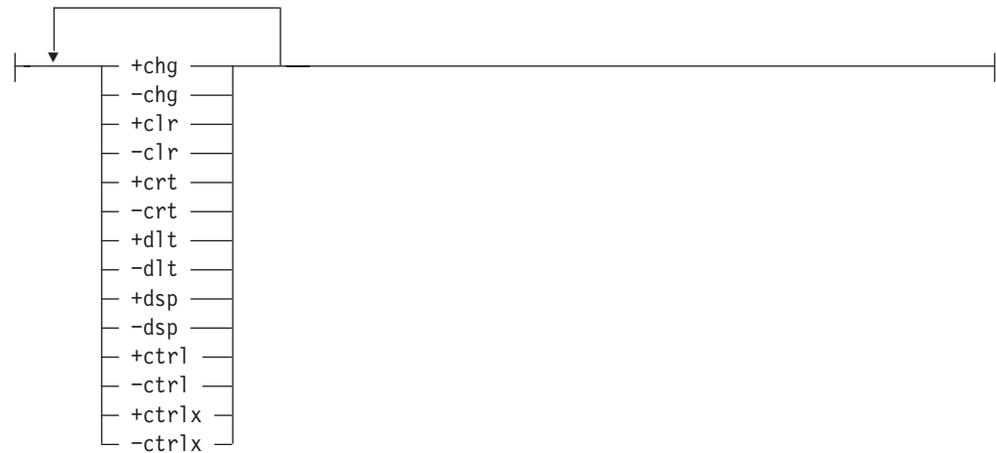
Syntax



MQI authorizations:



Context authorizations:

**Administration authorizations:****Generic authorizations:****Description**

Use `setmqaut` both to *set* an authorization, that is, give a user group or principal permission to perform an operation, and to *reset* an authorization, that is, remove the permission to perform an operation. You must specify the user groups and principals to which the authorizations apply, the queue manager, object type, and the profile name identifying the object or objects. You can specify any number of groups and principals in a single command.

Note: In WebSphere MQ for UNIX systems, if you specify a set of authorizations for a principal, the same authorizations are given to all principals in the same primary group.

The authorizations that can be given are categorized as follows:

- Authorizations for issuing MQI calls
- Authorizations for MQI context
- Authorizations for issuing commands for administration tasks
- Generic authorizations

Each authorization to be changed is specified in an authorization list as part of the command. Each item in the list is a string prefixed by + or -. For example, if you include +put in the authorization list, you give authority to issue **MQPUT** calls against a queue. Alternatively, if you include -put in the authorization list, you remove the authorization to issue **MQPUT** calls.

Authorizations can be specified in any order provided that they do not clash. For example, specifying allmqi with set causes a clash.

You can specify as many groups or authorizations as you require in a single command.

If a user ID is a member of more than one group, and if the groups have conflicting authorizations, the reset option does not override the set option, and the authorizations that apply are the union of the authorizations of each group to which that user ID belongs.

Required parameters

-t *ObjectType*

The type of object for which to change authorizations.

Possible values are:

authinfo	Authentication information object, for use with Secure Sockets Layer (SSL) channel security
channel or ch	A channel
clntconn or clcn	A client connection channel
lsr or listener	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
svrc 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. The profile name can be generic, using wildcard characters to specify a range of names as explained in "Using OAM generic profiles" on page 142.

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 “Principals and groups” on page 132.

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

Removes a profile. The authorizations associated with the profile no longer apply to WebSphere MQ objects with names that match the profile name specified.

Authorizations

The authorizations to be given or removed. Each item in the list is prefixed by a + indicating that authority is to be given, or a -, indicating that authority is to be removed.

For example, to give authority to issue an **MQPUT** call from the MQI, specify +put in the list. To remove authority to issue an **MQPUT** call, specify -put.

Table 22 shows the authorities that can be given to the different object types.

Table 22. 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	Yes	Yes	Yes	Yes
none	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
altusr	No	No	Yes	No	No	No	No	No	No

setmqaut

Table 22. Specifying authorities for different object types (continued)

Authority	Queue	Process	Queue manager	Namelist	Auth info	Clntconn	Channel	Listener	Service
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	No	No	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

1

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 have to 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.

1. Some of the authorities are part of +allmqi. Although they cannot be set individually, they can be reset individually using the **setmqaut** command.

Authorizations for commands

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

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 user group tango and the associated authorization list specifies that user group tango can:

- Issue **MQINQ** calls
 - Perform all administration operations on that object
2. In this example, the authorization list specifies that user group foxy:
 - Cannot issue any calls from the MQI to the specified queue
 - Can perform all administration operations on the specified queue

If the queue does not exist, the command fails.

setmqaut

```
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 will apply to any object with a name that matches the profile name.

```
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

setmqcrl (set certificate revocation list (CRL) LDAP server definitions)

Purpose

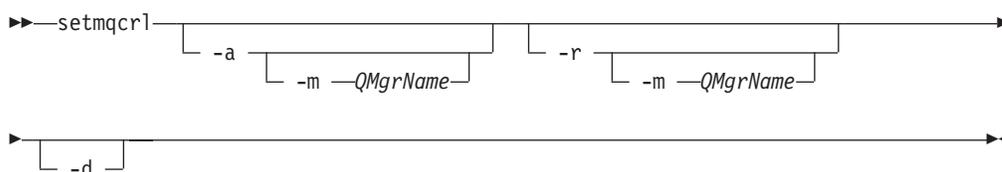
The **setmqcrl** command applies to WebSphere MQ for Windows only.

Use the **setmqcrl** command to configure and administer support for publishing CRL (certificate revocation list) LDAP definitions in an Active Directory.

A domain administrator must use this command, or **setmqscp**, initially to prepare the Active Directory for WebSphere MQ usage and to grant WebSphere MQ users and administrators the relevant authorities to access and update the WebSphere MQ Active Directory objects. You can also use the **setmqcrl** command to display all the currently configured CRL server definitions available on the Active Directory, that is, those definitions referred to by the queue manager's CRL namelist.

The only types of CRL servers supported are LDAP servers.

Syntax



Optional parameters

You must specify one of **-a** (add), **-r** (remove) or **-d** (display).

-a Adds the WebSphere MQ client connections Active Directory container, if it does not already exist. You must be a user with the appropriate privileges to create subcontainers in the *System* container of your domain. The WebSphere MQ folder is called CN=IBM-MQClientConnections. Do not delete this folder in any other way than by using the **setmqscp** command.

-d Displays the WebSphere MQ CRL server definitions.

-r Removes the WebSphere MQ CRL server definitions.

-m [* | qmgr]

Modifies the specified parameter (**-a** or **-r**) so that only the specified queue manager is affected. You must include this option with the **-a** parameter.

*** | qmgr**

* specifies that all queue managers are affected. This enables you to migrate a specific WebSphere MQ CRL server definitions file from one queue manager alone.

Examples

The following command creates the IBM-MQClientConnections folder and allocates the required permissions to WebSphere MQ administrators for the folder, and to child objects created subsequently. (In this, it is functionally equivalent to **setmqscp -a**.)

```
setmqcrl -a
```

setmqcrl

The following command migrates existing CRL server definitions from a local queue manager, Paint.queue.manager, to the Active Directory, **deleting any other CRL definitions from the Active Directory first**:

```
setmqcrl -a -m Paint.queue.manager
```

setmqscp (set service connection points)

Purpose

The **setmqscp** command applies to WebSphere MQ for Windows only.

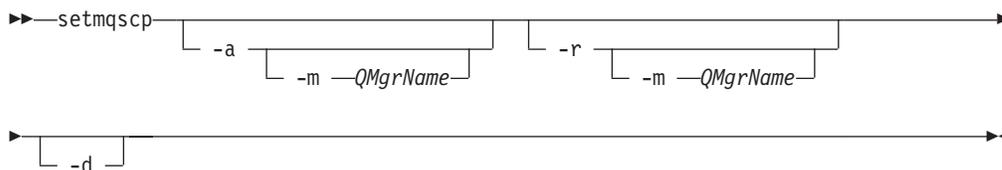
Use the **setmqscp** command to configure and administer support for publishing client connection channel definitions in an Active Directory.

Initially, this command is used by a domain administrator to:

- Prepare the Active Directory for WebSphere MQ use
- Grant WebSphere MQ users and administrators the relevant authorities to access and update the WebSphere MQ Active Directory objects

You can also use the **setmqscp** command to display all the currently configured client connection channel definitions available on the Active Directory.

Syntax



Optional parameters

You must specify one of -a (add), -r (remove) or -d (display).

- a** Adds the WebSphere MQ client connections Active Directory container, if it does not already exist. You must be a user with the appropriate privileges to create subcontainers in the *System* container of your domain. The WebSphere MQ folder is called CN=IBM-MQClientConnections. Do not delete this folder in any other way than by using the **setmqscp -r** command.
- d** Displays the service connection points.
- r** Removes the service connection points. If you omit -m, and no client connection definitions exist in the IBM-MQClientConnections folder, the folder itself is removed from the Active Directory.
- m [* | qmgr]**
Modifies the specified parameter (-a or -r) so that only the specified queue manager is affected.
 - * | qmgr**
* specifies that all queue managers are affected. This enables you to migrate a specific client connection table file from one queue manager alone, if required.

Examples

The following command creates the IBM-MQClientConnections folder and allocates the required permissions to WebSphere MQ administrators for the folder, and to child objects created subsequently:

```
setmqscp -a
```

setmqscp

The following command migrates existing client connection definitions from a local queue manager, `Paint.queue.manager`, to the Active Directory:

```
setmqscp -a -m Paint.queue.manager
```

The following command migrates all client connection definitions on the local server to the Active Directory:

```
setmqscp -a -m *
```


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

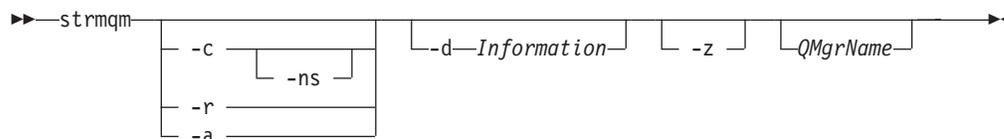
strmqm (start queue manager)

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 will show intermittent messages detailing the start up progress. For more information on these messages see *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 235.

-a Activate the specified backup queue manager. The backup queue manager is not started.

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 235.

-d Information

Specifies whether information messages are displayed. Possible values for *Information* follow:

strmqm

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` Suppresses error messages.

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

crtmqm	Create a queue manager
dltmqm	Delete a queue manager
endmqm	End a queue manager

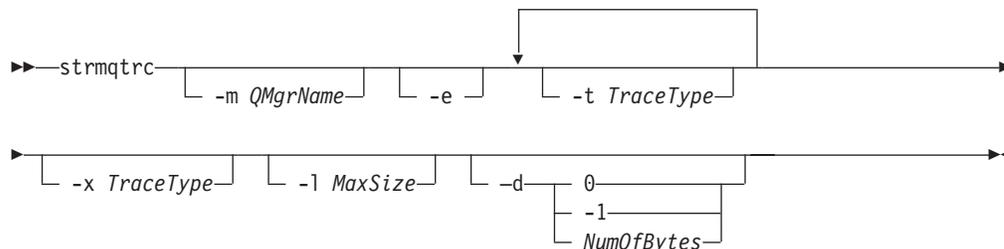
strmqtrc (Start trace)

Purpose

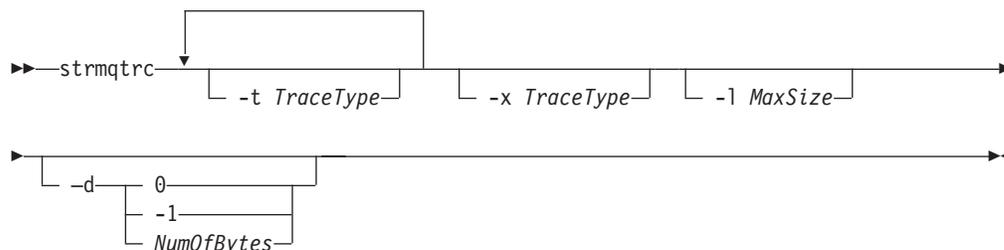
Use the **strmqtrc** command to enable tracing. This command can be run regardless of whether tracing is enabled. If tracing is already enabled, the trace options in effect are modified to those specified on the latest invocation of the command.

Syntax

The syntax of this command in WebSphere MQ for UNIX systems is as follows:



The syntax of this command in WebSphere MQ for Windows is as follows:



Description

You can request different levels of trace detail. For each *tracetype* value you specify, including `-t all`, specify either `-t parms` or `-t detail` to obtain the appropriate level of trace detail. If you do not specify either `-t parms` or `-t detail` for any particular trace type, only a default-detail trace is generated for that trace type.

You can use the `-x` flag with *tracetype* values to exclude those entry points you do not want to record. This is useful in reducing the amount of trace produced.

In WebSphere MQ for Windows, the output file is created in the `\<mqmwork>\trace` directory, where `<mqmwork>` is the directory selected to hold WebSphere MQ data files.

In WebSphere MQ for AIX, HP-UX, Solaris, and Linux, the output file is always created in the directory `/var/mqm/trace`.

For examples of trace data generated by this command see “Tracing” on page 259.

Optional parameters

-m *QMgrName*

The name of the queue manager to trace.

A queue manager name and the -m flag can be specified on the same command as the -e flag. If more than one trace specification applies to a given entity being traced, the actual trace includes all the specified options.

It is an error to omit the -m flag and queue manager name, unless you specify the -e flag.

This parameter is not valid in WebSphere MQ for Windows.

- e** Requests early tracing, making it possible to trace the creation or startup of a queue manager. If you include this flag, any process belonging to any component of any queue manager traces its early processing. The default is not to perform early tracing.

This parameter is not valid in WebSphere MQ for Windows.

-t *TraceType*

The points to trace and the amount of trace detail to record. By default **all** trace points are enabled and a default-detail trace is generated.

Alternatively, you can supply one or more of the options in the following list.

If you supply multiple trace types, each must have its own -t flag. You can include any number of -t flags, provided that each has a valid trace type associated with it.

It is not an error to specify the same trace type on multiple -t flags.

all	Output data for every trace point in the system (the default). The all parameter activates tracing at default detail level.
api	Output data for trace points associated with the MQI and major queue manager components.
commentary	Output data for trace points associated with comments in the WebSphere MQ components.
comms	Output data for trace points associated with data flowing over communications networks.
csdata	Output data for trace points associated with internal data buffers in common services.
csflows	Output data for trace points associated with processing flow in common services.
detail	Activate tracing at high-detail level for flow processing trace points.
Explorer	Output data for trace points associated with the WebSphere MQ Explorer.
lqmdata	Output data for trace points associated with internal data buffers in the local queue manager.
lqmflows	Output data for trace points associated with processing flow in the local queue manager.
otherdata	Output data for trace points associated with internal data buffers in other components.
otherflows	Output data for trace points associated with processing flow in other components.

parms	Activate tracing at default-detail level for flow processing trace points.
remotedata	Output data for trace points associated with internal data buffers in the communications component.
remoteflows	Output data for trace points associated with processing flow in the communications component.
servicedata	Output data for trace points associated with internal data buffers in the service component.
serviceflows	Output data for trace points associated with processing flow in the service component.
soap	Output data for trace points associated with WebSphere MQ Transport for SOAP.
ssl	Output data associated with using GSKit to enable Secure Sockets Layer (SSL) channel security.
	This parameter is not valid in WebSphere MQ for Windows.
versiondata	Output data for trace points associated with the version of WebSphere MQ running.

-x *TraceType*

The points **not** to trace. By default **all** trace points are enabled and a default-detail trace is generated. The trace points you can specify are those listed for the **-t** flag.

If you supply multiple trace types, each must have its own **-x** flag. You can include any number of **-x** flags, provided that each has a valid trace type associated with it.

-l *MaxSize*

The maximum size of a trace file (*AMQppppp.qq.TRC*) in millions of bytes. For example, if you specify a *MaxSize* of 1, the size of the trace is limited to 1 MB.

When a trace file reaches the specified maximum, it is renamed to *AMQppppp.qq.TRS* and a new *AMQppppp.qq.TRC* file is started. All trace files are restarted when the maximum limit is reached. If a previous copy of an *AMQppppp.qq.TRS* file exists, it is deleted.

The highest value that *MaxSize* can be set to is 2048 MB.

-d 0

Trace no user data.

-d -1

Trace all user data.

-d *NumOfBytes*

- For a communication trace; trace the specified number of bytes of data including the transmission segment header (TSH).
- For an MQPUT or MQGET call; trace the specified number of bytes of message data held in the message buffer.

Return codes

- AMQ7024 Non-valid arguments supplied to the command.
 AMQ8304 Nine concurrent traces (the maximum) already running.

strmqtrc

Examples

This command enables tracing of processing flow from common services and the local queue manager for a queue manager called QM1 in WebSphere MQ for UNIX systems. Trace data is generated at the default level of detail.

```
strmqtrc -m QM1 -t csflows -t lqmflows -t parms
```

This command disables tracing of SSL activity on a queue manager called QM1 in WebSphere MQ for UNIX systems. Other trace data is generated at the parms level of detail.

```
strmqtrc -m QM1 -x ssl -t parms
```

This command enables high-detail tracing of the processing flow for all components in WebSphere MQ for Windows:

```
strmqtrc -t all -t detail
```

Related commands

dspmqtrc	Display formatted trace output
endmqtrc	End trace

Chapter 18. Managing keys and certificates

To manage keys, certificates, and certificate requests, use one of the following:

The **gsk7cmd** command

The **gsk7cmd** command is available on UNIX systems only.

The **gsk7cmd** command provides functions that are similar to those of the iKeyman GUI, described in *WebSphere MQ Security*. The **gsk7cmd** command provides a shell script to run **iKeycmd**.

The **runmqckm** command

The **runmqckm** command is available on Windows systems only.

The **runmqckm** command provides functions that are similar to those of the iKeyman GUI, described in *WebSphere MQ Security*.

Use the **gsk7cmd** and **runmqckm** commands to do the following:

- Create the type of CMS key database files that WebSphere MQ requires
- Create certificate requests
- Import personal certificates
- Import CA certificates
- Manage self-signed certificates

Both the **gsk7cmd** and **runmqckm** commands execute an underlying WebSphere MQ component called **iKeycmd**. A default properties file, **ikeycmd.properties**, is provided as a sample file that you can modify.

This chapter contains the following sections:

- “Preparing to use the **gsk7cmd** command”
- “**gsk7cmd** and **runmqckm** commands”
- “**gsk7cmd** and **runmqckm** options” on page 392

Preparing to use the **gsk7cmd** command

The **gsk7cmd** command is available on UNIX systems only.

To run the **gsk7cmd** command line interface, set up environment variables as follows:

1. Set the **JAVA_HOME** environment variable:

AIX	export JAVA_HOME=/usr/mqm/ssl/jre
HP-UX	export JAVA_HOME=/opt/mqm/ssl
Linux	export JAVA_HOME=/opt/mqm/ssl/jre
Solaris	export JAVA_HOME=/opt/mqm/ssl

2. Set your **PATH** to include **/usr/bin**.

gsk7cmd and **runmqckm** commands

Each command specifies at least one *object*. Commands for PKCS #11 device operations might specify additional objects. Commands for key database, certificate, and certificate request objects also specify an *action*.

gsk7cmd and runmqckm commands

This section describes commands according to the object of the command. The object can be one of the following:

-keydb	Actions apply to a key database
-cert	Actions apply to a certificate
-certreq	Actions apply to a certificate request
-help	Displays help
-version	Displays version information

The following sections describe the actions that you can take on key database, certificate, and certificate request objects:

- “Commands for a CMS key database only”
- “Commands for CMS or PKCS #12 key databases”
- “Commands for cryptographic device operations” on page 390

See “gsk7cmd and runmqckm options” on page 392 for a description of the options on these commands.

Commands for a CMS key database only

-keydb -changepw

Change the password for a CMS key database:

```
-keydb -changepw -db filename -pw password -new_pw new_password  
-stash -expire days
```

-keydb -create

Create a CMS key database:

```
-keydb -create -db filename -pw password -type cms -expire days -stash
```

-keydb -stashpw

Stash the password of a CMS key database into a file:

```
-keydb -stashpw -db filename -pw password
```

-cert -getdefault

Get the default personal certificate:

```
-cert -getdefault -db filename -pw password
```

-cert -modify

Modify a certificate:

Note: Currently, the only field that can be modified is the Certificate Trust field.

```
-cert -modify -db filename -pw password -label label  
-trust enable | disable
```

-cert -setdefault

Set the default personal certificate:

```
-cert -setdefault -db filename -pw password -label label
```

Commands for CMS or PKCS #12 key databases

-keydb -changepw

Change the password for a key database:

```
-keydb -changepw -db filename -pw password -new_pw new_password -expire days
```

-keydb -convert

Convert the key database from one format to another:

```
-keydb -convert -db filename -pw password
      -old_format cms | pkcs12 -new_format cms
```

-keydb -create

Create a key database:

```
-keydb -create -db filename -pw password -type cms | pkcs12
```

-keydb -delete

Delete a key database:

```
-keydb -delete -db filename -pw password
```

-keydb -list

List currently-supported types of key database:

```
-keydb -list
```

-cert -add

Add a certificate from a file into a key database:

```
-cert -add -db filename -pw password -label label -file filename
      -format ascii | binary
```

-cert -create

Create a self-signed certificate:

```
-cert -create -db filename -pw password -label label -dn distinguished_name
      -size 1024 | 512 -x509version 3 | 1 | 2 -expire days
```

-cert -delete

Delete a certificate:

```
-cert -delete -db filename -pw password -label label
```

-cert -details

List the detailed information for a specific certificate:

```
-cert -details -db filename -pw password -label label
```

-cert -export

Export a personal certificate and its associated private key from a key database into a PKCS#12 file, or to another key database:

```
-cert -export -db filename -pw password -label label -type cms | pkcs12
      -target filename -target_pw password -target_type cms | pkcs12
```

-cert -extract

Extract a certificate from a key database:

```
-cert -extract -db filename -pw password -label label -target filename
      -format ascii | binary
```

-cert -import

Import a personal certificate from a key database:

```
-cert -import -file filename -pw password -type pkcs12 -target filename
      -target_pw password -target_type cms
```

-cert -list

List all certificates in a key database:

```
-cert -list all | personal | CA
      -db filename -pw password
```

-cert -receive

Receive a certificate from a file:

```
-cert -receive -file filename -db filename -pw password
      -format ascii | binary -default_cert yes | no
```

gsk7cmd and runmqckm commands

-cert -sign

Sign a certificate:

```
-cert -sign -file filename -db filename -pw password  
-label label -target filename  
-format ascii | binary -expire days
```

-certreq -create

Create a certificate request:

```
-certreq -create -db filename -pw password  
-label label -dn distinguished_name  
-size 1024 | 512 -file filename
```

-certreq -delete

Delete a certificate request:

```
-certreq -delete -db filename -pw password -label label
```

-certreq -details

List the detailed information of a specific certificate request:

```
-certreq -details -db filename -pw password -label label
```

List the detailed information about a certificate request and show the full certificate request:

```
-certreq -details -showOID -db filename  
-pw password -label label
```

-certreq -extract

Extract a certificate request from a certificate request database into a file:

```
-certreq -extract -db filename -pw password  
-label label -target filename
```

-certreq -list

List all certificate requests in the certificate request database:

```
-certreq -list -db filename -pw password
```

-certreq -recreate

Recreate a certificate request:

```
-certreq -recreate -dn distinguished_name -pw password  
-label label -target filename
```

Commands for cryptographic device operations

-keydb -changepw

Change the password for a cryptographic device:

```
-keydb -changepw -crypto module_name -tokenlabel token_label  
-pw password -new_pw new_password
```

-keydb -list

List currently-supported types of key database:

```
-keydb -list
```

-cert -add

Add a certificate from a file to a cryptographic device:

```
-cert -add -crypto module_name -tokenlabel token_label  
-pw password -label label -file filename -format ascii | binary
```

-cert -create

Create a self-signed certificate on a cryptographic device:

```
-cert -create -crypto module_name -tokenlabel token_label  
-pw password -label label -dn distinguished_name -size 1024 | 512  
-x509version 3 | 1 | 2 -default_cert no | yes -expire days
```

-cert -delete

Delete a certificate on a cryptographic device:

```
-cert -delete -crypto module_name -tokenlabel token_label
-pw password -label label
```

-cert -details

List the detailed information for a specific certificate on a cryptographic device:

```
-cert -details -crypto module_name -tokenlabel token_label
-pw password -label label
```

List the detailed information and show the full certificate for a specific certificate on a cryptographic device:

```
-cert -details -showOID -crypto module_name -tokenlabel token_label
-pw password -label label
```

-cert -extract

Extract a certificate from a key database:

```
-cert -extract -crypto module_name -tokenlabel token_label
-pw password -label label -target filename -format ascii | binary
```

-cert -import

Import a certificate to a cryptographic device with secondary key database support:

```
-cert -import -db filename -pw password -label label -type cms
-crypto module_name -tokenlabel token_label -pw password
-secondaryDB filename -secondaryDBpw password
```

Import a PKCS #12 certificate to a cryptographic device with secondary key database support:

```
-cert -import -file filename -pw password -type pkcs12
-crypto module_name -tokenlabel token_label -pw password
-secondaryDB filename -secondaryDBpw password
```

Note: You cannot import a certificate containing multiple OU (organizational unit) attributes in the distinguished name.

-cert -list

List all certificates on a cryptographic device:

```
-cert -list all | personal | CA
-crypto module_name -tokenlabel token_label -pw password
```

-cert -receive

Receive a certificate from a file to a cryptographic device with secondary key database support:

```
-cert -receive -file filename -crypto module_name -tokenlabel token_label
-pw password -default_cert yes | no
-secondaryDB filename -secondaryDBpw password -format ascii | binary
```

-certreq -create

Create a certificate request on a cryptographic device:

```
-certreq -create -crypto module_name -tokenlabel token_label
-pw password -label label -dn distinguished_name
-size 1024 | 512 -file filename
```

-certreq -delete

Delete a certificate request from a cryptographic device:

```
-certreq -delete -crypto module_name -tokenlabel token_label
-pw password -label label
```

gsk7cmd and runmqckm commands

-certreq -details

List the detailed information of a specific certificate request on a cryptographic device:

```
-certreq -details -crypto module_name -tokenlabel token_label
-pw password -label label
```

List the detailed information about a certificate request and show the full certificate request on a cryptographic device:

```
-certreq -details -showOID -crypto module_name -tokenlabel token_label
-pw password -label label
```

-certreq -extract

Extract a certificate request from a certificate request database on a cryptographic device into a file:

```
-certreq -extract -crypto module_name -tokenlabel token_label
-pw password -label label -target filename
```

-certreq -list

List all certificate requests in the certificate request database on a cryptographic device:

```
-certreq -list -crypto module_name -tokenlabel token_label
-pw password
```

gsk7cmd and runmqckm options

Table 23 lists the options that can be present on the command line. Note that the meaning of an option can depend on the object and action specified in the command.

Table 23. Options that can be used with gsk7cmd and runmqckm

Option	Description
-crypto	Name of the module to manage a PKCS #11 cryptographic device. The value after -crypto is optional if you specify the module name in the properties file
-db	Fully qualified path name of a key database.
-default_cert	Sets a certificate as the default certificate. The value can be yes or no. The default is no.
-dn	X.500 distinguished name. The value is a string enclosed in double quotes, for example "CN=John Smith,O=IBM ,OU=Test ,C=GB". Note that only the CN, O, and C attributes are required. Note: Avoid using multiple OU attributes in distinguished names when you create self-signed certificates. When you create such certificates, only the last entered OU value is accepted into the certificate.
-encryption	Strength of encryption used in certificate export command. The value can be strong or weak. The default is strong.
-expire	Expiration time in days of either a certificate or a database password. The defaults are 365 days for a certificate and 60 days for a database password.
-file	File name of a certificate or certificate request.
-format	Format of a certificate. The value can be <code>ascii</code> for Base64_encoded ASCII or <code>binary</code> for Binary DER data. The default is <code>ascii</code> .
-label	Label attached to a certificate or certificate request.
-new_format	New format of key database.

Table 23. Options that can be used with gsk7cmd and runmqckm (continued)

Option	Description
-new_pw	New database password.
-old_format	Old format of key database.
-pw	Password for the key database or PKCS#12 file.
-secondaryDB	Name of a secondary key database for PKCS #11 device operations.
-secondaryDBpw	Password for the secondary key database for PKCS #11 device operations.
-showOID	Displays the full certificate or certificate request.
-size	Key size. The value can be 512 or 1024. The default is 1024.
-stash	Stash the key database password to a file.
-target	Destination file or database.
-target_pw	Password for the key database if -target specifies a key database.
-target_type	Type of database specified by -target operand. See -type option for permitted values.
-tokenLabel	Label of a PKCS #11 cryptographic device.
-trust	Trust status of a CA certificate. The value can be enable or disable. The default is enable.
-type	Type of database. The value can be: <ul style="list-style-type: none"> • cms for a CMS key database • pkcs12 for a PKCS#12 file
-x509version	Version of X.509 certificate to create. The value can be 1, 2, or 3. The default is 3.

gsk7cmd and runmqckm options

Part 7. WebSphere MQ installable services and the API exit

Chapter 19. Installable services and components	401
Why installable services?	401
Functions and components	402
Entry-points	403
Return codes	403
Component data	403
Initialization	404
Primary initialization	404
Secondary initialization	404
Primary termination	404
Secondary termination	404
Configuring services and components	404
Service stanza format	405
Service stanza format for Windows systems	405
Service component stanza format	405
Creating your own service component	406
Using multiple service components	406
Example of using multiple components	407
What the components do	407
How the component is used	407
Omitting entry points when using multiple components	407
Example of entry points used with multiple components	407
Chapter 20. Authorization service	409
Object authority manager (OAM)	409
Defining the service to the operating system	409
Refreshing the OAM after changing a user's authorization	409
Migrating from MQSeries Version 5.1	410
Authorization service on UNIX systems	410
Configuring authorization service stanzas: UNIX systems	410
Authorization service on Windows systems	411
Configuring authorization service stanzas: Windows systems	411
Authorization service interface	412
Chapter 21. Name service	415
How the name service works	415
Name service interface	416
Chapter 22. Installable services interface reference information	419
How the functions are shown	420
Parameters and data types	420
MQZEP – Add component entry point	421
Syntax	421
Parameters	421
Hconfig (MQHCONFIG) – input	421
Function (MQLONG) – input	421
EntryPoint (PMQFUNC) – input	421
CompCode (MQLONG) – output	421
Reason (MQLONG) – output	421
C invocation	422
MQHCONFIG – Configuration handle	422
C declaration	422
PMQFUNC – Pointer to function	422
C declaration	422
MQZ_AUTHENTICATE_USER – Authenticate user	423
Syntax	423
Parameters	423
QMgrName (MQCHAR48) – input	423
SecurityParms (MQCSP) – input	423
ApplicationContext (MQZAC) – input	423
IdentityContext (MQZIC) – input/output	423
CorrelationPtr (MQPTR) – output	424
ComponentData (MQBYTE×ComponentDataLength) – input/output	424
Continuation (MQLONG) – output	424
CompCode (MQLONG) – output	424
Reason (MQLONG) – output	424
C invocation	424
MQZ_CHECK_AUTHORITY – Check authority	426
Syntax	426
Parameters	426
QMgrName (MQCHAR48) – input	426
EntityName (MQCHAR12) – input	426
EntityType (MQLONG) – input	426
ObjectName (MQCHAR48) – input	426
ObjectType (MQLONG) – input	427
Authority (MQLONG) – input	427
ComponentData (MQBYTE×ComponentDataLength) – input/output	429
Continuation (MQLONG) – output	429
CompCode (MQLONG) – output	429
Reason (MQLONG) – output	429
C invocation	430
MQZ_CHECK_AUTHORITY_2 – Check authority (extended)	431
Syntax	431
Parameters	431
QMgrName (MQCHAR48) – input	431
EntityData (MQZED) – input	431
EntityType (MQLONG) – input	431
ObjectName (MQCHAR48) – input	431
ObjectType (MQLONG) – input	432
Authority (MQLONG) – input	432
ComponentData (MQBYTE×ComponentDataLength) – input/output	434
Continuation (MQLONG) – output	434
CompCode (MQLONG) – output	434
Reason (MQLONG) – output	435
C invocation	435
MQZ_COPY_ALL_AUTHORITY – Copy all authority	436

Syntax.	436	ObjectName (MQCHAR48) – input	447
Parameters	436	ObjectType (MQLONG) – input	448
QMgrName (MQCHAR48) – input	436	Authority (MQLONG) – output	448
RefObjectName (MQCHAR48) – input	436	ComponentData	
ObjectName (MQCHAR48) – input	436	(MQBYTE×ComponentDataLength) –	
ObjectType (MQLONG) – input	436	input/output	448
ComponentData		Continuation (MQLONG) – output	448
(MQBYTE×ComponentDataLength) –		CompCode (MQLONG) – output.	449
input/output	437	Reason (MQLONG) – output	449
Continuation (MQLONG) – output	437	C invocation.	449
CompCode (MQLONG) – output.	437	MQZ_GET_AUTHORITY_2 – Get authority	
Reason (MQLONG) – output	437	(extended)	450
C invocation.	438	Syntax.	450
MQZ_DELETE_AUTHORITY – Delete authority	439	Parameters	450
Syntax.	439	QMgrName (MQCHAR48) – input	450
Parameters	439	EntityData (MQZED) – input	450
QMgrName (MQCHAR48) – input	439	EntityType (MQLONG) – input	450
ObjectName (MQCHAR48) – input	439	ObjectName (MQCHAR48) – input	450
ObjectType (MQLONG) – input	439	ObjectType (MQLONG) – input	451
ComponentData		Authority (MQLONG) – output	451
(MQBYTE×ComponentDataLength) –		ComponentData	
input/output	440	(MQBYTE×ComponentDataLength) –	
Continuation (MQLONG) – output	440	input/output	451
CompCode (MQLONG) – output.	440	Continuation (MQLONG) – output	451
Reason (MQLONG) – output	440	CompCode (MQLONG) – output.	452
C invocation.	441	Reason (MQLONG) – output	452
MQZ_ENUMERATE_AUTHORITY_DATA –		C invocation.	452
Enumerate authority data	442	MQZ_GET_EXPLICIT_AUTHORITY – Get explicit	
Syntax.	442	authority	453
Parameters	442	Syntax.	453
QMgrName (MQCHAR48) – input	442	Parameters	453
StartEnumeration (MQLONG) – input	442	QMgrName (MQCHAR48) – input	453
Filter (MQZAD) – input	442	EntityName (MQCHAR12) – input	453
AuthorityBufferLength (MQLONG) – input	443	EntityType (MQLONG) – input	453
AuthorityBuffer (MQZAD) – output.	443	ObjectName (MQCHAR48) – input	453
AuthorityDataLength (MQLONG) – output	443	ObjectType (MQLONG) – input	454
ComponentData		Authority (MQLONG) – output	454
(MQBYTE×ComponentDataLength) –		ComponentData	
input/output	443	(MQBYTE×ComponentDataLength) –	
Continuation (MQLONG) – output	443	input/output	454
CompCode (MQLONG) – output.	444	Continuation (MQLONG) – output	454
Reason (MQLONG) – output	444	CompCode (MQLONG) – output.	455
C invocation.	444	Reason (MQLONG) – output	455
MQZ_FREE_USER – Free user.	445	C invocation.	455
Syntax.	445	MQZ_GET_EXPLICIT_AUTHORITY_2 – Get	
Parameters	445	explicit authority (extended)	456
QMgrName (MQCHAR48) – input	445	Syntax.	456
FreeParms (MQZFP) – input	445	Parameters	456
ComponentData		QMgrName (MQCHAR48) – input	456
(MQBYTE×ComponentDataLength) –		EntityData (MQZED) – input	456
input/output	445	EntityType (MQLONG) – input	456
Continuation (MQLONG) – output	445	ObjectName (MQCHAR48) – input	456
CompCode (MQLONG) – output.	445	ObjectType (MQLONG) – input	457
Reason (MQLONG) – output	446	Authority (MQLONG) – output	457
C invocation.	446	ComponentData	
MQZ_GET_AUTHORITY – Get authority	447	(MQBYTE×ComponentDataLength) –	
Syntax.	447	input/output	457
Parameters	447	Continuation (MQLONG) – output	457
QMgrName (MQCHAR48) – input	447	CompCode (MQLONG) – output.	458
EntityName (MQCHAR12) – input	447	Reason (MQLONG) – output	458
EntityType (MQLONG) – input	447	C invocation.	458

MQZ_INIT_AUTHORITY – Initialize authorization service	459	EntityData (MQZED) – input	471
Syntax	459	EntityType (MQLONG) – input	471
Parameters	459	ObjectName (MQCHAR48) – input	471
Hconfig (MQHCONFIG) – input	459	ObjectType (MQLONG) – input	472
Options (MQLONG) – input	459	Authority (MQLONG) – input	472
QMgrName (MQCHAR48) – input	459	ComponentData (MQBYTE×ComponentDataLength) – input/output	472
ComponentDataLength (MQLONG) – input	459	Continuation (MQLONG) – output	472
ComponentData (MQBYTE×ComponentDataLength) – input/output	459	CompCode (MQLONG) – output	473
Version (MQLONG) – input/output	460	Reason (MQLONG) – output	473
CompCode (MQLONG) – output	460	C invocation	473
Reason (MQLONG) – output	460	MQZ_TERM_AUTHORITY – Terminate authorization service	474
C invocation	460	Syntax	474
MQZ_INQUIRE – Inquire authorization service	462	Parameters	474
Syntax	462	Hconfig (MQHCONFIG) – input	474
Parameters	462	Options (MQLONG) – input	474
QMgrName (MQCHAR48) – input	462	QMgrName (MQCHAR48) – input	474
SelectorCount (MQLONG) – input	462	ComponentData (MQBYTE×ComponentDataLength) – input/output	474
Selectors (MQLONG×SelectorCount) – input	462	CompCode (MQLONG) – output	475
IntAttrCount (MQLONG) – input	463	Reason (MQLONG) – output	475
IntAttrs (MQLONG×IntAttrCount) – output	463	C invocation	475
CharAttrCount (MQLONG) – input	463	MQZAC – Application context	476
CharAttrs (MQLONG×CharAttrCount) – output	463	Fields	476
SelectorReturned (MQLONG×SelectorCount) – input	463	StrucId (MQCHAR4)	476
ComponentData (MQBYTE×ComponentDataLength) – input/output	463	Version (MQLONG)	476
Continuation (MQLONG) – output	464	ProcessId (MQPID)	477
CompCode (MQLONG) – output	464	ThreadId (MQTID)	477
Reason (MQLONG) – output	464	AppName (MQCHAR28)	477
C invocation	465	UserId (MQCHAR12)	477
MQZ_REFRESH_CACHE – Refresh all authorizations	466	EffectiveUserId (MQCHAR12)	477
Syntax	466	Environment (MQLONG)	477
Parameters	466	CallerType (MQLONG)	477
C invocation	467	AuthenticationType (MQLONG)	478
MQZ_SET_AUTHORITY – Set authority	468	BindType (MQLONG)	478
Syntax	468	C declaration	478
Parameters	468	MQZAD – Authority data	478
QMgrName (MQCHAR48) – input	468	Fields	479
EntityName (MQCHAR12) – input	468	StrucId (MQCHAR4)	479
EntityType (MQLONG) – input	468	Version (MQLONG)	479
ObjectName (MQCHAR48) – input	468	ProfileName (MQCHAR48)	479
ObjectType (MQLONG) – input	469	ObjectType (MQLONG)	480
Authority (MQLONG) – input	469	Authority (MQLONG)	480
ComponentData (MQBYTE×ComponentDataLength) – input/output	469	EntityDataPtr (PMQZED)	480
Continuation (MQLONG) – output	469	EntityType (MQLONG)	480
CompCode (MQLONG) – output	470	Options (MQAUTHOPT)	481
Reason (MQLONG) – output	470	C declaration	481
C invocation	470	MQZED – Entity descriptor	483
MQZ_SET_AUTHORITY_2 – Set authority (extended)	471	Fields	483
Syntax	471	StrucId (MQCHAR4)	483
Parameters	471	Version (MQLONG)	483
QMgrName (MQCHAR48) – input	471	EntityNamePtr (PMQCHAR)	483
		EntityDomainPtr (PMQCHAR)	483
		SecurityId (MQBYTE40)	484
		CorrelationPtr (MQPTR)	484
		C declaration	484
		MQZIC – Identity context	485
		Fields	485

StrucId (MQCHAR4)	485	Continuation (MQLONG) – output	497
Version (MQLONG)	485	CompCode (MQLONG) – output	497
UserIdentifier (MQCHAR12)	486	Reason (MQLONG) – output	497
AccountingToken (MQBYTE32)	486	C invocation.	497
AppIdentityData (MQCHAR32)	486	MQZ_TERM_NAME – Terminate name service	499
C declaration	486	Syntax.	499
MQZFP – Free parameters	487	Parameters	499
Fields	487	Hconfig (MQHCONFIG) – input	499
StrucId (MQCHAR4)	487	Options (MQLONG) – input	499
Version (MQLONG)	487	QMgrName (MQCHAR48) – input	499
Reserved (MQBYTE8)	487	ComponentData	
CorrelationPtr (MQPTR)	487	(MQBYTE×ComponentDataLength) –	
C declaration	488	input/output	499
MQZ_DELETE_NAME – Delete name	489	CompCode (MQLONG) – output	500
Syntax.	489	Reason (MQLONG) – output	500
Parameters	489	C invocation.	500
QMgrName (MQCHAR48) – input	489	Chapter 23. API exits	501
QName (MQCHAR48) – input	489	Why use API exits.	501
ComponentData		How you use API exits	501
(MQBYTE×ComponentDataLength) –		How to configure WebSphere MQ for API exits	501
input/output	489	How to write an API exit	502
Continuation (MQLONG) – output	489	What happens when an API exit runs?	503
CompCode (MQLONG) – output	490	Configuring API exits	503
Reason (MQLONG) – output	490	Configuring API exits on UNIX systems	503
C invocation.	490	Attributes for all stanzas	504
MQZ_INIT_NAME – Initialize name service	491	Sample stanzas	505
Syntax.	491	Changing the configuration information	505
Parameters	491	Configuring API exits on Windows systems	506
Hconfig (MQHCONFIG) – input	491	Chapter 24. API exit reference information.	507
Options (MQLONG) – input	491	General usage notes	507
QMgrName (MQCHAR48) – input	491	MQACH – API exit chain header.	509
ComponentDataLength (MQLONG) – input	491	Fields	509
ComponentData		StrucId (MQCHAR4)	509
(MQBYTE×ComponentDataLength) –		Version (MQLONG)	510
input/output	491	StrucLength (MQLONG)	510
Version (MQLONG) – input/output	492	ChainAreaLength (MQLONG)	510
CompCode (MQLONG) – output	492	ExitInfoName (MQCHAR48)	511
Reason (MQLONG) – output	492	NextChainAreaPtr (PMQACH)	511
C invocation.	492	C declaration	511
MQZ_INSERT_NAME – Insert name	494	MQAXC – API exit context.	512
Syntax.	494	Fields	512
Parameters	494	StrucId (MQCHAR4)	512
QMgrName (MQCHAR48) – input	494	Version (MQLONG)	512
QName (MQCHAR48) – input	494	Environment (MQLONG)	513
ResolvedQMgrName (MQCHAR48) – input	494	UserId (MQCHAR12)	513
ComponentData		SecurityId (MQBYTE40)	513
(MQBYTE×ComponentDataLength) –		ConnectionName (MQCHAR264)	514
input/output	494	LongMCAUserIdLength (MQLONG)	514
Continuation (MQLONG) – output	494	LongRemoteUserIdLength (MQLONG)	514
CompCode (MQLONG) – output	495	LongMCAUserIdPtr (MQPTR)	514
Reason (MQLONG) – output	495	LongRemoteUserIdPtr (MQPTR)	514
C invocation.	495	AppIName (MQCHAR28)	514
MQZ_LOOKUP_NAME – Lookup name	496	AppIType (MQLONG)	514
Syntax.	496	ProcessId (MQPID)	515
Parameters	496	ThreadId (MQTID)	515
QMgrName (MQCHAR48) – input	496	C declaration	515
QName (MQCHAR48) – input	496	MQAXP – API exit parameter	516
ResolvedQMgrName (MQCHAR48) – output	496	Fields	516
ComponentData		StrucId (MQCHAR4)	516
(MQBYTE×ComponentDataLength) –			
input/output	496		

Version (MQLONG)	516	pExitContext (PMQAXC) – input/output	530
ExitId (MQLONG).	517	pHconn (PMQHCONN) – input/output	530
ExitReason (MQLONG)	517	pCompCode (PMQLONG) – input/output	530
ExitResponse (MQLONG)	518	pReason (PMQLONG) – input/output	530
ExitResponse2 (MQLONG)	519	C invocation.	530
Feedback (MQLONG)	520	MQ_CONN_EXIT – Connect queue manager	
APICallerType (MQLONG).	520	(extended)	531
ExitUserArea (MQBYTE16)	520	Syntax.	531
ExitData (MQCHAR32)	521	Parameters	531
ExitInfoName (MQCHAR48)	521	pExitParms (PMQAXP) – input/output.	531
ExitPDArea (MQBYTE48)	521	pExitContext (PMQAXC) – input/output	531
QMgrName (MQCHAR48)	521	pQMgrName (MQCHAR48) – input/output	531
ExitChainAreaPtr (PMQACH)	521	ppConnectOpts (PPMQCNO) – input/output	531
Hconfig (MQHCONFIG)	522	ppHconn (PPMQHCONN) – input/output	531
Function (MQLONG).	522	pCompCode (PMQLONG) – input/output	531
C declaration	523	pReason (PMQLONG) – input/output	531
MQXEP – Register entry point.	524	Usage notes	531
Syntax.	524	C invocation.	532
Parameters	524	MQ_DISC_EXIT – Disconnect queue manager	533
Hconfig (MQHCONFIG) – input	524	Syntax.	533
ExitReason (MQLONG) – input	524	Parameters	533
Function (MQLONG) – input	524	pExitParms (PMQAXP) – input/output.	533
EntryPoint (PMQFUNC) – input	525	pExitContext (PMQAXC) – input/output	533
Reserved (MQPTR) – input.	525	ppHconn (PPMQHCONN) – input/output	533
pCompCode (PMQLONG) – output	525	pCompCode (PMQLONG) – input/output	533
pReason (PMQLONG) – output	526	pReason (PMQLONG) – input/output	533
C invocation.	526	C invocation.	533
MQ_BACK_EXIT – Back out changes	527	MQ_GET_EXIT – Get message	534
Syntax.	527	Syntax.	534
Parameters	527	Parameters	534
pExitParms (PMQAXP) – input/output.	527	pExitParms (PMQAXP) – input/output.	534
pExitContext (PMQAXC) – input/output	527	pExitContext (PMQAXC) – input/output	534
pHconn (PMQHCONN) – input/output	527	pHconn (PMQHCONN) – input/output	534
pCompCode (PMQLONG) – input/output	527	pHobj (PMQHOBJ) – input/output	534
pReason (PMQLONG) – input/output	527	ppMsgDesc (PPMQMD) – input/output	534
C invocation.	527	ppGetMsgOpts (PPMQGMO) – input/output	534
MQ_BEGIN_EXIT – Begin unit of work	528	pBufferLength (PMQLONG) – input/output	534
Syntax.	528	ppBuffer (PPMQVOID) – input/output.	534
Parameters	528	ppDataLength (PPMQLONG) – input/output	534
pExitParms (PMQAXP) – input/output.	528	pCompCode (PMQLONG) – input/output	534
pExitContext (PMQAXC) – input/output	528	pReason (PMQLONG) – input/output	534
pHconn (PMQHCONN) – input/output	528	Usage notes	534
ppBeginOptions (PPMQBO) – input/output	528	C invocation.	535
pCompCode (PMQLONG) – input/output	528	MQ_INIT_EXIT – Initialize exit environment	536
pReason (PMQLONG) – input/output	528	Syntax.	536
C invocation.	528	Parameters	536
MQ_CLOSE_EXIT – Close object	529	pExitParms (PMQAXP) – input/output.	536
Syntax.	529	pExitContext (PMQAXC) – input/output	536
Parameters	529	pCompCode (PMQLONG) – input/output	536
pExitParms (PMQAXP) – input/output.	529	pReason (PMQLONG) – input/output	536
pExitContext (PMQAXC) – input/output	529	Usage notes	536
pHconn (PMQHCONN) – input/output	529	C invocation.	536
ppHobj (PPMQHOBJ) – input/output	529	MQ_INQ_EXIT – Inquire object attributes	537
pOptions (PMQLONG) – input/output.	529	Syntax.	537
pCompCode (PMQLONG) – input/output	529	Parameters	537
pReason (PMQLONG) – input/output	529	pExitParms (PMQAXP) – input/output.	537
C invocation.	529	pExitContext (PMQAXC) – input/output	537
MQ_COMMIT_EXIT – Commit changes.	530	pHconn (PMQHCONN) – input/output	537
Syntax.	530	pHobj (PMQHOBJ) – input/output	537
Parameters	530	pSelectorCount (PMQLONG) – input/output	537
pExitParms (PMQAXP) – input/output.	530	ppSelectors (PPMQLONG) – input/output	537

pIntAttrCount (PMQLONG) – input/output	537	pCharAttrLength (PMQLONG) –	
ppIntAttrs (PPMQLONG) – input/output	537	input/output 544
pCharAttrLength (PMQLONG) –		ppCharAttrs (PPMQCHAR) – input/output	544
input/output 537	pCompCode (PMQLONG) – input/output	544
ppCharAttrs (PPMQCHAR) – input/output	537	pReason (PMQLONG) – input/output	. . . 544
pCompCode (PMQLONG) – input/output	537	C invocation. 544
pReason (PMQLONG) – input/output	. . . 537	MQ_TERM_EXIT – Terminate exit environment	546
C invocation. 537	Syntax. 546
MQ_OPEN_EXIT – Open object 539	Parameters 546
Syntax. 539	pExitParms (PMQAXP) – input/output.	. . . 546
Parameters 539	pExitContext (PMQAXC) – input/output	. . . 546
pExitParms (PMQAXP) – input/output.	. . . 539	pCompCode (PMQLONG) – input/output	546
pExitContext (PMQAXC) – input/output	. . . 539	pReason (PMQLONG) – input/output	. . . 546
pHconn (PMQHCONN) – input/output	. . . 539	Usage notes 546
ppObjDesc (PPMQOD) – input/output.	. . . 539	C invocation. 546
pOptions (PMQLONG) – input/output.	. . . 539	MQ_PUT_EXIT – Put message. 540
ppHobj (PPMQHOBJ) – input/output	. . . 539	Syntax. 540
pCompCode (PMQLONG) – input/output	539	Parameters 540
pReason (PMQLONG) – input/output	. . . 539	pExitParms (PMQAXP) – input/output.	. . . 540
C invocation. 539	pExitContext (PMQAXC) – input/output	. . . 540
MQ_PUT_EXIT – Put message. 540	pHconn (PMQHCONN) – input/output	. . . 540
Syntax. 540	pHobj (PMQHOBJ) – input/output 540
Parameters 540	ppMsgDesc (PPMQMD) – input/output	. . . 540
pExitParms (PMQAXP) – input/output.	. . . 540	ppPutMsgOpts (PPMQPMO) – input/output	540
pExitContext (PMQAXC) – input/output	. . . 540	pBufferLength (PMQLONG) – input/output	540
pHconn (PMQHCONN) – input/output	. . . 540	ppBuffer (PPMQVOID) – input/output.	. . . 540
pHobj (PMQHOBJ) – input/output 540	pCompCode (PMQLONG) – input/output	540
ppMsgDesc (PPMQMD) – input/output	. . . 540	pReason (PMQLONG) – input/output	. . . 540
ppPutMsgOpts (PPMQPMO) – input/output	540	Usage notes 540
pBufferLength (PMQLONG) – input/output	540	C invocation. 540
ppBuffer (PPMQVOID) – input/output.	. . . 540	MQ_PUT1_EXIT – Put one message. 542
pCompCode (PMQLONG) – input/output	540	Syntax. 542
pReason (PMQLONG) – input/output	. . . 540	Parameters 542
Usage notes 540	pExitParms (PMQAXP) – input/output.	. . . 542
C invocation. 540	pExitContext (PMQAXC) – input/output	. . . 542
MQ_PUT1_EXIT – Put one message. 542	pHconn (PMQHCONN) – input/output	. . . 542
Syntax. 542	ppObjDesc (PPMQOD) – input/output.	. . . 542
Parameters 542	ppMsgDesc (PPMQMD) – input/output	. . . 542
pExitParms (PMQAXP) – input/output.	. . . 542	ppPutMsgOpts (PPMQPMO) – input/output	542
pExitContext (PMQAXC) – input/output	. . . 542	pBufferLength (PMQLONG) – input/output	542
pHconn (PMQHCONN) – input/output	. . . 542	ppBuffer (PPMQVOID) – input/output.	. . . 542
ppObjDesc (PPMQOD) – input/output.	. . . 542	pCompCode (PMQLONG) – input/output	542
ppMsgDesc (PPMQMD) – input/output	. . . 542	pReason (PMQLONG) – input/output	. . . 542
ppPutMsgOpts (PPMQPMO) – input/output	542	C invocation. 542
pBufferLength (PMQLONG) – input/output	542	MQ_SET_EXIT – Set object attributes 544
ppBuffer (PPMQVOID) – input/output.	. . . 542	Syntax. 544
pCompCode (PMQLONG) – input/output	542	Parameters 544
pReason (PMQLONG) – input/output	. . . 542	pExitParms (PMQAXP) – input/output.	. . . 544
C invocation. 542	pExitContext (PMQAXC) – input/output	. . . 544
MQ_SET_EXIT – Set object attributes 544	pHconn (PMQHCONN) – input/output	. . . 544
Syntax. 544	pHobj (PMQHOBJ) – input/output 544
Parameters 544	pSelectorCount (PMQLONG) – input/output	544
pExitParms (PMQAXP) – input/output.	. . . 544	ppSelectors (PPMQLONG) – input/output	544
pExitContext (PMQAXC) – input/output	. . . 544	pIntAttrCount (PMQLONG) – input/output	544
pHconn (PMQHCONN) – input/output	. . . 544	ppIntAttrs (PPMQLONG) – input/output	544
pHobj (PMQHOBJ) – input/output 544		
pSelectorCount (PMQLONG) – input/output	544		
ppSelectors (PPMQLONG) – input/output	544		
pIntAttrCount (PMQLONG) – input/output	544		
ppIntAttrs (PPMQLONG) – input/output	544		

Chapter 19. Installable services and components

This chapter introduces the installable services and the functions and components associated with them. We document the interface to these functions so that you, or software vendors, can supply components.

The chapter includes:

- “Why installable services?”
- “Functions and components” on page 402
- “Initialization” on page 404
- “Configuring services and components” on page 404
- “Creating your own service component” on page 406
- “Using multiple service components” on page 406

The installable services interface is described in Chapter 22, “Installable services interface reference information,” on page 419.

Why installable services?

The main reasons for providing WebSphere MQ installable services are:

- To provide you with the flexibility of choosing whether to use components provided by WebSphere MQ products, or replace or augment them with others.
- To allow vendors to participate, by providing components that might use new technologies, without making internal changes to WebSphere MQ products.
- To allow WebSphere MQ to exploit new technologies faster and cheaper, and so provide products earlier and at lower prices.

Installable services and *service components* are part of the WebSphere MQ product structure. At the center of this structure is the part of the queue manager that implements the function and rules associated with the Message Queue Interface (MQI). This central part requires a number of service functions, called *installable services*, in order to perform its work. The installable services are:

- Authorization service
- Name service

Each installable service is a related set of functions implemented using one or more *service components*. Each component is invoked using a properly-architected, publicly-available interface. This enables independent software vendors and other third parties to provide installable components to augment or replace those provided by the WebSphere MQ products. Table 24 summarizes the services and components that can be used on the WebSphere MQ Version 6.0 platforms.

Table 24. Installable service components summary

Installable service	Supplied component	Function	Requirements
Authorization service	Object Authority Manager (OAM)	Provides authorization checking on commands and MQI calls. Users can write their own component to augment or replace the OAM.	(Appropriate platform authorization facilities are assumed)

Installable services

Table 24. Installable service components summary (continued)

Installable service	Supplied component	Function	Requirements
Name service	None	<ul style="list-style-type: none">User defined <p>Note: Shared queues must have their <i>Scope</i> attribute set to CELL.</p>	<ul style="list-style-type: none">A third-party or user-written name manager

Functions and components

Each service consists of a set of related functions. For example, the name service contains function for:

- Looking up a queue name and returning the name of the queue manager where the queue is defined
- Inserting a queue name into the service's directory
- Deleting a queue name from the service's directory

It also contains initialization and termination functions.

An installable service is provided by one or more service components. Each component can perform some or all of the functions that are defined for that service. For example, in WebSphere MQ for AIX, the supplied authorization service component, the OAM, performs all the available functions. See "Authorization service interface" on page 412 for more information. The component is also responsible for managing any underlying resources or software (for example, an LDAP directory) that it needs to implement the service. Configuration files provide a standard way of loading the component and determining the addresses of the functional routines that it provides.

Figure 33 on page 403 shows how services and components are related:

- A service is defined to a queue manager by stanzas in a configuration file.
- Each service is supported by supplied code in the queue manager. Users cannot change this code and therefore cannot create their own services.
- Each service is implemented by one or more components; these can be supplied with the product or user-written. Multiple components for a service can be invoked, each supporting different facilities within the service.
- Entry points connect the service components to the supporting code in the queue manager.

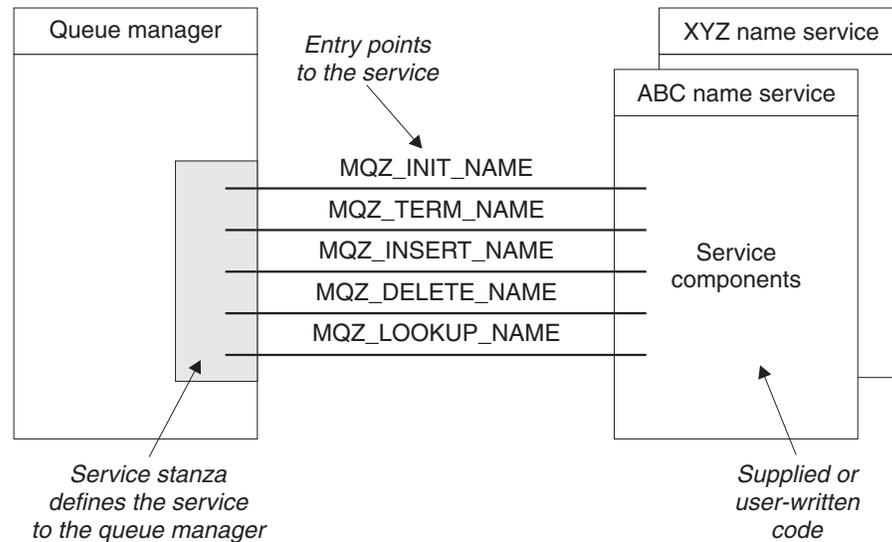


Figure 33. Understanding services, components, and entry points

Entry-points

Each service component is represented by a list of the entry-point addresses of the routines that support a particular installable service. The installable service defines the function to be performed by each routine.

The ordering of the service components when they are configured defines the order in which entry-points are called in an attempt to satisfy a request for the service.

In the supplied header file `cmqzc.h`, the supplied entry points to each service have an `MQZID_` prefix.

Return codes

Service components provide return codes to the queue manager to report on a variety of conditions. They report the success or failure of the operation, and indicate whether the queue manager is to proceed to the next service component. A separate *Continuation* parameter carries this indication.

Component data

A single service component might require data to be shared between its various functions. Installable services provide an optional data area to be passed on each invocation of a given service component. This data area is for the exclusive use of the service component. It is shared by all the invocations of a given function, even if they are made from different address spaces or processes. It is guaranteed to be addressable from the service component whenever it is called. You must declare the size of this area in the *ServiceComponent* stanza.

Initialization

When the component initialization routine is invoked, it must call the queue manager **MQZEP** function for each entry-point supported by the component. **MQZEP** defines an entry-point to the service. All the undefined exit points are assumed to be NULL.

Primary initialization

A component is always invoked with this option once, before it is invoked in any other way.

Secondary initialization

A component can be invoked with this option on certain platforms. For example, it can be invoked once for each operating system process, thread, or task by which the service is accessed.

If secondary initialization is used:

- The component can be invoked more than once for secondary initialization. For each such call, a matching call for secondary termination is issued when the service is no longer needed.

For naming services this is the `MQZ_TERM_NAME` call.

For authorization services this is the `MQZ_TERM_AUTHORITY` call.

- The entry points must be re-specified (by calling **MQZEP**) each time the component is called for primary and secondary initialization.
- Only one copy of component data is used for the component; there is not a different copy for each secondary initialization.
- The component is not invoked for any other calls to the service (from the operating system process, thread, or task, as appropriate) before secondary initialization has been carried out.
- The component must set the *Version* parameter to the same value for primary and secondary initialization.

Primary termination

The primary termination component is always invoked with this option once, when it is no longer required. No further calls are made to this component.

Secondary termination

The secondary termination component is invoked with this option, if it has been invoked for secondary initialization.

Configuring services and components

Configure service components using the queue manager configuration files, except on Windows systems, where each queue manager has its own stanza in the Registry. Each service used must have a *Service* stanza, which defines the service to the queue manager.

For each component within a service, there must be a *ServiceComponent* stanza. This identifies the name and path of the module containing the code for that component.

The authorization service component, known as the Object Authority Manager (OAM), is supplied with the product. When you create a queue manager, the queue manager configuration file (or the Registry on Windows systems) is automatically updated to include the appropriate stanzas for the authorization service and for the default component (the OAM). For the other components, you must configure the queue manager configuration file manually.

The code for each service component is loaded into the queue manager when the queue manager is started, using dynamic binding, where this is supported on the platform.

Service stanza format

The format of the *Service* stanza is:

```
Service:
  Name=<service_name>
  EntryPoints=<entries>
```

where:

<service_name>

The name of the service. This is defined by the service.

<entries>

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

Service stanza format for Windows systems

On Windows systems, the *Service* stanza has a third attribute, `SecurityPolicy`. The format of the stanza is:

```
Service:
  Name=<service_name>
  EntryPoints=<entries>
  SecurityPolicy=<policy>
```

where:

<service_name>

The name of the service. This is defined by the service.

<entries>

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

<policy>

NTSIDsRequired (the Windows Security Identifier), or Default. If you do not specify NTSIDsRequired, the Default value is used. This attribute is valid only if Name has a value of AuthorizationService.

Service component stanza format

The format of the *Service component* stanza is:

```
ServiceComponent:
  Service=<service_name>
  Name=<component_name>
  Module=<module_name>
  ComponentDataSize=<size>
```

where:

Configuring

<service_name>

The name of the service. This must match the Name specified in a service stanza.

<component_name>

A descriptive name of the service component. This must be unique, and contain only the 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. We recommend that you use a name starting with a company trademark or similar distinguishing string.

<module_name>

The name of the module to contain the code for this component. Specify a full path name.

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

These two stanzas can appear in any order and the stanza keys under them can also appear in any order. For either of these stanzas, all the stanza keys must be present. If a stanza key is duplicated, the last one is used.

At startup time, the queue manager processes each service component entry in the configuration file in turn. It then loads the specified component module, invoking the entry-point of the component (which must be the entry-point for initialization of the component), passing it a configuration handle.

Creating your own service component

To create your own service component:

- For all platforms, ensure that the header file `cmqzc.h` is included in your program.
- **For UNIX systems:** Create the shared library by compiling the program and linking it with the shared libraries `libmqm*` and `libmqmzf*`. (The threading suffixes and file extensions vary by platform.)

Note: Because the agent can run in a threaded environment, you must build the OAM and Name Service to run in a threaded environment. This includes using the threaded versions of `libmqm` and `libmqmzf`.

- **For Windows:** Create a DLL by compiling the program, and linking it with the libraries `MQM.LIB` and `MQMZF.LIB`.
See the *WebSphere MQ Application Programming Guide* for details of how to compile and link code for shared libraries.
- Add stanzas to the queue manager configuration file to define the service to the queue manager and to specify the location of the module. Refer to the individual chapters for each service, for more information.
- Stop and restart the queue manager to activate the component.

Using multiple service components

You can install more than one component for a given service. This allows components to provide only partial implementations of the service, and to rely on other components to provide the remaining functions.

Example of using multiple components

Suppose you create two a name services components called `ABC_name_serv` and `XYZ_name_serv`.

`ABC_name_serv`

This component supports inserting a name in, or deleting a name from, the service directory, but does not support looking up a queue name.

`XYZ_name_serv`

This component supports looking up a queue name, but does not support inserting a name in, or deleting a name from, the service directory.

What the components do

Component `ABC_name_serv` holds a database of queue names, and uses two simple algorithms to either insert, or delete, a name from the service directory.

Component `XYZ_name_serv` uses a simple algorithm that returns a fixed queue-manager name for any queue name with which it is invoked. It does not hold a database of queue names, and therefore does not support the insert and delete functions.

How the component is used

The components are installed on the same queue manager. The *ServiceComponent* stanzas are ordered so that component `ABC_name_serv` is invoked first. Any calls to insert or delete a queue in a component directory are handled by component `ABC_name_serv`; it is the only one that implements these functions. However, a lookup call that component `ABC_name_serv` cannot resolve is passed on to the lookup-only component, `XYZ_name_serv`. This component supplies a queue-manager name from its simple algorithm.

Omitting entry points when using multiple components

If you decide to use multiple components to provide a service, you can design a service component that does not implement certain functions. The installable services framework places no restrictions on which you can omit. However, for specific installable services, omission of one or more functions might be logically inconsistent with the purpose of the service.

Example of entry points used with multiple components

Table 25 on page 408 shows an example of the installable name service for which the two components have been installed. Each supports a different set of functions associated with this particular installable service. For insert function, the `ABC` component entry-point is invoked first. Entry points that have not been defined to the service (using `MQZEP`) are assumed to be `NULL`. An entry-point for initialization is provided in the table, but this is not required because initialization is carried out by the main entry-point of the component.

When the queue manager has to use an installable service, it uses the entry-points defined for that service (the columns in Table 25 on page 408). Taking each component in turn, the queue manager determines the address of the routine that implements the required function. It then calls the routine, if it exists. If the operation is successful, any results and status information are used by the queue manager.

Using multiple service components

Table 25. Example of entry-points for an installable service

Function number	ABC name service component	XYZ name service component
MQZID_INIT_NAME (Initialize)	ABC_initialize()	XYZ_initialize()
MQZID_TERM_NAME (Terminate)	ABC_terminate()	XYZ_terminate()
MQZID_INSERT_NAME (Insert)	ABC_Insert()	NULL
MQZID_DELETE_NAME (Delete)	ABC_Delete()	NULL
MQZID_LOOKUP_NAME (Lookup)	NULL	XYZ_Lookup()

If the routine does not exist, the queue manager repeats this process for the next component in the list. In addition, if the routine does exist but returns a code indicating that it could not perform the operation, the attempt continues with the next available component. Routines in service components might return a code that indicates that no further attempts to perform the operation should be made.

Chapter 20. Authorization service

The authorization service is an installable service that enables queue managers to invoke authorization facilities, for example, checking that a user ID has authority to open a queue.

This service is a component of the WebSphere MQ security enabling interface (SEI), which is part of the WebSphere MQ framework.

This chapter discusses:

- “Object authority manager (OAM)”
- “Authorization service on UNIX systems” on page 410
- “Authorization service on Windows systems” on page 411
- “Authorization service interface” on page 412

Object authority manager (OAM)

The authorization service component supplied with the WebSphere MQ products is called the Object Authority Manager (OAM). By default, the OAM is active and works with the control commands **dspmqa** (display authority), **dmpmqaut** (dump authority), and **setmqaut** (set or reset authority).

The syntax of these commands and how to use them are described in Chapter 17, “The control commands,” on page 281.

The OAM works with the *entity* of a principal or group. These entities vary from platform to platform.

When an MQI request is made or a command is issued, the OAM checks the authorization of the entity associated with the operation to see whether it can:

- Perform the requested operation.
- Access the specified queue manager resources.

The authorization service enables you to augment or replace the authority checking provided for queue managers by writing your own authorization service component.

Defining the service to the operating system

The authorization service stanzas in the queue manager configuration file `qm.ini` (or Registry on Windows systems), define the authorization service to the queue manager. See “Configuring services and components” on page 404 for information about the types of stanza.

Refreshing the OAM after changing a user’s authorization

In WebSphere MQ, you can update the OAM’s authorization group information immediately after changing a user’s authorization group membership, reflecting changes made at the operating system level, without needing to stop and restart the queue manager.

Note: When you change authorizations with the **setmqaut** command, the OAM implements such changes immediately.

Authorization service

Queue managers store authorization data on a local queue called `SYSTEM.AUTH.DATA.QUEUE`. This data is managed by `amqzfuma.exe`.

Migrating from MQSeries Version 5.1

All authorization data is migrated from the authorization files to the authorization queue the first time that you restart the queue manager after migrating from MQSeries Version 5.1. If the OAM detects a missing file:

1. If the authorization applies to a single object, the OAM gives the mqm group (or Administrator on Windows systems) access to the object and continues with the migration. Message AMQ5528 is written to the queue manager's error log. Refer to *WebSphere MQ Messages* for more information about message AMQ5528.
2. If the authorization applies to a class of objects, the OAM stops the migration. The queue manager does not start until the file has been replaced.

Authorization service on UNIX systems

On these platforms:

Principal

Is a UNIX system user ID, or an ID associated with an application program running on behalf of a user.

Group Is a UNIX system-defined collection of principals.

Authorizations can be granted or revoked at the group level only. A request to grant or revoke a user's authority updates the primary group for that user.

Configuring authorization service stanzas: UNIX systems

On UNIX systems, each queue manager has its own queue manager configuration file.

For example, on AIX, the default path and file name of the queue manager configuration file for queue manager QMNAME is `/var/mqm/qmgrs/QMNAME/qm.ini`.

The *Service* stanza and the *ServiceComponent* stanza for the default authorization component are added to `qm.ini` automatically, but can be overridden by `mqsnout`. Any other *ServiceComponent* stanzas must be added manually.

For example, the following stanzas in the queue manager configuration file define two authorization service components on WebSphere MQ for AIX:

```

Service:
  Name=AuthorizationService
  EntryPoints=13

ServiceComponent:
  Service=AuthorizationService
  Name=MQSeries.UNIX.auth.service
  Module=/usr/mqm/lib/amqzfu
  ComponentDataSize=0

ServiceComponent:
  Service=AuthorizationService
  Name=user.defined.authorization.service
  Module=/usr/bin/udas01
  ComponentDataSize=96

```

Figure 34. UNIX authorization service stanzas in *qm.ini*

The service component stanza (`MQSeries.UNIX.auth.service`) defines the default authorization service component, the OAM. If you remove this stanza and restart the queue manager, the OAM is disabled and no authorization checks are made.

Authorization service on Windows systems

On Windows systems:

Principal

Is a Windows user ID, or an ID associated with an application program running on behalf of a user.

Group Is a Windows group.

Authorizations can be granted or revoked at the principal or group level.

Configuring authorization service stanzas: Windows systems

On WebSphere MQ for Windows each queue manager has its own stanza in the registry.

The *Service* stanza and the *ServiceComponent* stanza for the default authorization component are added to the Registry automatically, but can be overridden using *mqsnout*. Any other *ServiceComponent* stanzas must be added manually.

You can also add the *SecurityPolicy* attribute using the WebSphere MQ services. The *SsecurityPolicy* attribute applies only if the service specified on the *Service* stanza is the authorization service, that is, the default OAM. The *SecurityPolicy* attribute allows you to specify the security policy for each queue manager. The possible values are:

Default

Specify `Default` if you want the default security policy to take effect. If a Windows security identifier (NT SID) is not passed to the OAM for a particular user ID, an attempt is made to obtain the appropriate SID by searching the relevant security databases.

NTSIDsRequired

Requires that an NT SID is passed to the OAM when performing security checks.

For more general information about security, see Chapter 10, “WebSphere MQ security,” on page 129.

The service component stanza, `MQSeries.WindowsNT.auth.service` defines the default authorization service component, the OAM. If you remove this stanza and restart the queue manager, the OAM is disabled and no authorization checks are made.

Authorization service interface

The authorization service provides the following entry points for use by the queue manager:

MQZ_AUTHENTICATE_USER

Authenticates a user ID and password, and can set identity context fields.

MQZ_CHECK_AUTHORITY

Checks whether an entity has authority to perform one or more operations on a specified object.

MQZ_COPY_ALL_AUTHORITY

Copies all the current authorizations that exist for a referenced object to another object.

MQZ_DELETE_AUTHORITY

Deletes all authorizations associated with a specified object.

MQZ_ENUMERATE_AUTHORITY_DATA

Retrieves all the authority data that matches the selection criteria specified.

MQZ_FREE_USER

Frees associated allocated resources.

MQZ_GET_AUTHORITY

Gets the authority that an entity has to access a specified object.

MQZ_GET_EXPLICIT_AUTHORITY

Gets either the authority that a named group has to access a specified object (but without the additional authority of the **nobody** group) or the authority that the primary group of the named principal has to access a specified object.

MQZ_INIT_AUTHORITY

Initializes authorization service component.

MQZ_INQUIRE

Queries the supported functionality of the authorization service.

MQZ_REFRESH_CACHE

Refresh all authorizations.

MQZ_SET_AUTHORITY

Sets the authority that an entity has to a specified object.

MQZ_TERM_AUTHORITY

Terminates authorization service component.

In addition, on WebSphere MQ for Windows, the authorization service provides the following entry points for use by the queue manager:

- **MQZ_CHECK_AUTHORITY_2**
- **MQZ_GET_AUTHORITY_2**
- **MQZ_GET_EXPLICIT_AUTHORITY_2**

- **MQZ_SET_AUTHORITY_2**

These entry points support the use of the Windows Security Identifier (NT SID).

These names are defined as **typedefs**, in the header file `cmqzc.h`, which can be used to prototype the component functions.

The initialization function (**MQZ_INIT_AUTHORITY**) must be the main entry point for the component. The other functions are invoked through the entry point address that the initialization function has added into the component entry point vector.

See “Creating your own service component” on page 406 for more information.

Chapter 21. Name service

The name service is an installable service that provides support to the queue manager for looking up the name of the queue manager that owns a specified queue. No other queue attributes can be retrieved from a name service.

The name service enables an application to open remote queues for output as if they were local queues. A name service is not invoked for objects other than queues.

Note: The remote queues *must* have their *Scope* attribute set to CELL.

When an application opens a queue, it looks for the name of the queue first in the queue manager's directory. If it does not find it there, it looks in as many name services as have been configured, until it finds one that recognizes the queue name. If none recognizes the name, the open fails.

The name service returns the owning queue manager for that queue. The queue manager then continues with the MQOPEN request as if the command had specified the queue and queue manager name in the original request.

The name service interface (NSI) is part of the WebSphere MQ framework.

This chapter discusses:

- "How the name service works"

How the name service works

If a queue definition specifies the *Scope* attribute as queue manager, that is, SCOPE(QMGR) in MQSC, the queue definition (along with all the queue attributes) is stored in the queue manager's directory only. This cannot be replaced by an installable service.

If a queue definition specifies the *Scope* attribute as cell, that is, SCOPE(CELL) in MQSC, the queue definition is again stored in the queue manager's directory, along with all the queue attributes. However, the queue and queue-manager name are also stored in a name service. If no service is available that can store this information, a queue with the *Scope* cell cannot be defined.

The directory in which the information is stored can be managed by the service, or the service can use an underlying service, for example, an LDAP directory, for this purpose. In either case, definitions stored in the directory must persist, even after the component and queue manager have terminated, until they are explicitly deleted.

Notes:

1. To send a message to a remote host's local queue definition (with a scope of CELL) on a different queue manager within a naming directory cell, you need to define a channel.
2. You cannot get messages directly from the remote queue, even when it has a scope of CELL.

Name service

3. No remote queue definition is required when sending to a queue with a scope of CELL.
4. The naming service centrally defines the destination queue, although you still need a transmission queue to the destination queue manager and a pair of channel definitions. In addition, the transmission queue on the local system must have the same name as the queue manager owning the target queue, with the scope of cell, on the remote system.

For example, if the remote queue manager has the name QM01, the transmission queue on the local system must also have the name QM01. See *WebSphere MQ Intercommunication* for further information.

Name service interface

A name service provides the following entry points for use by the queue manager:

MQZ_INIT_NAME

Initialize the name service component.

MQZ_TERM_NAME

Terminate the name service component.

MQZ_LOOKUP_NAME

Look up the queue-manager name for the specified queue.

MQZ_INSERT_NAME

Insert an entry containing the owning queue-manager name for the specified queue into the directory used by the service.

MQZ_DELETE_NAME

Delete the entry for the specified queue from the directory used by the service.

If there is more than one name service configured:

- For lookup, the MQZ_LOOKUP_NAME function is invoked for each service in the list until the queue name is resolved (unless any component indicates that the search should stop).
- For insert, the MQZ_INSERT_NAME function is invoked for the first service in the list that supports this function.
- For delete, the MQZ_DELETE_NAME function is invoked for the first service in the list that supports this function.

Do not have more than one component that supports the insert and delete functions. However, a component that only supports lookup is feasible, and could be used, for example, as the last component in the list to resolve any name that is not known by any other name service component to a queue manager at which the name can be defined.

In the C programming language the names are defined as function datatypes using the typedef statement. These can be used to prototype the service functions, to ensure that the parameters are correct.

The header file that contains all the material specific to installable services is `cmqzc.h` for the C language.

Apart from the initialization function (MQZ_INIT_NAME), which must be the component's main entry point, functions are invoked by the entry point address that the initialization function has added, using the MQZEP call.

The following examples of UNIX configuration file stanzas for the name service specify a name service component provided by the (fictitious) ABC company.

```
# Stanza for name service
Service:
  Name=NameService
  EntryPoints=5

# Stanza for name service component, provided by ABC
ServiceComponent:
  Service=NameService
  Name=ABC.Name.Service
  Module=/usr/lib/abcname
  ComponentDataSize=1024
```

Figure 35. Name service stanzas in `qm.ini` (for UNIX systems)

Note: On Windows systems, name service stanza information is stored in the Registry.

Name service

Chapter 22. Installable services interface reference information

This chapter provides reference information for the installable services. It includes:

- “How the functions are shown” on page 420
- “MQZEP – Add component entry point” on page 421
- “MQZ_AUTHENTICATE_USER – Authenticate user” on page 423
- “MQZ_CHECK_AUTHORITY – Check authority” on page 426
- “MQZ_CHECK_AUTHORITY_2 – Check authority (extended)” on page 431
- “MQZ_COPY_ALL_AUTHORITY – Copy all authority” on page 436
- “MQZ_DELETE_AUTHORITY – Delete authority” on page 439
- “MQZ_ENUMERATE_AUTHORITY_DATA – Enumerate authority data” on page 442
- “MQZ_FREE_USER – Free user” on page 445
- “MQZ_GET_AUTHORITY – Get authority” on page 447
- “MQZ_GET_AUTHORITY_2 – Get authority (extended)” on page 450
- “MQZ_GET_EXPLICIT_AUTHORITY – Get explicit authority” on page 453
- “MQZ_GET_EXPLICIT_AUTHORITY_2 – Get explicit authority (extended)” on page 456
- “MQZ_INIT_AUTHORITY – Initialize authorization service” on page 459
- “MQZ_INQUIRE – Inquire authorization service” on page 462
- “MQZ_REFRESH_CACHE – Refresh all authorizations” on page 466
- “MQZ_SET_AUTHORITY – Set authority” on page 468
- “MQZ_SET_AUTHORITY_2 – Set authority (extended)” on page 471
- “MQZ_TERM_AUTHORITY – Terminate authorization service” on page 474
- “MQZAC – Application context” on page 476
- “MQZAD – Authority data” on page 478
- “MQZED – Entity descriptor” on page 483
- “MQZIC – Identity context” on page 485
- “MQZFP – Free parameters” on page 487
- “MQZ_DELETE_NAME – Delete name” on page 489
- “MQZ_INIT_NAME – Initialize name service” on page 491
- “MQZ_INSERT_NAME – Insert name” on page 494
- “MQZ_LOOKUP_NAME – Lookup name” on page 496
- “MQZ_TERM_NAME – Terminate name service” on page 499

The functions and data types are in alphabetic order within the group for each service type.

Table 26. Installable services functions

<i>Service type</i>	<i>Functions</i>	<i>page</i>
All	MQZEP – Add component entry point	421
	MQHCONFIG – Configuration handle	422
	PMQFUNC – Pointer to function	422

Installable services

Table 26. Installable services functions (continued)

<i>Service type</i>	<i>Functions</i>	<i>page</i>
Authorization	MQZ_AUTHENTICATE_USER	423
	MQZ_CHECK_AUTHORITY	426
	MQZ_COPY_ALL_AUTHORITY	436
	MQZ_DELETE_AUTHORITY	439
	MQZ_ENUMERATE_AUTHORITY_DATA	442
	MQZ_FREE_USER	445
	MQZ_GET_AUTHORITY	447
	MQZ_GET_EXPLICIT_AUTHORITY	453
	MQZ_INIT_AUTHORITY	459
	MQZ_INQUIRE	462466
	MQZ_REFRESH_CACHE	468
	MQZ_SET_AUTHORITY	474
	MQZ_TERM_AUTHORITY	
Extended authorization	MQZ_CHECK_AUTHORITY_2	431
	MQZ_GET_AUTHORITY_2	450
	MQZ_GET_EXPLICIT_AUTHORITY_2	456
	MQZ_SET_AUTHORITY_2	471
Name	MQZ_DELETE_NAME	489
	MQZ_INIT_NAME	491
	MQZ_INSERT_NAME	494
	MQZ_LOOKUP_NAME	496
	MQZ_TERM_NAME	499

How the functions are shown

For each function there is a description, including the function identifier (for MQZEP).

The *parameters* are shown listed in the order they must occur. They must all be present.

Parameters and data types

Each parameter name is followed by its data type in parentheses. These are the elementary data types described in the *WebSphere MQ Application Programming Reference* manual.

The C language invocation is also given, after the description of the parameters.

MQZEP – Add component entry point

This function is invoked by a service component, during initialization, to add an entry point to the entry point vector for that service component.

Syntax

MQZEP (Hconfig, Function, EntryPoint, CompCode, Reason)

Parameters

The MQZEP call has the following parameters.

Hconfig (MQHCONFIG) – input

Configuration handle.

This handle represents the component which is being configured for this particular installable service. It must be the same as the one passed to the component configuration function by the queue manager on the component initialization call.

Function (MQLONG) – input

Function identifier.

Valid values for this are defined for each installable service.

If MQZEP is called more than once for the same function, the last call made provides the entry point which is used.

EntryPoint (PMQFUNC) – input

Function entry point.

This is the address of the entry point provided by the component to perform the function.

The value NULL is valid, and indicates that the function is not provided by this component. NULL is assumed for entry points which are not defined using MQZEP.

CompCode (MQLONG) – output

Completion code.

It is one of the following:

MQCC_OK

Successful completion.

MQCC_FAILED

Call failed.

Reason (MQLONG) – output

Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE

(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQZEP call

MQRC_FUNCTION_ERROR

(2281, X'8E9') Function identifier not valid.

MQRC_HCONFIG_ERROR

(2280, X'8E8') Configuration handle not valid.

For more information on these reason codes, see the *WebSphere MQ Application Programming Reference*.

C invocation

```
MQZEP (Hconfig, Function, EntryPoint, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHCONFIG Hconfig;    /* Configuration handle */
MQLONG    Function;   /* Function identifier */
PMQFUNC   EntryPoint; /* Function entry point */
MQLONG    CompCode;   /* Completion code */
MQLONG    Reason;    /* Reason code qualifying CompCode */
```

MQHCONFIG – Configuration handle

The MQHCONFIG data type represents a configuration handle, that is, the component that is being configured for a particular installable service. A configuration handle must be aligned on its natural boundary.

Note: Applications must test variables of this type for equality only.

C declaration

```
typedef void MQPOINTER MQHCONFIG;
```

PMQFUNC – Pointer to function

Pointer to a function.

C declaration

```
typedef void MQPOINTER PMQFUNC;
```

MQZ_AUTHENTICATE_USER – Authenticate user

This function is provided by a MQZAS_VERSION_5 authorization service component, and is invoked by the queue manager to authenticate a user, or to set identity context fields. It is invoked when WebSphere MQ's user application context is established. This happens during connect calls at the point where the application's user context is initialized, and at each point where the application's user context is changed. Each time a connect call is made, the application's user context information is reacquired in the *IdentityContext* field.

The function identifier for this function (for MQZEP) is MQZID_AUTHENTICATE_USER.

Syntax

```
MQZ_AUTHENTICATE_USER (QMgrName, SecurityParms, ApplicationContext,
                      IdentityContext, CorrelationPtr, ComponentData, Continuation, CompCode,
                      Reason)
```

Parameters

The MQZ_AUTHENTICATE_USER call has the following parameters.

QMgrName (MQCHAR48) – input

Queue manager name.

The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue-manager name is passed to the component for information; the authorization service interface does not require the component to make use of it in any defined manner.

SecurityParms (MQCSP) – input

Security parameters.

Data relating to the user ID, password, and authentication type. If the *AuthenticationType* attribute of the MQCSP structure is specified as MQCSP_AUTH_USER_ID_AND_PWD, both the user ID and password are compared against the equivalent fields in the *IdentityContext* (MQZIC) parameter to determine whether they match. For more information, see the *WebSphere MQ Application Programming Reference*.

During an MQCONN MQI call this parameter contains null, or default values.

ApplicationContext (MQZAC) – input

Application context.

Data relating to the calling application. See "MQZAC – Application context" on page 476 for details.

During every MQCONN or MQCONNX MQI call, the user context information in the MQZAC structure is reacquired.

IdentityContext (MQZIC) – input/output

Identity context.

MQZ_AUTHENTICATE_USER

On input to the authenticate user function, this identifies the current identity context. The authenticate user function can change this, at which point the queue manager adopts the new identity context. See “MQZIC – Identity context” on page 485 for more details on the MQZIC structure.

CorrelationPtr (MQPTR) – output

Correlation pointer.

Specifies the address of any correlation data. This pointer is subsequently passed on to other OAM calls.

ComponentData (MQBYTE×ComponentDataLength) – input/output

Component data.

This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions provided by this component are preserved, and presented the next time one of this component’s functions is called.

The length of this data area is passed by the queue manager in the *ComponentDataLength* parameter of the MQZ_INIT_AUTHORITY call.

Continuation (MQLONG) – output

Continuation flag.

The following values can be specified:

MQZCI_DEFAULT

Continuation dependent on other components.

MQZCI_STOP

Do not continue with next component.

CompCode (MQLONG) – output

Completion code.

It is one of the following:

MQCC_OK

Successful completion.

MQCC_FAILED

Call failed.

Reason (MQLONG) – output

Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE

(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQRC_SERVICE_ERROR

(2289, X'8F1') Unexpected error occurred accessing service.

C invocation

```
MQZ_AUTHENTICATE_USER (QMgrName, SecurityParms, ApplicationContext,  
IdentityContext, &CorrelationPtr, ComponentData,  
&Continuation, &CompCode, &Reason);
```

The parameters passed to the service are declared as follows:

```
MQCHAR48  QMgrName;          /* Queue manager name */
MQCSP     SecurityParms;     /* Security parameters */
MQZAC     ApplicationContext; /* Application context */
MQZIC     IdentityContext;   /* Identity context */
MQPTR     CorrelationPtr;    /* Correlation pointer */
MQBYTE    ComponentData[n]; /* Component data */
MQLONG    Continuation;     /* Continuation indicator set by
                             component */
MQLONG    CompCode;         /* Completion code */
MQLONG    Reason;          /* Reason code qualifying CompCode */
```

MQZ_CHECK_AUTHORITY – Check authority

This function is provided by a MQZAS_VERSION_1 authorization service component, and is invoked by the queue manager to check whether an entity has authority to perform a particular action, or actions, on a specified object.

The function identifier for this function (for MQZEP) is MQZID_CHECK_AUTHORITY.

Syntax

```
MQZ_CHECK_AUTHORITY (QMgrName, EntityName, EntityType, ObjectName,
                    ObjectType, Authority, ComponentData, Continuation, CompCode, Reason)
```

Parameters

The MQZ_CHECK_AUTHORITY call has the following parameters.

QMgrName (MQCHAR48) – input

Queue manager name.

The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue-manager name is passed to the component for information; the authorization service interface does not require the component to make use of it in any defined manner.

EntityName (MQCHAR12) – input

Entity name.

The name of the entity whose authorization to the object is to be checked. The maximum length of the string is 12 characters; if it is shorter than that it is padded to the right with blanks. The name is not terminated by a null character.

It is not essential for this entity to be known to the underlying security service. If it is not known, the authorizations of the special **nobody** group (to which all entities are assumed to belong) are used for the check. An all-blank name is valid and can be used in this way.

EntityType (MQLONG) – input

Entity type.

The type of entity specified by *EntityName*. It is one of the following:

MQZAET_PRINCIPAL

Principal.

MQZAET_GROUP

Group.

ObjectName (MQCHAR48) – input

Object name.

The name of the object to which access is required. The maximum length of the string is 48 characters; if it is shorter than that it is padded to the right with blanks. The name is not terminated by a null character.

If *ObjectType* is MQOT_Q_MGR, this name is the same as *QMgrName*.

ObjectType (MQLONG) – input

Object type.

The type of entity specified by *ObjectName*. It is one of the following:

MQOT_AUTH_INFO

Authentication information.

MQOT_CHANNEL

Channel.

MQOT_CLNTCONN_CHANNEL

Client connection channel.

MQOT_LISTENER

Listener.

MQOT_NAMELIST

Namelist.

MQOT_PROCESS

Process definition.

MQOT_Q

Queue.

MQOT_Q_MGR

Queue manager.

MQOT_SERVICE

Service.

Authority (MQLONG) – input

Authority to be checked.

If one authorization is being checked, this field is equal to the appropriate authorization operation (MQZAO_* constant). If more than one authorization is being checked, it is the bitwise OR of the corresponding MQZAO_* constants.

The following authorizations apply to use of the MQI calls:

MQZAO_CONNECT

Ability to use the MQCONN call.

MQZAO_BROWSE

Ability to use the MQGET call with a browse option.

This allows the MQGMO_BROWSE_FIRST, MQGMO_BROWSE_MSG_UNDER_CURSOR, or MQGMO_BROWSE_NEXT option to be specified on the MQGET call.

MQZAO_INPUT

Ability to use the MQGET call with an input option.

This allows the MQOO_INPUT_SHARED, MQOO_INPUT_EXCLUSIVE, or MQOO_INPUT_AS_Q_DEF option to be specified on the MQOPEN call.

MQZAO_OUTPUT

Ability to use the MQPUT call.

This allows the MQOO_OUTPUT option to be specified on the MQOPEN call.

MQZAO_INQUIRE

Ability to use the MQINQ call.

This allows the MQOO_INQUIRE option to be specified on the MQOPEN call.

MQZ_CHECK_AUTHORITY

MQZAO_SET

Ability to use the MQSET call.

This allows the MQOO_SET option to be specified on the MQOPEN call.

MQZAO_PASS_IDENTITY_CONTEXT

Ability to pass identity context.

This allows the MQOO_PASS_IDENTITY_CONTEXT option to be specified on the MQOPEN call, and the MQPMO_PASS_IDENTITY_CONTEXT option to be specified on the MQPUT and MQPUT1 calls.

MQZAO_PASS_ALL_CONTEXT

Ability to pass all context.

This allows the MQOO_PASS_ALL_CONTEXT option to be specified on the MQOPEN call, and the MQPMO_PASS_ALL_CONTEXT option to be specified on the MQPUT and MQPUT1 calls.

MQZAO_SET_IDENTITY_CONTEXT

Ability to set identity context.

This allows the MQOO_SET_IDENTITY_CONTEXT option to be specified on the MQOPEN call, and the MQPMO_SET_IDENTITY_CONTEXT option to be specified on the MQPUT and MQPUT1 calls.

MQZAO_SET_ALL_CONTEXT

Ability to set all context.

This allows the MQOO_SET_ALL_CONTEXT option to be specified on the MQOPEN call, and the MQPMO_SET_ALL_CONTEXT option to be specified on the MQPUT and MQPUT1 calls.

MQZAO_ALTERNATE_USER_AUTHORITY

Ability to use alternate user authority.

This allows the MQOO_ALTERNATE_USER_AUTHORITY option to be specified on the MQOPEN call, and the MQPMO_ALTERNATE_USER_AUTHORITY option to be specified on the MQPUT1 call.

MQZAO_ALL_MQI

All of the MQI authorizations.

This enables all of the authorizations described above.

The following authorizations apply to administration of a queue manager:

MQZAO_CREATE

Ability to create objects of a specified type.

MQZAO_DELETE

Ability to delete a specified object.

MQZAO_DISPLAY

Ability to display the attributes of a specified object.

MQZAO_CHANGE

Ability to change the attributes of a specified object.

MQZAO_CLEAR

Ability to delete all messages from a specified queue.

MQZAO_AUTHORIZE

Ability to authorize other users for a specified object.

MQZAO_CONTROL

Ability to start or stop a listener, service, or non-client channel object, and the ability to ping a non-client channel object.

MQZAO_CONTROL_EXTENDED

Ability to reset a sequence number, or resolve an indoubt message on a non-client channel object.

MQZAO_ALL_ADMIN

All of the administration authorizations, other than MQZAO_CREATE.

The following authorizations apply to both use of the MQI and to administration of a queue manager:

MQZAO_ALL

All authorizations, other than MQZAO_CREATE.

MQZAO_NONE

No authorizations.

ComponentData (MQBYTE×ComponentDataLength) – input/output

Component data.

This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions provided by this component are preserved, and presented the next time one of this component's functions is called.

The length of this data area is passed by the queue manager in the *ComponentDataLength* parameter of the MQZ_INIT_AUTHORITY call.

Continuation (MQLONG) – output

Continuation indicator set by component.

The following values can be specified:

MQZCI_DEFAULT

Continuation dependent on queue manager.

For MQZ_CHECK_AUTHORITY this has the same effect as MQZCI_STOP.

MQZCI_CONTINUE

Continue with next component.

MQZCI_STOP

Do not continue with next component.

CompCode (MQLONG) – output

Completion code.

It is one of the following:

MQCC_OK

Successful completion.

MQCC_FAILED

Call failed.

Reason (MQLONG) – output

Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQZ_CHECK_AUTHORITY

MQRC_NONE

(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQRC_NOT_AUTHORIZED

(2035, X'7F3') Not authorized for access.

MQRC_SERVICE_ERROR

(2289, X'8F1') Unexpected error occurred accessing service.

MQRC_SERVICE_NOT_AVAILABLE

(2285, X'8ED') Underlying service not available.

For more information on these reason codes, see the *WebSphere MQ Application Programming Reference*.

C invocation

```
MQZ_CHECK_AUTHORITY (QMgrName, EntityName, EntityType, ObjectName,  
                    ObjectType, Authority, ComponentData,  
                    &Continuation, &CompCode, &Reason);
```

The parameters passed to the service are declared as follows:

```
MQCHAR48 QMgrName;          /* Queue manager name */  
MQCHAR12 EntityName;       /* Entity name */  
MQLONG   EntityType;       /* Entity type */  
MQCHAR48 ObjectName;      /* Object name */  
MQLONG   ObjectType;      /* Object type */  
MQLONG   Authority;       /* Authority to be checked */  
MQBYTE   ComponentData[n]; /* Component data */  
MQLONG   Continuation;    /* Continuation indicator set by  
                           component */  
MQLONG   CompCode;        /* Completion code */  
MQLONG   Reason;          /* Reason code qualifying CompCode */
```

MQZ_CHECK_AUTHORITY_2 – Check authority (extended)

This function is provided by a MQZAS_VERSION_2 authorization service component, and is invoked by the queue manager to check whether an entity has authority to perform a particular action, or actions, on a specified object.

The function identifier for this function (for MQZEP) is MQZID_CHECK_AUTHORITY.

MQZ_CHECK_AUTHORITY_2 is similar to MQZ_CHECK_AUTHORITY, but with the *EntityName* parameter replaced by the *EntityData* parameter.

Syntax

MQZ_CHECK_AUTHORITY_2 (*QMgrName*, *EntityData*, *EntityType*, *ObjectName*, *ObjectType*, *Authority*, *ComponentData*, *Continuation*, *CompCode*, *Reason*)

Parameters

The MQZ_CHECK_AUTHORITY_2 call has the following parameters.

QMgrName (MQCHAR48) – input

Queue manager name.

The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue-manager name is passed to the component for information; the authorization service interface does not require the component to make use of it in any defined manner.

EntityData (MQZED) – input

Entity data.

Data relating to the entity whose authorization to the object is to be checked. See “MQZED – Entity descriptor” on page 483 for details.

It is not essential for this entity to be known to the underlying security service. If it is not known, the authorizations of the special **nobody** group (to which all entities are assumed to belong) are used for the check. An all-blank name is valid and can be used in this way.

EntityType (MQLONG) – input

Entity type.

The type of entity specified by *EntityData*. It is one of the following:

MQZAET_PRINCIPAL

Principal.

MQZAET_GROUP

Group.

ObjectName (MQCHAR48) – input

Object name.

MQZ_CHECK_AUTHORITY_2

The name of the object to which access is required. The maximum length of the string is 48 characters; if it is shorter than that it is padded to the right with blanks. The name is not terminated by a null character.

If *ObjectType* is MQOT_Q_MGR, this name is the same as *QMgrName*.

ObjectType (MQLONG) – input

Object type.

The type of entity specified by *ObjectName*. It is one of the following:

MQOT_AUTH_INFO

Authentication information.

MQOT_CHANNEL

Channel.

MQOT_CLNTCONN_CHANNEL

Client connection channel.

MQOT_LISTENER

Listener.

MQOT_NAMELIST

Namelist.

MQOT_PROCESS

Process definition.

MQOT_Q

Queue.

MQOT_Q_MGR

Queue manager.

MQOT_SERVICE

Service.

Authority (MQLONG) – input

Authority to be checked.

If one authorization is being checked, this field is equal to the appropriate authorization operation (MQZAO_* constant). If more than one authorization is being checked, it is the bitwise OR of the corresponding MQZAO_* constants.

The following authorizations apply to use of the MQI calls:

MQZAO_CONNECT

Ability to use the MQCONN call.

MQZAO_BROWSE

Ability to use the MQGET call with a browse option.

This allows the MQGMO_BROWSE_FIRST, MQGMO_BROWSE_MSG_UNDER_CURSOR, or MQGMO_BROWSE_NEXT option to be specified on the MQGET call.

MQZAO_INPUT

Ability to use the MQGET call with an input option.

This allows the MQOO_INPUT_SHARED, MQOO_INPUT_EXCLUSIVE, or MQOO_INPUT_AS_Q_DEF option to be specified on the MQOPEN call.

MQZAO_OUTPUT

Ability to use the MQPUT call.

This allows the MQOO_OUTPUT option to be specified on the MQOPEN call.

MQZAO_INQUIRE

Ability to use the MQINQ call.

This allows the MQOO_INQUIRE option to be specified on the MQOPEN call.

MQZAO_SET

Ability to use the MQSET call.

This allows the MQOO_SET option to be specified on the MQOPEN call.

MQZAO_PASS_IDENTITY_CONTEXT

Ability to pass identity context.

This allows the MQOO_PASS_IDENTITY_CONTEXT option to be specified on the MQOPEN call, and the MQPMO_PASS_IDENTITY_CONTEXT option to be specified on the MQPUT and MQPUT1 calls.

MQZAO_PASS_ALL_CONTEXT

Ability to pass all context.

This allows the MQOO_PASS_ALL_CONTEXT option to be specified on the MQOPEN call, and the MQPMO_PASS_ALL_CONTEXT option to be specified on the MQPUT and MQPUT1 calls.

MQZAO_SET_IDENTITY_CONTEXT

Ability to set identity context.

This allows the MQOO_SET_IDENTITY_CONTEXT option to be specified on the MQOPEN call, and the MQPMO_SET_IDENTITY_CONTEXT option to be specified on the MQPUT and MQPUT1 calls.

MQZAO_SET_ALL_CONTEXT

Ability to set all context.

This allows the MQOO_SET_ALL_CONTEXT option to be specified on the MQOPEN call, and the MQPMO_SET_ALL_CONTEXT option to be specified on the MQPUT and MQPUT1 calls.

MQZAO_ALTERNATE_USER_AUTHORITY

Ability to use alternate user authority.

This allows the MQOO_ALTERNATE_USER_AUTHORITY option to be specified on the MQOPEN call, and the MQPMO_ALTERNATE_USER_AUTHORITY option to be specified on the MQPUT1 call.

MQZAO_ALL_MQI

All of the MQI authorizations.

This enables all of the authorizations described above.

The following authorizations apply to administration of a queue manager:

MQZAO_CREATE

Ability to create objects of a specified type.

MQZAO_DELETE

Ability to delete a specified object.

MQZAO_DISPLAY

Ability to display the attributes of a specified object.

MQZAO_CHANGE

Ability to change the attributes of a specified object.

MQZ_CHECK_AUTHORITY_2

MQZAO_CLEAR

Ability to delete all messages from a specified queue.

MQZAO_AUTHORIZE

Ability to authorize other users for a specified object.

MQZAO_CONTROL

Ability to start or stop a listener, service, or non-client channel object, and the ability to ping a non-client channel object.

MQZAO_CONTROL_EXTENDED

Ability to reset a sequence number, or resolve an indoubt message on a non-client channel object.

MQZAO_ALL_ADMIN

All of the administration authorizations, other than MQZAO_CREATE.

The following authorizations apply to both use of the MQI and to administration of a queue manager:

MQZAO_ALL

All authorizations, other than MQZAO_CREATE.

MQZAO_NONE

No authorizations.

ComponentData (MQBYTE×ComponentDataLength) – input/output

Component data.

This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions provided by this component are preserved, and presented the next time one of this component's functions is called.

The length of this data area is passed by the queue manager in the *ComponentDataLength* parameter of the MQZ_INIT_AUTHORITY call.

Continuation (MQLONG) – output

Continuation indicator set by component.

The following values can be specified:

MQZCI_DEFAULT

Continuation dependent on queue manager.

For MQZ_CHECK_AUTHORITY_2 this has the same effect as MQZCI_STOP.

MQZCI_CONTINUE

Continue with next component.

MQZCI_STOP

Do not continue with next component.

CompCode (MQLONG) – output

Completion code.

It is one of the following:

MQCC_OK

Successful completion.

MQCC_FAILED

Call failed.

Reason (MQLONG) – outputReason code qualifying *CompCode*.If *CompCode* is MQCC_OK:**MQRC_NONE**

(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:**MQRC_NOT_AUTHORIZED**

(2035, X'7F3') Not authorized for access.

MQRC_SERVICE_ERROR

(2289, X'8F1') Unexpected error occurred accessing service.

MQRC_SERVICE_NOT_AVAILABLE

(2285, X'8ED') Underlying service not available.

For more information on these reason codes, see the *WebSphere MQ Application Programming Reference*.

C invocation

```
MQZ_CHECK_AUTHORITY_2 (QMgrName, &EntityData, EntityType,
                      ObjectName, ObjectType, Authority, ComponentData,
                      &Continuation, &CompCode, &Reason);
```

The parameters passed to the service are declared as follows:

```
MQCHAR48 QMgrName;          /* Queue manager name */
MQZED    EntityData;        /* Entity data */
MQLONG   EntityType;        /* Entity type */
MQCHAR48 ObjectName;        /* Object name */
MQLONG   ObjectType;        /* Object type */
MQLONG   Authority;         /* Authority to be checked */
MQBYTE   ComponentData[n]; /* Component data */
MQLONG   Continuation;      /* Continuation indicator set by
                             component */
MQLONG   CompCode;          /* Completion code */
MQLONG   Reason;           /* Reason code qualifying CompCode */
```

MQZ_COPY_ALL_AUTHORITY – Copy all authority

This function is provided by an authorization service component. It is invoked by the queue manager to copy all of the authorizations that are currently in force for a reference object to another object.

The function identifier for this function (for MQZEP) is MQZID_COPY_ALL_AUTHORITY.

Syntax

MQZ_COPY_ALL_AUTHORITY (*QMgrName*, *RefObjectName*, *ObjectName*,
ObjectType, *ComponentData*, *Continuation*, *CompCode*, *Reason*)

Parameters

The MQZ_COPY_ALL_AUTHORITY call has the following parameters.

QMgrName (MQCHAR48) – input

Queue manager name.

The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue-manager name is passed to the component for information; the authorization service interface does not require the component to make use of it in any defined manner.

RefObjectName (MQCHAR48) – input

Reference object name.

The name of the reference object, the authorizations for which are to be copied. The maximum length of the string is 48 characters; if it is shorter than that it is padded to the right with blanks. The name is not terminated by a null character.

ObjectName (MQCHAR48) – input

Object name.

The name of the object for which accesses are to be set. The maximum length of the string is 48 characters; if it is shorter than that it is padded to the right with blanks. The name is not terminated by a null character.

ObjectType (MQLONG) – input

Object type.

The type of object specified by *RefObjectName* and *ObjectName*. It is one of the following:

MQOT_AUTH_INFO
Authentication information.

MQOT_CHANNEL
Channel.

MQOT_CLNTCONN_CHANNEL
Client connection channel.

MQOT_LISTENER
Listener.

MQOT_NAMELIST
Namelist.
MQOT_PROCESS
Process definition.
MQOT_Q
Queue.
MQOT_Q_MGR
Queue manager.
MQOT_SERVICE
Service.

ComponentData (MQBYTE×ComponentDataLength) – input/output

Component data.

This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions provided by this component are preserved, and presented the next time one of this component's functions is called.

The length of this data area is passed by the queue manager in the *ComponentDataLength* parameter of the MQZ_INIT_AUTHORITY call.

Continuation (MQLONG) – output

Continuation indicator set by component.

The following values can be specified:

MQZCI_DEFAULT
Continuation dependent on queue manager.
For MQZ_COPY_ALL_AUTHORITY this has the same effect as MQZCI_STOP.

MQZCI_CONTINUE
Continue with next component.

MQZCI_STOP
Do not continue with next component.

CompCode (MQLONG) – output

Completion code.

It is one of the following:

MQCC_OK
Successful completion.
MQCC_FAILED
Call failed.

Reason (MQLONG) – output

Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQRC_SERVICE_ERROR
(2289, X'8F1') Unexpected error occurred accessing service.

MQZ_COPY_ALL_AUTHORITY

MQRC_SERVICE_NOT_AVAILABLE

(2285, X'8ED') Underlying service not available.

MQRC_UNKNOWN_REF_OBJECT

(2294, X'8F6') Reference object unknown.

For more information on these reason codes, see the *WebSphere MQ Application Programming Reference*.

C invocation

```
MQZ_COPY_ALL_AUTHORITY (QMgrName, RefObjectName, ObjectName, ObjectType,  
                        ComponentData, &Continuation, &CompCode,  
                        &Reason);
```

The parameters passed to the service are declared as follows:

```
MQCHAR48 QMgrName;          /* Queue manager name */  
MQCHAR48 RefObjectName;     /* Reference object name */  
MQCHAR48 ObjectName;       /* Object name */  
MQLONG   ObjectType;       /* Object type */  
MQBYTE   ComponentData[n]; /* Component data */  
MQLONG   Continuation;     /* Continuation indicator set by  
                           component */  
MQLONG   CompCode;        /* Completion code */  
MQLONG   Reason;         /* Reason code qualifying CompCode */
```

MQZ_DELETE_AUTHORITY – Delete authority

This function is provided by an authorization service component, and is invoked by the queue manager to delete all of the authorizations associated with the specified object.

The function identifier for this function (for MQZEP) is MQZID_DELETE_AUTHORITY.

Syntax

MQZ_DELETE_AUTHORITY (*QMgrName*, *ObjectName*, *ObjectType*,
ComponentData, *Continuation*, *CompCode*, *Reason*)

Parameters

The MQZ_DELETE_AUTHORITY call has the following parameters.

QMgrName (MQCHAR48) – input

Queue manager name.

The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue-manager name is passed to the component for information; the authorization service interface does not require the component to make use of it in any defined manner.

ObjectName (MQCHAR48) – input

Object name.

The name of the object for which accesses are to be deleted. The maximum length of the string is 48 characters; if it is shorter than that it is padded to the right with blanks. The name is not terminated by a null character.

If *ObjectType* is MQOT_Q_MGR, this name is the same as *QMgrName*.

ObjectType (MQLONG) – input

Object type.

The type of entity specified by *ObjectName*. It is one of the following:

MQOT_AUTH_INFO

Authentication information.

MQOT_CHANNEL

Channel.

MQOT_CLNTCONN_CHANNEL

Client connection channel.

MQOT_LISTENER

Listener.

MQOT_NAMELIST

Namelist.

MQOT_PROCESS

Process definition.

MQOT_Q

Queue.

MQZ_DELETE_AUTHORITY

MQOT_Q_MGR
Queue manager.
MQOT_SERVICE
Service.

ComponentData (MQBYTE×ComponentDataLength) – input/output

Component data.

This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions provided by this component are preserved, and presented the next time one of this component's functions is called.

The length of this data area is passed by the queue manager in the *ComponentDataLength* parameter of the MQZ_INIT_AUTHORITY call.

Continuation (MQLONG) – output

Continuation indicator set by component.

The following values can be specified:

MQZCI_DEFAULT
Continuation dependent on queue manager.
For MQZ_DELETE_AUTHORITY this has the same effect as MQZCI_STOP.

MQZCI_CONTINUE
Continue with next component.

MQZCI_STOP
Do not continue with next component.

CompCode (MQLONG) – output

Completion code.

It is one of the following:

MQCC_OK
Successful completion.
MQCC_FAILED
Call failed.

Reason (MQLONG) – output

Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:
MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:
MQRC_SERVICE_ERROR
(2289, X'8F1') Unexpected error occurred accessing service.
MQRC_SERVICE_NOT_AVAILABLE
(2285, X'8ED') Underlying service not available.

For more information on these reason codes, see the *WebSphere MQ Application Programming Reference*.

C invocation

```
MQZ_DELETE_AUTHORITY (QMgrName, ObjectName, ObjectType, ComponentData,  
                      &Continuation, &CompCode, &Reason);
```

The parameters passed to the service are declared as follows:

```
MQCHAR48 QMgrName;          /* Queue manager name */  
MQCHAR48 ObjectName;       /* Object name */  
MQLONG   ObjectType;       /* Object type */  
MQBYTE   ComponentData[n]; /* Component data */  
MQLONG   Continuation;     /* Continuation indicator set by  
                           component */  
MQLONG   CompCode;        /* Completion code */  
MQLONG   Reason;         /* Reason code qualifying CompCode */
```

MQZ_ENUMERATE_AUTHORITY_DATA – Enumerate authority data

This function is provided by an MQZAS_VERSION_4 authorization service component, and is invoked repeatedly by the queue manager to retrieve all of the authority data that matches the selection criteria specified on the first invocation.

The function identifier for this function (for MQZEP) is MQZID_ENUMERATE_AUTHORITY_DATA.

Syntax

```
MQZ_ENUMERATE_AUTHORITY_DATA (QMgrName, StartEnumeration,
                               Filter, AuthorityBufferLength, AuthorityBuffer, AuthorityDataLength,
                               ComponentData, Continuation, CompCode, Reason)
```

Parameters

The MQZ_ENUMERATE_AUTHORITY_DATA call has the following parameters.

QMgrName (MQCHAR48) – input

Queue manager name.

The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue-manager name is passed to the component for information; the authorization service interface does not require the component to make use of it in any defined manner.

StartEnumeration (MQLONG) – input

Flag indicating whether call should start enumeration.

This indicates whether the call should start the enumeration of authority data, or continue the enumeration of authority data started by a previous call to MQZ_ENUMERATE_AUTHORITY_DATA. The value is one of the following:

MQZSE_START

Start enumeration.

The call is invoked with this value to start the enumeration of authority data. The *Filter* parameter specifies the selection criteria to be used to select the authority data returned by this and successive calls.

MQZSE_CONTINUE

Continue enumeration.

The call is invoked with this value to continue the enumeration of authority data. The *Filter* parameter is ignored in this case, and can be specified as the null pointer (the selection criteria are determined by the *Filter* parameter specified by the call that had *StartEnumeration* set to MQZSE_START).

Filter (MQZAD) – input

Filter.

If *StartEnumeration* is MQZSE_START, *Filter* specifies the selection criteria to be used to select the authority data to return. If *Filter* is the null pointer, no selection

criteria are used, that is, all authority data is returned. See “MQZAD – Authority data” on page 478 for details of the selection criteria that can be used.

If *StartEnumeration* is MQZSE_CONTINUE, *Filter* is ignored, and can be specified as the null pointer.

AuthorityBufferLength (MQLONG) – input

Length of *AuthorityBuffer*.

This is the length in bytes of the *AuthorityBuffer* parameter. The authority buffer must be big enough to accommodate the data to be returned.

AuthorityBuffer (MQZAD) – output

Authority data.

This is the buffer in which the authority data is returned. The buffer must be big enough to accommodate an MQZAD structure, an MQZED structure, plus the longest entity name and longest domain name defined.

Note: This parameter is defined as an MQZAD, as the MQZAD always occurs at the start of the buffer. However, if the buffer is actually declared as an MQZAD, the buffer will be too small – it needs to be bigger than an MQZAD so that it can accommodate the MQZAD, MQZED, plus entity and domain names.

AuthorityDataLength (MQLONG) – output

Length of data returned in *AuthorityBuffer*.

This is the length of the data returned in *AuthorityBuffer*. If the authority buffer is too small, *AuthorityDataLength* is set to the length of the buffer required, and the call returns completion code MQCC_FAILED and reason code MQRC_BUFFER_LENGTH_ERROR.

ComponentData (MQBYTE×ComponentDataLength) – input/output

Component data.

This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions provided by this component are preserved, and presented the next time one of this component’s functions is called.

The length of this data area is passed by the queue manager in the *ComponentDataLength* parameter of the MQZ_INIT_AUTHORITY call.

Continuation (MQLONG) – output

Continuation indicator set by component.

The following values can be specified:

MQZCI_DEFAULT

Continuation dependent on queue manager.

For MQZ_ENUMERATE_AUTHORITY_DATA this has the same effect as MQZCI_CONTINUE.

MQZCI_CONTINUE

Continue with next component.

MQZ_ENUMERATE_AUTHORITY_DATA

MQZCI_STOP

Do not continue with next component.

CompCode (MQLONG) – output

Completion code.

It is one of the following:

MQCC_OK

Successful completion.

MQCC_FAILED

Call failed.

Reason (MQLONG) – output

Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE

(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQRC_BUFFER_LENGTH_ERROR

(2005, X'7D5') Buffer length parameter not valid.

MQRC_NO_DATA_AVAILABLE

(2379, X'94B') No data available.

MQRC_SERVICE_ERROR

(2289, X'8F1') Unexpected error occurred accessing service.

For more information on these reason codes, see the *WebSphere MQ Application Programming Reference*.

C invocation

```
MQZ_ENUMERATE_AUTHORITY_DATA (QMgrName, StartEnumeration, &Filter,  
                               AuthorityBufferLength,  
                               &AuthorityBuffer,  
                               &AuthorityDataLength, ComponentData,  
                               &Continuation, &CompCode,  
                               &Reason);
```

The parameters passed to the service are declared as follows:

```
MQCHAR48  QMgrName;           /* Queue manager name */  
MQLONG    StartEnumeration;   /* Flag indicating whether call should  
                               start enumeration */  
  
MQZAD     Filter;             /* Filter */  
MQLONG    AuthorityBufferLength; /* Length of AuthorityBuffer */  
MQZAD     AuthorityBuffer;    /* Authority data */  
MQLONG    AuthorityDataLength; /* Length of data returned in  
                               AuthorityBuffer */  
  
MQBYTE    ComponentData[n];   /* Component data */  
MQLONG    Continuation;       /* Continuation indicator set by  
                               component */  
  
MQLONG    CompCode;           /* Completion code */  
MQLONG    Reason;             /* Reason code qualifying CompCode */
```

MQZ_FREE_USER – Free user

This function is provided by a MQZAS_VERSION_5 authorization service component, and is invoked by the queue manager to free associated allocated resource. It is invoked when an application has finished running under all user contexts, for example during an MQDISC MQI call.

The function identifier for this function (for MQZEP) is MQZID_FREE_USER.

Syntax

```
MQZ_FREE_USER (QMgrName, FreeParms, ComponentData, Continuation,
              CompCode, Reason)
```

Parameters

The MQZ_FREE_USER call has the following parameters.

QMgrName (MQCHAR48) – input

Queue manager name.

The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue-manager name is passed to the component for information; the authorization service interface does not require the component to make use of it in any defined manner.

FreeParms (MQZFP) – input

Free parameters.

A structure containing data relating to the resource to be freed. See “MQZFP – Free parameters” on page 487 for details.

ComponentData (MQBYTE×ComponentDataLength) – input/output

Component data.

This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions provided by this component are preserved, and presented the next time one of this component’s functions is called.

Continuation (MQLONG) – output

Continuation flag.

The following values can be specified:

MQZCI_DEFAULT

Continuation dependent on other components.

MQZCI_STOP

Do not continue with next component.

CompCode (MQLONG) – output

Completion code.

It is one of the following:

MQZ_FREE_USER

MQCC_OK
Successful completion.
MQCC_FAILED
Call failed.

Reason (MQLONG) – output

Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQRC_SERVICE_ERROR
(2289, X'8F1') Unexpected error occurred accessing service.

C invocation

```
MQZ_AUTHENTICATE_USER (QMgrName, SecurityParms, ApplicationContext,  
                      IdentityContext, CorrelationPtr, ComponentData,  
                      &Continuation, &CompCode, &Reason);
```

The parameters passed to the service are declared as follows:

```
MQCHAR48  QMgrName;           /* Queue manager name */  
MQZFP     FreeParms;          /* Resource to be freed */  
MQBYTE    ComponentData[n];  /* Component data */  
MQLONG    Continuation;      /* Continuation indicator set by  
                             component */  
MQLONG    CompCode;          /* Completion code */  
MQLONG    Reason;            /* Reason code qualifying CompCode */
```

MQZ_GET_AUTHORITY – Get authority

This function is provided by a MQZAS_VERSION_1 authorization service component, and is invoked by the queue manager to retrieve the authority that an entity has to access the specified object.

The function identifier for this function (for MQZEP) is MQZID_GET_AUTHORITY.

Syntax

```
MQZ_GET_AUTHORITY (QMgrName, EntityName, EntityType, ObjectName,
                  ObjectType, Authority, ComponentData, Continuation, CompCode, Reason)
```

Parameters

The MQZ_GET_AUTHORITY call has the following parameters.

QMgrName (MQCHAR48) – input

Queue manager name.

The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue-manager name is passed to the component for information; the authorization service interface does not require the component to make use of it in any defined manner.

EntityName (MQCHAR12) – input

Entity name.

The name of the entity whose access to the object is to be retrieved. The maximum length of the string is 12 characters; if it is shorter than that it is padded to the right with blanks. The name is not terminated by a null character.

EntityType (MQLONG) – input

Entity type.

The type of entity specified by *EntityName*. The following value can be specified:

MQZAET_PRINCIPAL

Principal.

MQZAET_GROUP

Group.

ObjectName (MQCHAR48) – input

Object name.

The name of the object for which the entity's authority is to be retrieved. The maximum length of the string is 48 characters; if it is shorter than that it is padded to the right with blanks. The name is not terminated by a null character.

If *ObjectType* is MQOT_Q_MGR, this name is the same as *QMgrName*.

MQZ_GET_AUTHORITY

ObjectType (MQLONG) – input

Object type.

The type of entity specified by *ObjectName*. It is one of the following:

MQOT_AUTH_INFO

Authentication information.

MQOT_CHANNEL

Channel.

MQOT_CLNTCONN_CHANNEL

Client connection channel.

MQOT_LISTENER

Listener.

MQOT_NAMELIST

Namelist.

MQOT_PROCESS

Process definition.

MQOT_Q

Queue.

MQOT_Q_MGR

Queue manager.

MQOT_SERVICE

Service.

Authority (MQLONG) – output

Authority of entity.

If the entity has one authority, this field is equal to the appropriate authorization operation (MQZAO_* constant). If it has more than one authority, this field is the bitwise OR of the corresponding MQZAO_* constants.

ComponentData (MQBYTE×ComponentDataLength) – input/output

Component data.

This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions provided by this component are preserved, and presented the next time one of this component's functions is called.

The length of this data area is passed by the queue manager in the *ComponentDataLength* parameter of the MQZ_INIT_AUTHORITY call.

Continuation (MQLONG) – output

Continuation indicator set by component.

The following values can be specified:

MQZCI_DEFAULT

Continuation dependent on queue manager.

For MQZ_GET_AUTHORITY this has the same effect as MQZCI_CONTINUE.

MQZCI_CONTINUE

Continue with next component.

MQZCI_STOP

Do not continue with next component.

CompCode (MQLONG) – output

Completion code.

It is one of the following:

MQCC_OK

Successful completion.

MQCC_FAILED

Call failed.

Reason (MQLONG) – output

Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE

(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQRC_NOT_AUTHORIZED

(2035, X'7F3') Not authorized for access.

MQRC_SERVICE_ERROR

(2289, X'8F1') Unexpected error occurred accessing service.

MQRC_SERVICE_NOT_AVAILABLE

(2285, X'8ED') Underlying service not available.

MQRC_UNKNOWN_ENTITY

(2292, X'8F4') Entity unknown to service.

For more information on these reason codes, see the *WebSphere MQ Application Programming Reference*.

C invocation

```
MQZ_GET_AUTHORITY (QMgrName, EntityName, EntityType, ObjectName,
                  ObjectType, &Authority, ComponentData,
                  &Continuation, &CompCode, &Reason);
```

The parameters passed to the service are declared as follows:

```
MQCHAR48 QMgrName;          /* Queue manager name */
MQCHAR12 EntityName;        /* Entity name */
MQLONG   EntityType;        /* Entity type */
MQCHAR48 ObjectName;       /* Object name */
MQLONG   ObjectType;       /* Object type */
MQLONG   Authority;        /* Authority of entity */
MQBYTE   ComponentData[n]; /* Component data */
MQLONG   Continuation;     /* Continuation indicator set by
                           component */
MQLONG   CompCode;        /* Completion code */
MQLONG   Reason;         /* Reason code qualifying CompCode */
```

MQZ_GET_AUTHORITY_2 – Get authority (extended)

This function is provided by a MQZAS_VERSION_2 authorization service component, and is invoked by the queue manager to retrieve the authority that an entity has to access the specified object.

The function identifier for this function (for MQZEP) is MQZID_GET_AUTHORITY.

MQZ_GET_AUTHORITY_2 is similar to MQZ_GET_AUTHORITY, but with the *EntityName* parameter replaced by the *EntityData* parameter.

Syntax

```
MQZ_GET_AUTHORITY_2 (QMgrName, EntityData, EntityType, ObjectName,
                    ObjectType, Authority, ComponentData, Continuation, CompCode, Reason)
```

Parameters

The MQZ_GET_AUTHORITY_2 call has the following parameters.

QMgrName (MQCHAR48) – input

Queue manager name.

The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue-manager name is passed to the component for information; the authorization service interface does not require the component to make use of it in any defined manner.

EntityData (MQZED) – input

Entity data.

Data relating to the entity whose access to the object is to be retrieved. See “MQZED – Entity descriptor” on page 483 for details.

EntityType (MQLONG) – input

Entity type.

The type of entity specified by *EntityData*. The following value can be specified:

MQZAET_PRINCIPAL

Principal.

MQZAET_GROUP

Group.

ObjectName (MQCHAR48) – input

Object name.

The name of the object for which the entity’s authority is to be retrieved. The maximum length of the string is 48 characters; if it is shorter than that it is padded to the right with blanks. The name is not terminated by a null character.

If *ObjectType* is MQOT_Q_MGR, this name is the same as *QMgrName*.

ObjectType (MQLONG) – input

Object type.

The type of entity specified by *ObjectName*. It is one of the following:

MQOT_AUTH_INFO	Authentication information.
MQOT_CHANNEL	Channel.
MQOT_CLNTCONN_CHANNEL	Client connection channel.
MQOT_LISTENER	Listener.
MQOT_NAMELIST	Namelist.
MQOT_PROCESS	Process definition.
MQOT_Q	Queue.
MQOT_Q_MGR	Queue manager.
MQOT_SERVICE	Service.

Authority (MQLONG) – output

Authority of entity.

If the entity has one authority, this field is equal to the appropriate authorization operation (MQZAO_* constant). If it has more than one authority, this field is the bitwise OR of the corresponding MQZAO_* constants.

ComponentData (MQBYTE×ComponentDataLength) – input/output

Component data.

This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions provided by this component are preserved, and presented the next time one of this component's functions is called.

The length of this data area is passed by the queue manager in the *ComponentDataLength* parameter of the MQZ_INIT_AUTHORITY call.

Continuation (MQLONG) – output

Continuation indicator set by component.

The following values can be specified:

MQZCI_DEFAULT	Continuation dependent on queue manager. For MQZ_GET_AUTHORITY_2 this has the same effect as MQZCI_CONTINUE.
MQZCI_CONTINUE	Continue with next component.
MQZCI_STOP	Do not continue with next component.

MQZ_GET_AUTHORITY_2

CompCode (MQLONG) – output

Completion code.

It is one of the following:

MQCC_OK

Successful completion.

MQCC_FAILED

Call failed.

Reason (MQLONG) – output

Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE

(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQRC_NOT_AUTHORIZED

(2035, X'7F3') Not authorized for access.

MQRC_SERVICE_ERROR

(2289, X'8F1') Unexpected error occurred accessing service.

MQRC_SERVICE_NOT_AVAILABLE

(2285, X'8ED') Underlying service not available.

MQRC_UNKNOWN_ENTITY

(2292, X'8F4') Entity unknown to service.

For more information on these reason codes, see the *WebSphere MQ Application Programming Reference*.

C invocation

```
MQZ_GET_AUTHORITY_2 (QMgrName, &EntityData, EntityType, ObjectName,  
                    ObjectType, &Authority, ComponentData,  
                    &Continuation, &CompCode, &Reason);
```

The parameters passed to the service are declared as follows:

```
MQCHAR48  QMgrName;           /* Queue manager name */  
MQZED     EntityData;         /* Entity data */  
MQLONG    EntityType;         /* Entity type */  
MQCHAR48  ObjectName;        /* Object name */  
MQLONG    ObjectType;         /* Object type */  
MQLONG    Authority;          /* Authority of entity */  
MQBYTE    ComponentData[n];  /* Component data */  
MQLONG    Continuation;       /* Continuation indicator set by  
                               component */  
MQLONG    CompCode;           /* Completion code */  
MQLONG    Reason;            /* Reason code qualifying CompCode */
```

MQZ_GET_EXPLICIT_AUTHORITY – Get explicit authority

This function is provided by a MQZAS_VERSION_1 authorization service component, and is invoked by the queue manager to retrieve the authority that a named group has to access a specified object (but without the additional authority of the **nobody** group), or the authority that the primary group of the named principal has to access a specified object.

The function identifier for this function (for MQZEP) is MQZID_GET_EXPLICIT_AUTHORITY.

Syntax

```
MQZ_GET_EXPLICIT_AUTHORITY (QMgrName, EntityName, EntityType,
                             ObjectName, ObjectType, Authority, ComponentData, Continuation, CompCode,
                             Reason)
```

Parameters

The MQZ_GET_EXPLICIT_AUTHORITY call has the following parameters.

QMgrName (MQCHAR48) – input

Queue manager name.

The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue-manager name is passed to the component for information; the authorization service interface does not require the component to make use of it in any defined manner.

EntityName (MQCHAR12) – input

Entity name.

The name of the entity whose access to the object is to be retrieved. The maximum length of the string is 12 characters; if it is shorter than that it is padded to the right with blanks. The name is not terminated by a null character.

EntityType (MQLONG) – input

Entity type.

The type of entity specified by *EntityName*. The following value can be specified:

MQZAET_PRINCIPAL

Principal.

MQZAET_GROUP

Group.

ObjectName (MQCHAR48) – input

Object name.

The name of the object for which the entity's authority is to be retrieved. The maximum length of the string is 48 characters; if it is shorter than that it is padded to the right with blanks. The name is not terminated by a null character.

If *ObjectType* is MQOT_Q_MGR, this name is the same as *QMgrName*.

MQZ_GET_EXPLICIT_AUTHORITY

ObjectType (MQLONG) – input

Object type.

The type of entity specified by *ObjectName*. It is one of the following:

- MQOT_AUTH_INFO**
Authentication information.
- MQOT_CHANNEL**
Channel.
- MQOT_CLNTCONN_CHANNEL**
Client connection channel.
- MQOT_LISTENER**
Listener.
- MQOT_NAMELIST**
Namelist.
- MQOT_PROCESS**
Process definition.
- MQOT_Q**
Queue.
- MQOT_Q_MGR**
Queue manager.
- MQOT_SERVICE**
Service.

Authority (MQLONG) – output

Authority of entity.

If the entity has one authority, this field is equal to the appropriate authorization operation (MQZAO_* constant). If it has more than one authority, this field is the bitwise OR of the corresponding MQZAO_* constants.

ComponentData (MQBYTE×ComponentDataLength) – input/output

Component data.

This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions provided by this component are preserved, and presented the next time one of this component's functions is called.

The length of this data area is passed by the queue manager in the *ComponentDataLength* parameter of the MQZ_INIT_AUTHORITY call.

Continuation (MQLONG) – output

Continuation indicator set by component.

The following values can be specified:

- MQZCI_DEFAULT**
Continuation dependent on queue manager.
For MQZ_GET_EXPLICIT_AUTHORITY this has the same effect as MQZCI_CONTINUE.
- MQZCI_CONTINUE**
Continue with next component.
- MQZCI_STOP**
Do not continue with next component.

CompCode (MQLONG) – output

Completion code.

It is one of the following:

MQCC_OK

Successful completion.

MQCC_FAILED

Call failed.

Reason (MQLONG) – output

Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE

(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQRC_NOT_AUTHORIZED

(2035, X'7F3') Not authorized for access.

MQRC_SERVICE_ERROR

(2289, X'8F1') Unexpected error occurred accessing service.

MQRC_SERVICE_NOT_AVAILABLE

(2285, X'8ED') Underlying service not available.

MQRC_UNKNOWN_ENTITY

(2292, X'8F4') Entity unknown to service.

For more information on these reason codes, see the *WebSphere MQ Application Programming Reference*.

C invocation

```
MQZ_GET_EXPLICIT_AUTHORITY (QMgrName, EntityName, EntityType,
                             ObjectName, ObjectType, &Authority,
                             ComponentData, &Continuation,
                             &CompCode, &Reason);
```

The parameters passed to the service are declared as follows:

```
MQCHAR48 QMgrName;           /* Queue manager name */
MQCHAR12 EntityName;         /* Entity name */
MQLONG   EntityType;         /* Entity type */
MQCHAR48 ObjectName;        /* Object name */
MQLONG   ObjectType;        /* Object type */
MQLONG   Authority;         /* Authority of entity */
MQBYTE   ComponentData[n];  /* Component data */
MQLONG   Continuation;      /* Continuation indicator set by
                             component */
MQLONG   CompCode;          /* Completion code */
MQLONG   Reason;           /* Reason code qualifying CompCode */
```

MQZ_GET_EXPLICIT_AUTHORITY_2 – Get explicit authority (extended)

This function is provided by a MQZAS_VERSION_2 authorization service component, and is invoked by the queue manager to retrieve the authority that a named group has to access a specified object (but without the additional authority of the **nobody** group), or the authority that the primary group of the named principal has to access a specified object.

The function identifier for this function (for MQZEP) is MQZID_GET_EXPLICIT_AUTHORITY.

MQZ_GET_EXPLICIT_AUTHORITY_2 is similar to MQZ_GET_EXPLICIT_AUTHORITY, but with the *EntityName* parameter replaced by the *EntityData* parameter.

Syntax

```
MQZ_GET_EXPLICIT_AUTHORITY_2 (QMgrName, EntityData, EntityType,
                               ObjectName, ObjectType, Authority, ComponentData, Continuation, CompCode,
                               Reason)
```

Parameters

The MQZ_GET_EXPLICIT_AUTHORITY_2 call has the following parameters.

QMgrName (MQCHAR48) – input

Queue manager name.

The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue-manager name is passed to the component for information; the authorization service interface does not require the component to make use of it in any defined manner.

EntityData (MQZED) – input

Entity data.

Data relating to the entity whose access to the object is to be retrieved. See “MQZED – Entity descriptor” on page 483 for details.

EntityType (MQLONG) – input

Entity type.

The type of entity specified by *EntityData*. The following value can be specified:

MQZAET_PRINCIPAL

Principal.

MQZAET_GROUP

Group.

ObjectName (MQCHAR48) – input

Object name.

The name of the object for which the entity’s authority is to be retrieved. The maximum length of the string is 48 characters; if it is shorter than that it is padded to the right with blanks. The name is not terminated by a null character.

If *ObjectType* is MQOT_Q_MGR, this name is the same as *QMgrName*.

ObjectType (MQLONG) – input

Object type.

The type of entity specified by *ObjectName*. It is one of the following:

MQOT_AUTH_INFO

Authentication information.

MQOT_CHANNEL

Channel.

MQOT_CLNTCONN_CHANNEL

Client connection channel.

MQOT_LISTENER

Listener.

MQOT_NAMELIST

Namelist.

MQOT_PROCESS

Process definition.

MQOT_Q

Queue.

MQOT_Q_MGR

Queue manager.

MQOT_SERVICE

Service.

Authority (MQLONG) – output

Authority of entity.

If the entity has one authority, this field is equal to the appropriate authorization operation (MQZAO_* constant). If it has more than one authority, this field is the bitwise OR of the corresponding MQZAO_* constants.

ComponentData (MQBYTE×ComponentDataLength) – input/output

Component data.

This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions provided by this component are preserved, and presented the next time one of this component's functions is called.

The length of this data area is passed by the queue manager in the *ComponentDataLength* parameter of the MQZ_INIT_AUTHORITY call.

Continuation (MQLONG) – output

Continuation indicator set by component.

The following values can be specified:

MQZCI_DEFAULT

Continuation dependent on queue manager.

For MQZ_GET_EXPLICIT_AUTHORITY_2 this has the same effect as MQZCI_CONTINUE.

MQZCI_CONTINUE

Continue with next component.

MQZ_GET_EXPLICIT_AUTHORITY_2

MQZCI_STOP

Do not continue with next component.

CompCode (MQLONG) – output

Completion code.

It is one of the following:

MQCC_OK

Successful completion.

MQCC_FAILED

Call failed.

Reason (MQLONG) – output

Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE

(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQRC_NOT_AUTHORIZED

(2035, X'7F3') Not authorized for access.

MQRC_SERVICE_ERROR

(2289, X'8F1') Unexpected error occurred accessing service.

MQRC_SERVICE_NOT_AVAILABLE

(2285, X'8ED') Underlying service not available.

MQRC_UNKNOWN_ENTITY

(2292, X'8F4') Entity unknown to service.

For more information on these reason codes, see the *WebSphere MQ Application Programming Reference*.

C invocation

```
MQZ_GET_EXPLICIT_AUTHORITY_2 (QMgrName, &EntityData, EntityType,  
                               ObjectName, ObjectType, &Authority,  
                               ComponentData, &Continuation,  
                               &CompCode, &Reason);
```

The parameters passed to the service are declared as follows:

```
MQCHAR48  QMgrName;           /* Queue manager name */  
MQZED     EntityData;         /* Entity data */  
MQLONG    EntityType;         /* Entity type */  
MQCHAR48  ObjectName;         /* Object name */  
MQLONG    ObjectType;         /* Object type */  
MQLONG    Authority;          /* Authority of entity */  
MQBYTE    ComponentData[n];   /* Component data */  
MQLONG    Continuation;       /* Continuation indicator set by  
                               component */  
MQLONG    CompCode;           /* Completion code */  
MQLONG    Reason;             /* Reason code qualifying CompCode */
```

MQZ_INIT_AUTHORITY – Initialize authorization service

This function is provided by an authorization service component, and is invoked by the queue manager during configuration of the component. It is expected to call MQZEP in order to provide information to the queue manager.

The function identifier for this function (for MQZEP) is MQZID_INIT_AUTHORITY.

Syntax

MQZ_INIT_AUTHORITY (*Hconfig, Options, QMgrName, ComponentDataLength, ComponentData, Version, CompCode, Reason*)

Parameters

The MQZ_INIT_AUTHORITY call has the following parameters.

Hconfig (MQHCONFIG) – input

Configuration handle.

This handle represents the particular component being initialized. It is to be used by the component when calling the queue manager with the MQZEP function.

Options (MQLONG) – input

Initialization options.

It is one of the following:

MQZIO_PRIMARY

Primary initialization.

MQZIO_SECONDARY

Secondary initialization.

QMgrName (MQCHAR48) – input

Queue manager name.

The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue-manager name is passed to the component for information; the authorization service interface does not require the component to make use of it in any defined manner.

ComponentDataLength (MQLONG) – input

Length of component data.

Length in bytes of the *ComponentData* area. This length is defined in the component configuration data.

ComponentData (MQBYTE×ComponentDataLength) – input/output

Component data.

This is initialized to all zeroes before calling the component's primary initialization function. This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions (including the

MQZ_INIT_AUTHORITY

initialization function) provided by this component are preserved, and presented the next time one of this component's functions is called.

Version (MQLONG) – input/output

Version number.

On input to the initialization function, this identifies the *highest* version number that the queue manager supports. The initialization function must change this, if necessary, to the version of the interface which *it* supports. If on return the queue manager does not support the version returned by the component, it calls the component's MQZ_TERM_AUTHORITY function and makes no further use of this component.

The following values are supported:

MQZAS_VERSION_1

Version 1.

MQZAS_VERSION_2

Version 2.

MQZAS_VERSION_3

Version 3.

MQZAS_VERSION_4

Version 4.

MQZAS_VERSION_5

Version 5.

CompCode (MQLONG) – output

Completion code.

It is one of the following:

MQCC_OK

Successful completion.

MQCC_FAILED

Call failed.

Reason (MQLONG) – output

Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE

(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQRC_INITIALIZATION_FAILED

(2286, X'8EE') Initialization failed for an undefined reason.

MQRC_SERVICE_NOT_AVAILABLE

(2285, X'8ED') Underlying service not available.

For more information on these reason codes, see the *WebSphere MQ Application Programming Reference*.

C invocation

```
MQZ_INIT_AUTHORITY (Hconfig, Options, QMgrName, ComponentDataLength,  
                   ComponentData, &Version, &CompCode,  
                   &Reason);
```

The parameters passed to the service are declared as follows:

MQZ_INIT_AUTHORITY

```
MQHCONFIG Hconfig;          /* Configuration handle */
MQLONG    Options;          /* Initialization options */
MQCHAR48  QMgrName;         /* Queue manager name */
MQLONG    ComponentDataLength; /* Length of component data */
MQBYTE    ComponentData[n]; /* Component data */
MQLONG    Version;          /* Version number */
MQLONG    CompCode;         /* Completion code */
MQLONG    Reason;           /* Reason code qualifying CompCode */
```

MQZ_INQUIRE – Inquire authorization service

This function is provided by a MQZAS_VERSION_5 authorization service component, and is invoked by the queue manager to query the supported functionality. Where multiple service components are used, service components are called in reverse order to the order they were installed in.

The function identifier for this function (for MQZEP) is MQZID_INQUIRE.

Syntax

```
MQZ_INQUIRE
    (QMgrName, SelectorCount, Selectors, IntAttrCount, IntAttrs, CharAttrLength,
    CharAttrs, SelectorReturned, ComponentData, Continuation, CompCode, Reason)
```

Parameters

The MQZ_INQUIRE call has the following parameters.

QMgrName (MQCHAR48) – input

Queue manager name.

The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue-manager name is passed to the component for information; the authorization service interface does not require the component to make use of it in any defined manner.

SelectorCount (MQLONG) – input

Number of selectors.

The number of selectors supplied in the Selectors parameter.

The value must be between zero and 256.

Selectors (MQLONG×SelectorCount) – input

Selectors.

Array of selectors. Each selector identifies a required attribute and must be of one of the following types:

- MQIACF_* (integer)
- MQCACF_* (character)

Selectors can be specified in any order. The number of selectors in the array is indicated by the SelectorCount parameter.

Integer attributes identified by selectors are returned in the IntAttrs parameter in the same order as they appear in Selectors.

Character attributes identified by selectors are returned in the CharAttrs parameter in the same order as they appear in Selectors.

IntAttrCount (MQLONG) – input

Number of integer attributes.

The number of integer attributes supplied in the IntAttrs parameter.

The value must be between zero and 256.

IntAttrs (MQLONG×IntAttrCount) – output

Integer attributes.

Array of integer attributes. The integer attributes are returned in the same order as the corresponding integer selectors in the Selectors array.

CharAttrCount (MQLONG) – input

Length of the character attributes buffer.

The length in bytes of the CharAttrs parameter.

The value must at least sum of the lengths of the requested character attributes. If no character attributes are requested, zero is a valid value.

CharAttrs (MQLONG×CharAttrCount) – output

Character attributes buffer.

Buffer containing character attributes, concatenated together. The character attributes are returned in the same order as the corresponding character selectors in the Selectors array.

The length of the buffer is given by the CharAttrCount parameter.

SelectorReturned (MQLONG×SelectorCount) – input

Selector returned.

Array of values identifying which attributes have been returned from the set requested for by the selectors in the Selectors parameter. The number of values in this array is indicated by the SelectorCount parameter. Each value in the array relates to the selector from the corresponding position in the Selectors array. Each value is one of the following:

MQZSL_RETURNED

The attribute requested by the corresponding selector in the Selectors parameter has been returned.

MQZSL_NOT_RETURNED

The attribute requested by the corresponding selector in the Selectors parameter has not been returned.

The array is initialized with all values as MQZSL_NOT_RETURNED. When an authorization service component returns an attribute, it sets the appropriate value in the array to MQZSL_RETURNED. This allows any other authorization service components, to which the inquire call is made, to identify which attributes have already been returned.

ComponentData (MQBYTE×ComponentDataLength) – input/output

Component data.

MQZ_INQUIRE

This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions provided by this component are preserved, and presented the next time one of this component's functions is called.

The length of this data area is passed by the queue manager in the *ComponentDataLength* parameter of the MQZ_INIT_AUTHORITY call.

Continuation (MQLONG) – output

Continuation flag.

The following values can be specified:

MQZCI_DEFAULT

Continuation dependent on other components.

MQZCI_STOP

Do not continue with next component.

CompCode (MQLONG) – output

Completion code.

It is one of the following:

MQCC_OK

Successful completion.

MQCC_WARNING

Partial completion.

MQCC_FAILED

Call failed.

Reason (MQLONG) – output

Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE

(0, X'000') No reason to report.

If *CompCode* is MQCC_WARNING:

MQRC_CHAR_ATTRS_TOO_SHORT

Not enough space for character attributes.

MQRC_INT_COUNT_TOO_SMALL

Not enough space for integer attributes.

If *CompCode* is MQCC_FAILED:

MQRC_SELECTOR_COUNT_ERROR

Number of selectors is not valid.

MQRC_SELECTOR_ERROR

Attribute selector not valid.

MQRC_SELECTOR_LIMIT_EXCEEDED

Too many selectors specified.

MQRC_INT_ATTR_COUNT_ERROR

Number of integer attributes is not valid.

MQRC_INT_ATTRS_ARRAY_ERROR

Integer attributes array not valid.

MQRC_CHAR_ATTR_LENGTH_ERROR

Number of character attributes is not valid.

MQRC_CHAR_ATTRS_ERROR

Character attributes string is not valid.

MQRC_SERVICE_ERROR

(2289, X'8F1') Unexpected error occurred accessing service.

C invocation

```
MQZ_INQUIRE (QMgrName, SelectorCount, Selectors, IntAttrCount,
             &IntAttrs, CharAttrLength, &CharAttrs,
             SelectorReturned, ComponentData, &Continuation,
             &CompCode, &Reason);
```

The parameters passed to the service are declared as follows:

```
MQCHAR48  QMgrName;           /* Queue manager name */
MQLONG    SelectorCount;     /* Selector count */
MQLONG    Selectors[n];     /* Selectors */
MQLONG    IntAttrCount;     /* IntAttrs count */
MQLONG    IntAttrs[n];     /* Integer attributes */
MQLONG    CharAttrCount;    /* CharAttrs count */
MQLONG    CharAttrs[n];    /* Character attributes */
MQLONG    SelectorReturned[n]; /* Selector returned */
MQBYTE    ComponentData[n]; /* Component data */
MQLONG    Continuation;     /* Continuation indicator set by
                             component */
MQLONG    CompCode;         /* Completion code */
MQLONG    Reason;          /* Reason code qualifying CompCode */
```

MQZ_REFRESH_CACHE – Refresh all authorizations

This function is provided by an MQZAS_VERSION_3 authorization service component, and is invoked by the queue manager to refresh the list of authorizations held internally by the component.

The function identifier for this function (for MQZEP) is MQZID_REFRESH_CACHE (8L).

Syntax

MQZ_REFRESH_CACHE (*QMgrName*, *ComponentData*, *Continuation*, *CompCode*, *Reason*)

Parameters

QMgrName (MQCHAR48) — input
Queue manager name.

The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue-manager name is passed to the component for information; the authorization service interface does not require the component to make use of it in any defined manner.

ComponentData (MQBYTE×*ComponentDataLength*) — input/output
Component data.

This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions provided by this component are preserved, and presented the next time one of this component's functions is called.

The length of this data area is passed by the queue manager in the *ComponentDataLength* parameter of the MQZ_INIT_AUTHORITY call.

Continuation (MQLONG) — output
Continuation indicator set by component.

The following values can be specified:

MQZCI_DEFAULT

Continuation dependent on queue manager.

For MQZ_REFRESH_CACHE this has the same effect as MQZCI_CONTINUE.

MQZCI_CONTINUE

Continue with next component.

MQZCI_STOP

Do not continue with next component.

CompCode (MQLONG) — output
Completion code.

It is one of the following:

MQCC_OK

Successful completion.

MQCC_FAILED

Call failed.

Reason (MQLONG) — output
Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE

(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQRC_SERVICE_ERROR

(2289, X'8F1') Unexpected error occurred accessing service.

For more information on this reason code, see the *WebSphere MQ Application Programming Reference* book.

C invocation

```
MQZ_REFRESH_CACHE (QMgrName, ComponentData,
                  &Continuation, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQCHAR48  QMgrName;           /* Queue manager name */
MQBYTE    ComponentData[n];  /* Component data */
MQLONG    Continuation;      /* Continuation indicator set by
                             component */
MQLONG    CompCode;          /* Completion code */
MQLONG    Reason;            /* Reason code qualifying CompCode */
```

MQZ_SET_AUTHORITY – Set authority

This function is provided by a MQZAS_VERSION_1 authorization service component, and is invoked by the queue manager to set the authority that an entity has to access the specified object.

The function identifier for this function (for MQZEP) is MQZID_SET_AUTHORITY.

Note: This function overrides any existing authorities. To preserve any existing authorities you must set them again with this function.

Syntax

```
MQZ_SET_AUTHORITY (QMgrName, EntityName, EntityType, ObjectName,
                  ObjectType, Authority, ComponentData, Continuation, CompCode, Reason)
```

Parameters

The MQZ_SET_AUTHORITY call has the following parameters.

QMgrName (MQCHAR48) – input

Queue manager name.

The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue-manager name is passed to the component for information; the authorization service interface does not require the component to make use of it in any defined manner.

EntityName (MQCHAR12) – input

Entity name.

The name of the entity whose access to the object is to be set. The maximum length of the string is 12 characters; if it is shorter than that it is padded to the right with blanks. The name is not terminated by a null character.

EntityType (MQLONG) – input

Entity type.

The type of entity specified by *EntityName*. The following value can be specified:

MQZAET_PRINCIPAL

Principal.

MQZAET_GROUP

Group.

ObjectName (MQCHAR48) – input

Object name.

The name of the object to which access is required. The maximum length of the string is 48 characters; if it is shorter than that it is padded to the right with blanks. The name is not terminated by a null character.

If *ObjectType* is MQOT_Q_MGR, this name is the same as *QMgrName*.

ObjectType (MQLONG) – input

Object type.

The type of entity specified by *ObjectName*. It is one of the following:

MQOT_AUTH_INFO	Authentication information.
MQOT_CHANNEL	Channel.
MQOT_CLNTCONN_CHANNEL	Client connection channel.
MQOT_LISTENER	Listener.
MQOT_NAMELIST	Namelist.
MQOT_PROCESS	Process definition.
MQOT_Q	Queue.
MQOT_Q_MGR	Queue manager.
MQOT_SERVICE	Service.

Authority (MQLONG) – input

Authority to be checked.

If one authorization is being set, this field is equal to the appropriate authorization operation (MQZAO_* constant). If more than one authorization is being set, it is the bitwise OR of the corresponding MQZAO_* constants.

ComponentData (MQBYTE×ComponentDataLength) – input/output

Component data.

This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions provided by this component are preserved, and presented the next time one of this component's functions is called.

The length of this data area is passed by the queue manager in the *ComponentDataLength* parameter of the MQZ_INIT_AUTHORITY call.

Continuation (MQLONG) – output

Continuation indicator set by component.

The following values can be specified:

MQZCI_DEFAULT	Continuation dependent on queue manager. For MQZ_SET_AUTHORITY this has the same effect as MQZCI_STOP.
MQZCI_CONTINUE	Continue with next component.
MQZCI_STOP	Do not continue with next component.

MQZ_SET_AUTHORITY

CompCode (MQLONG) – output

Completion code.

It is one of the following:

MQCC_OK

Successful completion.

MQCC_FAILED

Call failed.

Reason (MQLONG) – output

Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE

(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQRC_NOT_AUTHORIZED

(2035, X'7F3') Not authorized for access.

MQRC_SERVICE_ERROR

(2289, X'8F1') Unexpected error occurred accessing service.

MQRC_SERVICE_NOT_AVAILABLE

(2285, X'8ED') Underlying service not available.

MQRC_UNKNOWN_ENTITY

(2292, X'8F4') Entity unknown to service.

For more information on these reason codes, see the *WebSphere MQ Application Programming Reference*.

C invocation

```
MQZ_SET_AUTHORITY (QMgrName, EntityName, EntityType, ObjectName,  
                  ObjectType, Authority, ComponentData,  
                  &Continuation, &CompCode, &Reason);
```

The parameters passed to the service are declared as follows:

```
MQCHAR48 QMgrName;          /* Queue manager name */  
MQCHAR12 EntityName;       /* Entity name */  
MQLONG   EntityType;       /* Entity type */  
MQCHAR48 ObjectName;      /* Object name */  
MQLONG   ObjectType;      /* Object type */  
MQLONG   Authority;       /* Authority to be checked */  
MQBYTE   ComponentData[n]; /* Component data */  
MQLONG   Continuation;    /* Continuation indicator set by  
                           component */  
MQLONG   CompCode;       /* Completion code */  
MQLONG   Reason;        /* Reason code qualifying CompCode */
```

MQZ_SET_AUTHORITY_2 – Set authority (extended)

This function is provided by a MQZAS_VERSION_2 authorization service component, and is invoked by the queue manager to set the authority that an entity has to access the specified object.

The function identifier for this function (for MQZEP) is MQZID_SET_AUTHORITY.

Note: This function overrides any existing authorities. To preserve any existing authorities you must set them again with this function.

MQZ_SET_AUTHORITY_2 is similar to MQZ_SET_AUTHORITY, but with the *EntityName* parameter replaced by the *EntityData* parameter.

Syntax

```
MQZ_SET_AUTHORITY_2 (QMgrName, EntityData, EntityType, ObjectName,
                    ObjectType, Authority, ComponentData, Continuation, CompCode, Reason)
```

Parameters

The MQZ_SET_AUTHORITY_2 call has the following parameters.

QMgrName (MQCHAR48) – input

Queue manager name.

The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue-manager name is passed to the component for information; the authorization service interface does not require the component to make use of it in any defined manner.

EntityData (MQZED) – input

Entity data.

Data relating to the entity whose access to the object is to be set. See “MQZED – Entity descriptor” on page 483 for details.

EntityType (MQLONG) – input

Entity type.

The type of entity specified by *EntityData*. The following value can be specified:

MQZAET_PRINCIPAL

Principal.

MQZAET_GROUP

Group.

ObjectName (MQCHAR48) – input

Object name.

The name of the object to which access is required. The maximum length of the string is 48 characters; if it is shorter than that it is padded to the right with blanks. The name is not terminated by a null character.

MQZ_SET_AUTHORITY_2

If *ObjectType* is MQOT_Q_MGR, this name is the same as *QMgrName*.

ObjectType (MQLONG) – input

Object type.

The type of entity specified by *ObjectName*. It is one of the following:

MQOT_AUTH_INFO

Authentication information.

MQOT_CHANNEL

Channel.

MQOT_CLNTCONN_CHANNEL

Client connection channel.

MQOT_LISTENER

Listener.

MQOT_NAMELIST

Namelist.

MQOT_PROCESS

Process definition.

MQOT_Q

Queue.

MQOT_Q_MGR

Queue manager.

MQOT_SERVICE

Service.

Authority (MQLONG) – input

Authority to be checked.

If one authorization is being set, this field is equal to the appropriate authorization operation (MQZAO_* constant). If more than one authorization is being set, it is the bitwise OR of the corresponding MQZAO_* constants.

ComponentData (MQBYTE×ComponentDataLength) – input/output

Component data.

This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions provided by this component are preserved, and presented the next time one of this component's functions is called.

The length of this data area is passed by the queue manager in the *ComponentDataLength* parameter of the MQZ_INIT_AUTHORITY call.

Continuation (MQLONG) – output

Continuation indicator set by component.

The following values can be specified:

MQZCI_DEFAULT

Continuation dependent on queue manager.

For MQZ_SET_AUTHORITY_2 this has the same effect as MQZCI_STOP.

MQZCI_CONTINUE

Continue with next component.

MQZCI_STOP

Do not continue with next component.

CompCode (MQLONG) – output

Completion code.

It is one of the following:

MQCC_OK

Successful completion.

MQCC_FAILED

Call failed.

Reason (MQLONG) – output

Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE

(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQRC_NOT_AUTHORIZED

(2035, X'7F3') Not authorized for access.

MQRC_SERVICE_ERROR

(2289, X'8F1') Unexpected error occurred accessing service.

MQRC_SERVICE_NOT_AVAILABLE

(2285, X'8ED') Underlying service not available.

MQRC_UNKNOWN_ENTITY

(2292, X'8F4') Entity unknown to service.

For more information on these reason codes, see the *WebSphere MQ Application Programming Reference*.

C invocation

```
MQZ_SET_AUTHORITY_2 (QMGrName, &EntityData, EntityType, ObjectName,
                    ObjectType, Authority, ComponentData,
                    &Continuation, &CompCode, &Reason);
```

The parameters passed to the service are declared as follows:

```
MQCHAR48  QMGrName;           /* Queue manager name */
MQZED     EntityData;         /* Entity data */
MQLONG    EntityType;         /* Entity type */
MQCHAR48  ObjectName;        /* Object name */
MQLONG    ObjectType;         /* Object type */
MQLONG    Authority;          /* Authority to be checked */
MQBYTE    ComponentData[n];  /* Component data */
MQLONG    Continuation;      /* Continuation indicator set by
                             component */
MQLONG    CompCode;           /* Completion code */
MQLONG    Reason;            /* Reason code qualifying CompCode */
```

MQZ_TERM_AUTHORITY – Terminate authorization service

This function is provided by an authorization service component, and is invoked by the queue manager when it no longer requires the services of this component. The function must perform any cleanup required by the component.

The function identifier for this function (for MQZEP) is MQZID_TERM_AUTHORITY.

Syntax

MQZ_TERM_AUTHORITY (Hconfig, Options, QMgrName, ComponentData, CompCode, Reason)

Parameters

The MQZ_TERM_AUTHORITY call has the following parameters.

Hconfig (MQHCONFIG) – input

Configuration handle.

This handle represents the particular component being terminated.

Options (MQLONG) – input

Termination options.

It is one of the following:

MQZTO_PRIMARY

Primary termination.

MQZTO_SECONDARY

Secondary termination.

QMgrName (MQCHAR48) – input

Queue manager name.

The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue-manager name is passed to the component for information; the authorization service interface does not require the component to make use of it in any defined manner.

ComponentData (MQBYTE×ComponentDataLength) – input/output

Component data.

This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions provided by this component are preserved, and presented the next time one of this component's functions is called.

The length of this data area is passed by the queue manager in the *ComponentDataLength* parameter on the MQZ_INIT_AUTHORITY call.

When the MQZ_TERM_AUTHORITY call has completed, the queue manager discards this data.

CompCode (MQLONG) – output

Completion code.

It is one of the following:

MQCC_OK

Successful completion.

MQCC_FAILED

Call failed.

Reason (MQLONG) – output

Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE

(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQRC_SERVICE_NOT_AVAILABLE

(2285, X'8ED') Underlying service not available.

MQRC_TERMINATION_FAILED

(2287, X'8FF') Termination failed for an undefined reason.

For more information on these reason codes, see the *WebSphere MQ Application Programming Reference*.

C invocation

```
MQZ_TERM_AUTHORITY (Hconfig, Options, QMgrName, ComponentData,
                    &CompCode, &Reason);
```

The parameters passed to the service are declared as follows:

```
MQHCONFIG  Hconfig;           /* Configuration handle */
MQLONG     Options;           /* Termination options */
MQCHAR48   QMgrName;         /* Queue manager name */
MQBYTE     ComponentData[n]; /* Component data */
MQLONG     CompCode;         /* Completion code */
MQLONG     Reason;           /* Reason code qualifying CompCode */
```

MQZAC – Application context

The following table summarizes the fields in the structure.

Table 27. Fields in MQZAC

Field	Description	Page
<i>StrucId</i>	Structure identifier	476
<i>Version</i>	Structure version number	476
<i>ProcessId</i>	Process identifier	477
<i>ThreadId</i>	Thread identifier	477
<i>ApplName</i>	Application name	477
<i>UserID</i>	For UNIX systems the application's real user ID, for Windows the application's user ID	477
<i>EffectiveUserID</i>	For UNIX systems the application's effective user ID, for Windows this field is all blank.	477
<i>Environment</i>	Environment from which the call was made	477
<i>CallerType</i>	Type of program from which the call was made	477
<i>AuthenticationType</i>	Type of authentication being performed	478
<i>BindType</i>	Type of bindings	478

The MQZAC structure is used on the MQZ_AUTHENTICIAE_USER call for the *ApplicationContext* parameter. This parameter specifies data related to the calling application

Fields

StrucId (MQCHAR4)

Structure identifier.

The value is:

MQZAC_STRUC_ID

Identifier for application context structure.

For the C programming language, the constant MQZAC_STRUC_ID_ARRAY is also defined; this has the same value as MQZAC_STRUC_ID, but is an array of characters instead of a string.

This is an input field to the service.

Version (MQLONG)

Structure version number.

The value is:

MQZAC_VERSION_1

Version-1 application context structure.

The following constant specifies the version number of the current version:

MQZAC_CURRENT_VERSION

Current version of application context structure.

This is an input field to the service.

ProcessId (MQPID)

Process identifier.

The process identifier of the application.

ThreadId (MQTID)

Thread identifier.

The thread identifier of the application.

AppName (MQCHAR28)

Application name.

The application name.

UserID (MQCHAR12)

User identifier.

On UNIX systems this field specifies the application's real user ID. On Windows this field specifies the application's user ID.

EffectiveUserID (MQCHAR12)

Effective user identifier.

On UNIX systems this field specifies the application's effective user ID. On Windows this field is blank.

Environment (MQLONG)

Environment.

This field specifies the environment from which the call was made.

The value is one of the following:

MQXE_COMMAND_SERVER

Command server.

MQXE_MQSC

runmqsc command interpreter.

MQXE_MCA

Message channel agent

MQXE_OTHER

Undefined environment

CallerType (MQLONG)

Caller Type.

This field specifies the type of program that made the call.

The value is one of the following:

MQXACT_EXTERNAL

The call is external to the queue manager.

MQXACT_INTERNAL

The call is internal to the queue manager.

MQZAC – Application context

AuthenticationType (MQLONG)

Authentication Type.

This field specifies the type of authentication being performed.

The value is one of the following:

MQZAT_INITIAL_CONTEXT

The authentication call is due to user context being initialized. This value is used during an MQCONN or MQCONNX call.

MQZAT_CHANGE_CONTEXT

The authentication call is due to the user context being changed. This value is used when the MCA changes the user context.

BindType (MQLONG)

Bind Type.

This field specifies the type of binding in use.

The value is one of the following:

MQCNO_FASTPATH_BINDING

Fastpath binding.

MQCNO_SHARED_BINDING

Shared binding.

MQCNO_ISOLATED_BINDING

Isolated binding.

C declaration

```
typedef struct tagMQZAC MQZAC;
struct tagMQZAC {
    MQCHAR4   StrucId;           /* Structure identifier */
    MQLONG    Version;          /* Structure version number */
    MQPID     ProcessId;        /* Process identifier */
    MQTID     ThreadId;         /* Thread identifier */
    MQCHAR28  ApplName;         /* Application name */
    MQCHAR12  UserID;           /* User identifier */
    MQCHAR12  EffectiveUserID;   /* Effective user identifier */
    MQLONG    Environment;      /* Environment */
    MQLONG    CallerType;       /* Caller type */
    MQLONG    AuthenticationType; /* Authentication type */
    MQLONG    BindType;         /* Bind type */
};
```

MQZAD – Authority data

The following table summarizes the fields in the structure.

Table 28. Fields in MQZAD

Field	Description	Page
<i>StrucId</i>	Structure identifier	479
<i>Version</i>	Structure version number	479
<i>ProfileName</i>	Profile name	479
<i>ObjectType</i>	Object type	480
<i>Authority</i>	Authority	480
<i>EntityDataPtr</i>	Address of an MQZED structure identifying an entity	480

Table 28. Fields in MQZAD (continued)

Field	Description	Page
<i>EntityType</i>	Type of entity	480
<i>Options</i>	Options field	481

The MQZAD structure is used on the MQZ_ENUMERATE_AUTHORITY_DATA call for two parameters:

- MQZAD is used for the *Filter* parameter which is input to the call. This parameter specifies the selection criteria that are to be used to select the authority data returned by the call.
- MQZAD is also used for the *AuthorityBuffer* parameter which is output from the call. This parameter specifies the authorizations for one combination of profile name, object type, and entity.

Fields

StrucId (MQCHAR4)

Structure identifier.

The value is:

MQZAD_STRUC_ID

Identifier for authority data structure.

For the C programming language, the constant MQZAD_STRUC_ID_ARRAY is also defined; this has the same value as MQZAD_STRUC_ID, but is an array of characters instead of a string.

This is an input field to the service.

Version (MQLONG)

Structure version number.

The value is:

MQZAD_VERSION_1

Version-1 authority data structure.

The following constant specifies the version number of the current version:

MQZAD_CURRENT_VERSION

Current version of authority data structure.

This is an input field to the service.

ProfileName (MQCHAR48)

Profile name.

For the *Filter* parameter, this field is the profile name whose authority data is required. If the name is entirely blank up to the end of the field or the first null character, authority data for all profile names is returned.

For the *AuthorityBuffer* parameter, this field is the name of a profile that matches the specified selection criteria.

MQZAD – Authority data

ObjectType (MQLONG)

Object type.

For the *Filter* parameter, this field is the object type for which authority data is required. If the value is MQOT_ALL, authority data for all object types is returned.

For the *AuthorityBuffer* parameter, this field is the object type to which the profile identified by *ProfileName* applies.

The value is one of the following; for the *Filter* parameter, the value MQOT_ALL is also valid:

MQOT_AUTH_INFO	Authentication information.
MQOT_CHANNEL	Channel.
MQOT_CLNTCONN_CHANNEL	Client connection channel.
MQOT_LISTENER	Listener.
MQOT_NAMELIST	Namelist.
MQOT_PROCESS	Process definition.
MQOT_Q	Queue.
MQOT_Q_MGR	Queue manager.
MQOT_SERVICE	Service.

Authority (MQLONG)

Authority.

For the *Filter* parameter, this field is ignored.

For the *AuthorityBuffer* parameter, this field represents the authorizations that the entity has to the objects identified by *ProfileName* and *ObjectType*. If the entity has only one authority, the field is equal to the appropriate authorization value (MQZAO_* constant). If the entity has more than one authority, the field is the bitwise OR of the corresponding MQZAO_* constants.

EntityDataPtr (PMQZED)

Address of MQZED structure identifying an entity.

For the *Filter* parameter, this field points to an MQZED structure that identifies the entity whose authority data is required. If *EntityDataPtr* is the null pointer, authority data for all entities is returned.

For the *AuthorityBuffer* parameter, this field points to an MQZED structure that identifies the entity whose authority data has been returned.

EntityType (MQLONG)

Entity type.

For the *Filter* parameter, this field specifies the entity type for which authority data is required. If the value is MQZAET_NONE, authority data for all entity types is returned.

For the *AuthorityBuffer* parameter, this field specifies the type of the entity identified by the MQZED structure pointed to by *EntityDataPtr*.

The value is one of the following; for the *Filter* parameter, the value MQZAET_NONE is also valid:

MQZAET_PRINCIPAL

Principal.

MQZAET_GROUP

Group.

Options (MQAUTHOPT)

Options.

This field specifies options that give control over the profiles that are displayed.

One of the following must be specified:

MQAUTHOPT_NAME_ALL_MATCHING

Displays all profiles

MQAUTHOPT_NAME_EXPLICIT

Displays profiles that have exactly the same name as specified in the *ProfileName* field.

In addition, one of the following must also be specified:

MQAUTHOPT_ENTITY_SET

Display all profiles used to calculate the cumulative authority that the entity has to the object specified by *ProfileName*. The *ProfileName* field must not contain any wildcard characters.

- If the specified entity is a principal, for each member of the set {entity, groups} the most applicable profile that applies to the object is displayed.
- If the specified entity is a group, the most applicable profile from the group that applies to the object is displayed.
- If this value is specified, then the values of *ProfileName*, *ObjectType*, *EntityType*, and the entity name specified in the *EntityDataPtr* MQZED structure, must all be non-blank.

If you have specified MQAUTHOPT_NAME_ALL_MATCHING, you can also specify the following:

MQAUTHOPT_ENTITY_EXPLICIT

Displays profiles that have exactly the same entity name as the entity name specified in the *EntityDataPtr* MQZED structure.

C declaration

```
typedef struct tagMQZAD MQZAD;
struct tagMQZAD {
    MQCHAR4  StrucId;          /* Structure identifier */
    MQLONG   Version;         /* Structure version number */
    MQCHAR48 ProfileName;     /* Profile name */
    MQLONG   ObjectType;     /* Object type */
    MQLONG   Authority;       /* Authority */
    PMQZED   EntityDataPtr;  /* Address of MQZED structure identifying an
```

MQZAD – Authority data

```
entity */
MQLONG   EntityType; /* Entity type */
MQAUTHOPT Options;  /* Options */
};
```

MQZED – Entity descriptor

The following table summarizes the fields in the structure.

Table 29. Fields in MQZED

Field	Description	Page
<i>StrucId</i>	Structure identifier	483
<i>Version</i>	Structure version number	483
<i>EntityNamePtr</i>	Address of entity name	483
<i>EntityDomainPtr</i>	Address of entity domain name	483
<i>SecurityId</i>	Security identifier	484
<i>CorrelationPtr</i>	Address of correlation data	484

The MQZED structure describes the information that is passed to the MQZAS_VERSION_2 authorization service calls.

Fields

StrucId (MQCHAR4)

Structure identifier.

The value is:

MQZED_STRUC_ID

Identifier for entity descriptor structure.

For the C programming language, the constant MQZED_STRUC_ID_ARRAY is also defined; this has the same value as MQZED_STRUC_ID, but is an array of characters instead of a string.

This is an input field to the service.

Version (MQLONG)

Structure version number.

The value is:

MQZED_VERSION_1

Version-1 entity descriptor structure.

The following constant specifies the version number of the current version:

MQZED_CURRENT_VERSION

Current version of entity descriptor structure.

This is an input field to the service.

EntityNamePtr (PMQCHAR)

Address of entity name.

This is a pointer to the name of the entity whose authorization is to be checked.

EntityDomainPtr (PMQCHAR)

Address of entity domain name.

MQZED – Entity descriptor

This is a pointer to the name of the domain containing the definition of the entity whose authorization is to be checked.

SecurityId (MQBYTE40)

Security identifier.

This is the security identifier whose authorization is to be checked.

CorrelationPtr (MQPTR)

Correlation pointer.

This facilitates the passing of correlational data between the authenticate user function and other appropriate OAM functions.

C declaration

```
typedef struct tagMQZED MQZED;
struct tagMQZED {
    MQCHAR4   StrucId;           /* Structure identifier */
    MQLONG    Version;          /* Structure version number */
    PMQCHAR   EntityNamePtr;     /* Address of entity name */
    PMQCHAR   EntityDomainPtr;  /* Address of entity domain name */
    MQBYTE40  SecurityId;        /* Security identifier */
    MQPTR     CorrelationPtr;    /* Address of correlation data */
}
```

MQZIC – Identity context

The following table summarizes the fields in the structure.

Table 30. Fields in MQZIC

Field	Description	Page
<i>StrucId</i>	Structure identifier	485
<i>Version</i>	Structure version number	485
<i>UserIdentifier</i>	User identifier	486
<i>AccountingToken</i>	Accounting token	486
<i>ApplIdentityData</i>	Application data relating to identity	486

The MQZIC structure is used on the MQZ_AUTHENTICATE_USER call for the *IdentityContext* parameter.

The MQZIC structure contains identity context information, that identifies the user of the application that first put the message on a queue:

- The queue manager fills the *UserIdentifier* field with a name that identifies the user, the way that the queue manager can do this depends on the environment in which the application is running.
- The queue manager fills the *AccountingToken* field with a token or number that it determined from the application that put the message.
- Applications can use the *ApplIdentityData* field for any extra information that they want to include about the user (for example, an encrypted password).

Suitably authorized applications may set the identity context using the MQZ_AUTHENTICATE_USER function.

A Windows systems security identifier (SID) is stored in the *AccountingToken* field when a message is created under WebSphere MQ for Windows. The SID can be used to supplement the *UserIdentifier* field and to establish the credentials of a user.

Fields

StrucId (MQCHAR4)

Structure identifier.

The value is:

MQZIC_STRUC_ID

Identifier for identity context structure.

For the C programming language, the constant MQZIC_STRUC_ID_ARRAY is also defined; this has the same value as MQZIC_STRUC_ID, but is an array of characters instead of a string.

This is an input field to the service.

Version (MQLONG)

Structure version number.

The value is:

MQZIC – Identity context

MQZIC_VERSION_1

Version-1 identity context structure.

The following constant specifies the version number of the current version:

MQZIC_CURRENT_VERSION

Current version of identity context structure.

This is an input field to the service.

UserIdentifier (MQCHAR12)

User identifier.

This is part of the **identity context** of the message.

UserIdentifier specifies the user identifier of the application that originated the message. The queue manager treats this information as character data, but does not define the format of it. For more information on the *UserIdentifier* field, see *WebSphere MQ Application Programming Reference*.

AccountingToken (MQBYTE32)

Accounting token.

This is part of the **identity context** of the message.

AccountingToken allows an application to cause work done as a result of the message to be appropriately charged. The queue manager treats this information as a string of bits and does not check its content. For more information on the *AccountingToken* field, see *WebSphere MQ Application Programming Reference*.

ApplIdentityData (MQCHAR32)

Application data relating to identity.

This is part of the **identity context** of the message.

ApplIdentityData is information that is defined by the application suite that can be used to provide additional information about the origin of the message. For example, it could be set by applications running with suitable user authority to indicate whether the identity data is trusted. For more information on the *ApplIdentityData* field, see *WebSphere MQ Application Programming Reference*.

C declaration

```
typedef struct tagMQZED MQZED;
struct tagMQZED {
    MQCHAR4    StrucId;           /* Structure identifier */
    MQLONG    Version;           /* Structure version number */
    MQCHAR12   UserIdentifier;    /* User identifier */
    MQBYTE32   AccountingToken;  /* Accounting token */
    MQCHAR32   ApplIdentityData; /* Application data relating to identity */
};
```

MQZFP – Free parameters

The following table summarizes the fields in the structure.

Table 31. Fields in MQZFP

Field	Description	Page
<i>StrucId</i>	Structure identifier	487
<i>Version</i>	Structure version number	487
<i>Reserved</i>	Reserved field	487
<i>CorrelationPtr</i>	Specifies the address of correlation data	487

The MQZFP structure is used on the MQZ_FREE_USER call for the *FreeParms* parameter. This parameter specifies data related to resource to be freed.

Fields

StrucId (MQCHAR4)

Structure identifier.

The value is:

MQZFP_STRUC_ID

Identifier for free parameters structure.

For the C programming language, the constant MQZFP_STRUC_ID_ARRAY is also defined; this has the same value as MQZFP_STRUC_ID, but is an array of characters instead of a string.

This is an input field to the service.

Version (MQLONG)

Structure version number.

The value is:

MQZFP_VERSION_1

Version-1 free parameters structure.

The following constant specifies the version number of the current version:

MQZFP_CURRENT_VERSION

Current version of free parameters structure.

This is an input field to the service.

Reserved (MQBYTE8)

Reserved field.

The initial value is null.

CorrelationPtr (MQPTR)

Correlation pointer.

Address of correlation data relating to the resource to be freed.

MQZFP – Free parameters

C declaration

```
typedef struct tagMQZFP MQZFP;  
struct tagMQZFP {  
    MQCHAR4   StrucId;           /* Structure identifier */  
    MQLONG    Version;          /* Structure version number */  
    MQBYTE8   Reserved;        /* Reserved field */  
    MQPTR     CorrelationPtr;   /* Address of correlation data */  
};
```

MQZ_DELETE_NAME – Delete name

This function is provided by a name service component, and is invoked by the queue manager to delete an entry for the specified queue.

The function identifier for this function (for MQZEP) is MQZID_DELETE_NAME.

Syntax

```
MQZ_DELETE_NAME (QMgrName, QName, ComponentData, Continuation,
                CompCode, Reason)
```

Parameters

The MQZ_DELETE_NAME call has the following parameters.

QMgrName (MQCHAR48) – input

Queue manager name.

The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue-manager name is passed to the component for information; the name service interface does not require the component to make use of it in any defined manner.

QName (MQCHAR48) – input

Queue name.

The name of the queue for which an entry is to be deleted. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

ComponentData (MQBYTE×ComponentDataLength) – input/output

Component data.

This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions provided by this component are preserved, and presented the next time one of this component's functions is called.

The length of this data area is passed by the queue manager in the *ComponentDataLength* parameter of the MQZ_INIT_NAME call.

Continuation (MQLONG) – output

Continuation indicator set by component.

For MQZ_DELETE_NAME, the queue manager does not attempt to invoke another component, whatever is returned in *Continuation*.

The following values can be specified:

MQZCI_DEFAULT

Continuation dependent on queue manager.

MQZCI_STOP

Do not continue with next component.

MQZ_DELETE_NAME

CompCode (MQLONG) – output

Completion code.

It is one of the following:

MQCC_OK

Successful completion.

MQCC_WARNING

Warning (partial completion).

MQCC_FAILED

Call failed.

Reason (MQLONG) – output

Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE

(0, X'000') No reason to report.

If *CompCode* is MQCC_WARNING:

MQRC_UNKNOWN_Q_NAME

(2288, X'8F0') Queue name not found.

Note: It may not be possible to return this code if the underlying service simply responds with success for this case.

If *CompCode* is MQCC_FAILED:

MQRC_SERVICE_ERROR

(2289, X'8F1') Unexpected error occurred accessing service.

MQRC_SERVICE_NOT_AVAILABLE

(2285, X'8ED') Underlying service not available.

For more information on these reason codes, see the *WebSphere MQ Application Programming Reference*.

C invocation

```
MQZ_DELETE_NAME (QMgrName, QName, ComponentData, &Continuation,  
                &CompCode, &Reason);
```

The parameters passed to the service are declared as follows:

```
MQCHAR48 QMgrName;          /* Queue manager name */  
MQCHAR48 QName;            /* Queue name */  
MQBYTE   ComponentData[n]; /* Component data */  
MQLONG   Continuation;     /* Continuation indicator set by  
                           component */  
MQLONG   CompCode;        /* Completion code */  
MQLONG   Reason;         /* Reason code qualifying CompCode */
```

MQZ_INIT_NAME – Initialize name service

This function is provided by a name service component, and is invoked by the queue manager during configuration of the component. It is expected to call MQZEP in order to provide information to the queue manager.

The function identifier for this function (for MQZEP) is MQZID_INIT_NAME.

Syntax

```
MQZ_INIT_NAME (Hconfig, Options, QMgrName, ComponentDataLength,
              ComponentData, Version, CompCode, Reason)
```

Parameters

The MQZ_INIT_NAME call has the following parameters.

Hconfig (MQHCONFIG) – input

Configuration handle.

This handle represents the particular component being initialized. It is to be used by the component when calling the queue manager with the MQZEP function.

Options (MQLONG) – input

Initialization options.

It is one of the following:

MQZIO_PRIMARY

Primary initialization.

MQZIO_SECONDARY

Secondary initialization.

QMgrName (MQCHAR48) – input

Queue manager name.

The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue-manager name is passed to the component for information; the name service interface does not require the component to make use of it in any defined manner.

ComponentDataLength (MQLONG) – input

Length of component data.

Length in bytes of the *ComponentData* area. This length is defined in the component configuration data.

ComponentData (MQBYTE×ComponentDataLength) – input/output

Component data.

This is initialized to all zeroes before calling the component's primary initialization function. This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions (including the

MQZ_INIT_NAME

initialization function) provided by this component are preserved, and presented the next time one of this component's functions is called.

Component data is in shared memory accessible to all processes. Therefore primary initialization is the first process initialization and secondary initialization is any subsequent process initialization.

Version (MQLONG) – input/output

Version number.

On input to the initialization function, this identifies the *highest* version number that the queue manager supports. The initialization function must change this, if necessary, to the version of the interface which *it* supports. If on return the queue manager does not support the version returned by the component, it calls the component's MQZ_TERM_NAME function and makes no further use of this component.

The following value is supported:

MQZNS_VERSION_1

Version 1.

CompCode (MQLONG) – output

Completion code.

It is one of the following:

MQCC_OK

Successful completion.

MQCC_FAILED

Call failed.

Reason (MQLONG) – output

Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE

(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQRC_INITIALIZATION_FAILED

(2286, X'8EE') Initialization failed for an undefined reason.

MQRC_SERVICE_NOT_AVAILABLE

(2285, X'8ED') Underlying service not available.

For more information on these reason codes, see the *WebSphere MQ Application Programming Reference*.

C invocation

```
MQZ_INIT_NAME (Hconfig, Options, QMgrName, ComponentDataLength,  
              ComponentData, &Version, &CompCode, &Reason);
```

The parameters passed to the service are declared as follows:

```
MQHCONFIG  Hconfig;           /* Configuration handle */  
MQLONG     Options;          /* Initialization options */  
MQCHAR48   QMgrName;        /* Queue manager name */  
MQLONG     ComponentDataLength; /* Length of component data */  
MQBYTE     ComponentData[n]; /* Component data */
```

```
MQLONG    Version;          /* Version number */
MQLONG    CompCode;        /* Completion code */
MQLONG    Reason;          /* Reason code qualifying CompCode */
```

MQZ_INSERT_NAME – Insert name

This function is provided by a name service component, and is invoked by the queue manager to insert an entry for the specified queue, containing the name of the queue manager that owns the queue. If the queue is already defined in the service, the call fails.

The function identifier for this function (for MQZEP) is MQZID_INSERT_NAME.

Syntax

```
MQZ_INSERT_NAME (QMgrName, QName, ResolvedQMgrName, ComponentData,  
                Continuation, CompCode, Reason)
```

Parameters

The MQZ_INSERT_NAME call has the following parameters.

QMgrName (MQCHAR48) – input

Queue manager name.

The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue-manager name is passed to the component for information; the name service interface does not require the component to make use of it in any defined manner.

QName (MQCHAR48) – input

Queue name.

The name of the queue for which an entry is to be inserted. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

ResolvedQMgrName (MQCHAR48) – input

Resolved queue manager name.

The name of the queue manager to which the queue resolves. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

ComponentData (MQBYTE×ComponentDataLength) – input/output

Component data.

This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions provided by this component are preserved, and presented the next time one of this component's functions is called.

The length of this data area is passed by the queue manager in the *ComponentDataLength* parameter of the MQZ_INIT_NAME call.

Continuation (MQLONG) – output

Continuation indicator set by component.

For MQZ_INSERT_NAME, the queue manager does not attempt to invoke another component, whatever is returned in *Continuation*.

The following values can be specified:

MQZCI_DEFAULT

Continuation dependent on queue manager.

MQZCI_STOP

Do not continue with next component.

CompCode (MQLONG) – output

Completion code.

It is one of the following:

MQCC_OK

Successful completion.

MQCC_FAILED

Call failed.

Reason (MQLONG) – output

Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE

(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQRC_Q_ALREADY_EXISTS

(2290, X'8F2') Queue object already exists.

MQRC_SERVICE_ERROR

(2289, X'8F1') Unexpected error occurred accessing service.

MQRC_SERVICE_NOT_AVAILABLE

(2285, X'8ED') Underlying service not available.

For more information on these reason codes, see the *WebSphere MQ Application Programming Reference*.

C invocation

```
MQZ_INSERT_NAME (QMgrName, QName, ResolvedQMgrName, ComponentData,
                &Continuation, &CompCode, &Reason);
```

The parameters passed to the service are declared as follows:

```
MQCHAR48 QMgrName;          /* Queue manager name */
MQCHAR48 QName;             /* Queue name */
MQCHAR48 ResolvedQMgrName; /* Resolved queue manager name */
MQBYTE   ComponentData[n]; /* Component data */
MQLONG   Continuation;     /* Continuation indicator set by
                           component */
MQLONG   CompCode;         /* Completion code */
MQLONG   Reason;          /* Reason code qualifying CompCode */
```

MQZ_LOOKUP_NAME – Lookup name

This function is provided by a name service component, and is invoked by the queue manager to retrieve the name of the owning queue manager, for a specified queue.

The function identifier for this function (for MQZEP) is MQZID_LOOKUP_NAME.

Syntax

```
MQZ_LOOKUP_NAME (QMgrName, QName, ResolvedQMgrName, ComponentData,  
Continuation, CompCode, Reason)
```

Parameters

The MQZ_LOOKUP_NAME call has the following parameters.

QMgrName (MQCHAR48) – input

Queue manager name.

The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue-manager name is passed to the component for information; the name service interface does not require the component to make use of it in any defined manner.

QName (MQCHAR48) – input

Queue name.

The name of the queue which is to be resolved. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

ResolvedQMgrName (MQCHAR48) – output

Resolved queue manager name.

If the function completes successfully, this is the name of the queue manager that owns the queue.

The name returned by the service component must be padded on the right with blanks to the full length of the parameter; the name *must not* be terminated by a null character, or contain leading or embedded blanks.

ComponentData (MQBYTE×ComponentDataLength) – input/output

Component data.

This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions provided by this component are preserved, and presented the next time one of this component's functions is called.

Component data is in shared memory accessible to all processes.

The length of this data area is passed by the queue manager in the *ComponentDataLength* parameter of the MQZ_INIT_NAME call.

Continuation (MQLONG) – output

Continuation indicator set by component.

For MQZ_LOOKUP_NAME, the queue manager decides whether to invoke another name service component, as follows:

- If *CompCode* is MQCC_OK, no further components are invoked, whatever value is returned in *Continuation*.
- If *CompCode* is not MQCC_OK, a further component is invoked, unless *Continuation* is MQZCI_STOP. This value should not be set without good reason.

The following values can be specified:

MQZCI_DEFAULT

Continuation dependent on queue manager.

MQZCI_CONTINUE

Continue with next component.

MQZCI_STOP

Do not continue with next component.

CompCode (MQLONG) – output

Completion code.

It is one of the following:

MQCC_OK

Successful completion.

MQCC_FAILED

Call failed.

Reason (MQLONG) – output

Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE

(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQRC_SERVICE_ERROR

(2289, X'8F1') Unexpected error occurred accessing service.

MQRC_SERVICE_NOT_AVAILABLE

(2285, X'8ED') Underlying service not available.

MQRC_UNKNOWN_Q_NAME

(2288, X'8F0') Queue name not found.

For more information on these reason codes, see the *WebSphere MQ Application Programming Reference*.

C invocation

```
MQZ_LOOKUP_NAME (QMgrName, QName, ResolvedQMgrName, ComponentData,
                &Continuation, &CompCode, &Reason);
```

The parameters passed to the service are declared as follows:

```
MQCHAR48 QMgrName;          /* Queue manager name */
MQCHAR48 QName;             /* Queue name */
MQCHAR48 ResolvedQMgrName;  /* Resolved queue manager name */
MQBYTE   ComponentData[n]; /* Component data */
MQLONG   Continuation;     /* Continuation indicator set by
```

MQZ_LOOKUP_NAME

```
MQLONG    CompCode;          component */
MQLONG    Reason;           /* Completion code */
                                /* Reason code qualifying CompCode */
```

MQZ_TERM_NAME – Terminate name service

This function is provided by a name service component, and is invoked by the queue manager when it no longer requires the services of this component. The function must perform any cleanup required by the component.

The function identifier for this function (for MQZEP) is MQZID_TERM_NAME.

Syntax

MQZ_TERM_NAME (*Hconfig, Options, QMgrName, ComponentData, CompCode, Reason*)

Parameters

The MQZ_TERM_NAME call has the following parameters.

Hconfig (MQHCONFIG) – input

Configuration handle.

This handle represents the particular component being terminated.

Options (MQLONG) – input

Termination options.

It is one of the following:

MQZTO_PRIMARY

Primary termination.

MQZTO_SECONDARY

Secondary termination.

QMgrName (MQCHAR48) – input

Queue manager name.

The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue-manager name is passed to the component for information; the name service interface does not require the component to make use of it in any defined manner.

ComponentData (MQBYTE×ComponentDataLength) – input/output

Component data.

This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions provided by this component are preserved, and presented the next time one of this component's functions is called.

Component data is in shared memory accessible to all processes.

The length of this data area is passed by the queue manager in the *ComponentDataLength* parameter on the MQZ_INIT_NAME call.

MQZ_TERM_NAME

When the MQZ_TERM_NAME call has completed, the queue manager discards this data.

CompCode (MQLONG) – output

Completion code.

It is one of the following:

MQCC_OK

Successful completion.

MQCC_FAILED

Call failed.

Reason (MQLONG) – output

Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE

(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQRC_SERVICE_NOT_AVAILABLE

(2285, X'8ED') Underlying service not available.

MQRC_TERMINATION_FAILED

(2287, X'8FF') Termination failed for an undefined reason.

For more information on these reason codes, see the *WebSphere MQ Application Programming Reference*.

C invocation

```
MQZ_TERM_NAME (Hconfig, Options, QMgrName, ComponentData, &CompCode,  
              &Reason);
```

The parameters passed to the service are declared as follows:

```
MQHCONFIG  Hconfig;           /* Configuration handle */  
MQLONG     Options;           /* Termination options */  
MQCHAR48   QMgrName;         /* Queue manager name */  
MQBYTE     ComponentData[n]; /* Component data */  
MQLONG     CompCode;         /* Completion code */  
MQLONG     Reason;           /* Reason code qualifying CompCode */
```

Chapter 23. 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.

This chapter explains why you might want to use API exits, then describes what administration tasks are involved in enabling them. The sections are:

- “Why use API exits”
- “How you use API exits”
- “What happens when an API exit runs?” on page 503
- “Configuring API exits” on page 503

We give a brief introduction to writing API exits in “How to write an API exit” on page 502. For detailed information about writing API exits, aimed at application programmers, see the *WebSphere MQ Application Programming Guide*.

Why use API exits

There are many reasons why you might want to insert code that modifies the behavior of applications at the level of the queue manager. Each of your applications has a specific job to do, and its code should do that task as efficiently as possible. At a higher level, you might want to apply standards or business processes to a particular queue manager for **all** the applications that use that queue manager. It is more efficient to do this above the level of individual applications, and thus without having to change the code of each application affected.

Here are a few suggestions of areas in which API exits might be useful:

- For *security*, you can provide authentication, checking that applications are authorized to access a queue or queue manager. You can also police applications’ use of the API, authenticating the individual API calls, or even the parameters they use.
- For *flexibility*, you can respond to rapid changes in your business environment without changing the applications that rely on the data in that environment. You could, for example, have API exits that respond to changes in interest rates, currency exchange rates, or the price of components in a manufacturing environment.
- For *monitoring* use of a queue or queue manager, you can trace the flow of applications and messages, log errors in the API calls, set up audit trails for accounting purposes, or collect usage statistics for planning purposes.

How you use API exits

This section gives a brief overview of the tasks involved in setting up API exits.

How to configure WebSphere MQ for API exits

You configure WebSphere MQ to enable API exits by changing the configuration information in the usual ways:

- On WebSphere MQ for Windows, use the WebSphere MQ Explorer or the **amqmdain** command to make changes to configuration information within the Windows Registry.
- On WebSphere MQ for Linux (x86 platform), use the WebSphere MQ Explorer to update the WebSphere MQ configuration files, `mqs.ini` and `qm.ini`.
- On other systems directly update WebSphere MQ configuration files, `mqs.ini` and `qm.ini`.

In either case, you provide information to:

- Name the API exit
- Identify the module and entry point of the API exit code to run
- Optionally pass data with the exit
- Identify the sequence of this exit in relation to other exits

For detailed information on this configuration, see “Configuring API exits” on page 503. For a description of how API exits run, see “What happens when an API exit runs?” on page 503.

How to write an API exit

This section introduces writing API exits. For detailed information, aimed at application programmers, see the *WebSphere MQ Application Programming Guide*.

You write your exits using the C programming language. To help you do so, we provide a sample exit, `amqsaxe0`, that generates trace entries to a named file. When you start writing exits, we recommend that you use this as your starting point.

Exits are available for every API call, as follows:

- `MQCONN/MQCONNX`, to provide a queue manager connection handle for use on subsequent API calls
- `MQDISC`, to disconnect from a queue manager
- `MQBEGIN`, to begin a global unit of work (UOW)
- `MQBACK`, to back out a UOW
- `MQCMIT`, to commit a UOW
- `MQOPEN`, to open an MQSeries resource for subsequent access
- `MQCLOSE`, to close an MQSeries resource that had previously been opened for access
- `MQGET`, to retrieve a message from a queue that has previously been opened for access
- `MQPUT1`, to place a message on to a queue
- `MQPUT`, to place a message on to a queue that has previously been opened for access
- `MQINQ`, to inquire on the attributes of an MQSeries resource that has previously been opened for access
- `MQSET`, to set the attributes of a queue that has previously been opened for access

Within API exits, these calls take the general form:

```
MQ_call_EXIT (parameters)
```

where `call` is the API call name (PUT, GET, and so on), and the parameters control the function of the exit, primarily providing communication between the exit and the external control blocks MQAXP (the exit parameter structure) and MQAXC (the exit context structure).

What happens when an API exit runs?

The API exit routines to run are identified in stanzas on UNIX systems, and in Registry entries on Windows systems. In this section we talk about the stanzas in the configuration files `mqs.ini` and `qm.ini`, however equivalent information is available in the Registry on Windows systems, accessible through the WebSphere MQ Explorer.

The definition of the routines can occur in three places:

1. `ApiExitCommon`, in the `mqs.ini` file, identifies routines, for the whole of WebSphere MQ, applied when queue managers start up. These can be overridden by routines defined for individual queue managers.
2. `ApiExitTemplate`, in the `mqs.ini` file, identifies routines, for the whole of WebSphere MQ, copied to the `ApiExitLocal` set when a new queue manager is created.
3. `ApiExitLocal`, in the `qm.ini` file, identifies routines applicable to a particular queue manager.

When a new queue manager is created, the `ApiExitTemplate` definitions in `mqs.ini` are copied to the `ApiExitLocal` definitions in `qm.ini` for the new queue manager. When a queue manager is started, both the `ApiExitCommon` and `ApiExitLocal` definitions are used. The `ApiExitLocal` definitions replace the `ApiExitCommon` definitions if both identify a routine of the same name. The `Sequence` attribute, described in “Attributes for all stanzas” on page 504 determines the order in which the routines defined in the stanzas run.

Configuring API exits

This section tells you how to configure API exits. We start in “Configuring API exits on UNIX systems,” explaining how to add the stanzas, followed by “Configuring API exits on Windows systems” on page 506, which tells you how to use the WebSphere MQ Explorer.

Configuring API exits on UNIX systems

You define your API exits in new stanzas in the `mqs.ini` and `qm.ini` files. The sections below describe these stanzas, and the attributes within them that define the exit routines and the sequence in which they run. For guidance on the process of changing these stanzas, see “Changing the configuration information” on page 505.

On Linux (x86 platform) systems you can use the WebSphere MQ Explorer to edit these entries in `mqs.ini` and `qm.ini`.

Stanzas in `mqs.ini` are:

ApiExitCommon

When any queue manager starts, the attributes in this stanza are read, and then overridden by the API exits defined in `qm.ini`.

ApiExitTemplate

When any queue manager is created, the attributes in this stanza are copied into the newly created qm.ini file under the ApiExitLocal stanza.

The stanza in qm.ini is:

ApiExitLocal

When the queue manager starts, API exits defined here override the defaults defined in mqs.ini.

Attributes for all stanzas

All these stanzas have the following attributes:

Name=ApiExit_name

The descriptive name of the API exit passed to it in the ExitInfoName field of the MQAXP structure.

This name must be unique, no longer than 48 characters, and contain only valid characters for the names of WebSphere MQ objects (for example, queue names).

Function=function_name

The name of the function entry point into the module containing the API exit code. This entry point is the MQ_INIT_EXIT function.

The length of this field is limited to MQ_EXIT_NAME_LENGTH.

Module=module_name

The module containing the API exit code.

If this field contains the full path name of the module it is used as is.

If this field contains just the module name, the module is located using the ExitsDefaultPath attribute in the ExitPath in qm.ini.

On platforms that support separate threaded libraries (AIX, HP/UX, and Linux), you must provide both a non-threaded and a threaded version of the API exit module. The threaded version must have an *_r* suffix. The threaded version of the WebSphere MQ application stub implicitly appends *_r* to the given module name before it is loaded.

The length of this field is limited to the maximum path length the platform supports.

Data=data_name

Data to be passed to the API exit in the ExitData field of the MQAXP structure.

If you include this attribute, leading and trailing blanks are removed, the remaining string is truncated to 32 characters, and the result is passed to the exit. If you omit this attribute, the default value of 32 blanks is passed to the exit.

The maximum length of this field is 32 characters.

Sequence=sequence_number

The sequence in which this API exit is called relative to other API exits. An exit with a **low** sequence number is called before an exit with a **higher** sequence number. There is no need for the sequence numbering of exits to be contiguous; a sequence of 1, 2, 3 has the same result as a sequence of 7, 42, 1096. If two exits have the same sequence number, the queue manager decides which one to call first. You can tell which was called *after* the event

by putting the time or a marker in ExitChainArea indicated by the ExitChainAreaPtr in MQAXP or by writing your own log file.

This attribute is an unsigned numeric value.

Sample stanzas

The mqs.ini file below contains the following stanzas:

ApiExitTemplate

This stanza defines an exit with the descriptive name OurPayrollQueueAuditor, module name auditor, and sequence number 2. A data value of 123 is passed to the exit.

ApiExitCommon

This stanza defines an exit with the descriptive name MQPoliceman, module name tmqp, and sequence number 1. The data passed is an instruction (CheckEverything).

mqs.ini

```
ApiExitTemplate:
  Name=OurPayrollQueueAuditor
  Sequence=2
  Function=EntryPoint
  Module=/usr/ABC/auditor
  Data=123
ApiExitCommon:
  Name=MQPoliceman
  Sequence=1
  Function=EntryPoint
  Module=/usr/MQPolice/tmqp
  Data=CheckEverything
```

The qm.ini file below contains an ApiExitLocal definition of an exit with the descriptive name ClientApplicationAPIchecker, module name ClientAppChecker, and sequence number 3.

qm.ini

```
ApiExitLocal:
  Name=ClientApplicationAPIchecker
  Sequence=3
  Function=EntryPoint
  Module=/usr/Dev/ClientAppChecker
  Data=9.20.176.20
```

Changing the configuration information

The WebSphere MQ configuration file, mqs.ini, contains information relevant to all the queue managers on a particular node. You can find it in the /var/mqm directory.

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, held in the root of the directory tree occupied by the queue manager. For example, the path and the name for a configuration file for a queue manager called QMNAME is:

```
/var/mqm/qmgrs/QMNAME/qm.ini
```

Before editing 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

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.)

Configuring API exits on Windows systems

You configure API exits on Windows systems using the WebSphere MQ Explorer or the **amqmdain** command to update the Windows Registry.

When using the WebSphere MQ Explorer you can modify the ApiExitCommon, ApiExitTemplate and ApiExitLocal stanzas as follows:

- The ApiExitCommon and ApiExitTemplate stanza are defined on the WebSphere MQ properties page, under Exits.
- The ApiExitLocal stanza is defined on the queue manager properties page, under Exits.

When entering, or changing, the attributes for an exit, the attributes are those defined in “Attributes for all stanzas” on page 504.

Chapter 24. API exit reference information

This chapter provides reference information for the API exit. It includes:

- Data structures used by an API exit function:
 - “MQACH – API exit chain header” on page 509
 - “MQAXC – API exit context” on page 512
 - “MQAXP – API exit parameter” on page 516
- Calls an API exit function can issue:
 - “MQXEP – Register entry point” on page 524
- Definitions of the API exit functions:
 - “MQ_BACK_EXIT – Back out changes” on page 527
 - “MQ_BEGIN_EXIT – Begin unit of work” on page 528
 - “MQ_CLOSE_EXIT – Close object” on page 529
 - “MQ_COMMIT_EXIT – Commit changes” on page 530
 - “MQ_CONNX_EXIT – Connect queue manager (extended)” on page 531
 - “MQ_DISC_EXIT – Disconnect queue manager” on page 533
 - “MQ_GET_EXIT – Get message” on page 534
 - “MQ_INIT_EXIT – Initialize exit environment” on page 536
 - “MQ_INQ_EXIT – Inquire object attributes” on page 537
 - “MQ_OPEN_EXIT – Open object” on page 539
 - “MQ_PUT_EXIT – Put message” on page 540
 - “MQ_PUT1_EXIT – Put one message” on page 542
 - “MQ_SET_EXIT – Set object attributes” on page 544
 - “MQ_TERM_EXIT – Terminate exit environment” on page 546

The data structures, calls, and exits are described in the order shown above (alphabetic order within each type).

General usage notes

This section contains general usage notes that relate to all API exit functions.

1. All exit functions can issue the MQXEP call; this call is designed specifically for use from API exit functions.
2. The MQ_INIT_EXIT function cannot issue any MQ calls other than MQXEP.
3. All other exit functions can issue the following MQ calls:
 - MQBACK, MQBEGIN, MQCLOSE, MQCMIT, MQCONN, MQCONNX, MQDISC, MQGET, MQINQ, MQOPEN, MQPUT, MQPUT1, MQSET
4. If an exit function issues the MQCONN call, or the MQCONNX call with the MQCNO_HANDLE_SHARE_NONE option, the call completes with reason code MQRC_ALREADY_CONNECTED, and the handle returned is the same as the one passed to the exit as a parameter.
5. In general when an API exit function issues an MQI call, API exits are not be called recursively. However, if an exit function issues the MQCONNX call with the MQCNO_HANDLE_SHARE_BLOCK or MQCNO_HANDLE_SHARE_NO_BLOCK options, the call returns a new shared handle. This provides the exit suite with a connection handle of its own, and hence a unit of work that is independent of the application’s unit of work. The exit suite can use this handle to put and get messages within its own unit of work, and commit or back out that unit of work; all of this can be done without affecting the application’s unit of work in any way.

API exit – General usage notes

Because the exit function is using a connection handle that is different from the handle being used by the application, MQ calls issued by the exit function result in the relevant API exit functions being invoked. Exit functions can therefore be invoked recursively. Note that both the *ExitUserArea* field in MQAXP and the exit chain area have connection-handle scope. Consequently, an exit function cannot use those areas to signal to another instance of itself invoked recursively that it is already active.

6. Exit functions can also put and get messages within the application's unit of work. When the application commits or backs out the unit of work, all messages within the unit of work are committed or backed out together, regardless of who placed them in the unit of work (application or exit function). However, the exit can cause the application to exceed system limits sooner than would otherwise be the case (for example, by exceeding the maximum number of uncommitted messages in a unit of work).

When an exit function uses the application's unit of work in this way, the exit function should usually avoid issuing the MQCMIT call, as this commits the application's unit of work and may impair the correct functioning of the application. However, the exit function may sometimes need to issue the MQBACK call, if the exit function encounters a serious error that prevents the unit of work being committed (for example, an error putting a message as part of the application's unit of work). When MQBACK is called, take care to ensure that the application unit of work boundaries are not changed. In this situation the exit function must set the appropriate values to ensure that completion code MQCC_FAILED and reason code MQRC_BACKED_OUT are returned to the application, so that the application can detect the fact that the unit of work has been backed out.

If an exit function uses the application's connection handle to issue MQ calls, those calls do not themselves result in further invocations of API exit functions.

7. If an MQXR_BEFORE exit function terminates abnormally, the queue manager may be able to recover from the failure. If it can, the queue manager continues processing as though the exit function had returned MQXCC_FAILED. If the queue manager cannot recover, the application is terminated.
8. If an MQXR_AFTER exit function terminates abnormally, the queue manager may be able to recover from the failure. If it can, the queue manager continues processing as though the exit function had returned MQXCC_FAILED. If the queue manager cannot recover, the application is terminated. Be aware that in the latter case, messages retrieved outside a unit of work are lost (this is the same situation as the application failing immediately after removing a message from the queue).
9. The MCA process performs a two phase commit.

If an API exit intercepts an MQCMIT from a prepared MCA process and attempts to perform an action within the unit of work, then the action will fail with reason code MQRC_UOW_NOT_AVAILABLE.

MQACH – API exit chain header

The following table summarizes the fields in the structure.

Table 32. Fields in MQACH

Field	Description	Page
<i>StrucId</i>	Structure identifier	509
<i>Version</i>	Structure version number	510
<i>StrucLength</i>	Length of MQACH structure	510
<i>ChainAreaLength</i>	Total length of chain area	510
<i>ExitInfoName</i>	Exit information name	511
<i>NextChainAreaPtr</i>	Address of next chain area	511

The MQACH structure describes the header information that must be present at the start of each exit chain area.

- The address of the first area in the chain is given by the *ExitChainAreaPtr* field in MQAXP. If there is no chain, *ExitChainAreaPtr* is the null pointer.
- The address of the next area in the chain is given by the *NextChainAreaPtr* field in MQACH. For the last area in the chain, *NextChainAreaPtr* is the null pointer.

Any exit function can create a chain area in dynamically-obtained storage (for example, by using `malloc`), and add that area to the chain at the desired location (start, middle, or end). The exit function must ensure that it sets all fields in MQACH to valid values.

The exit suite that creates the chain area is responsible for destroying that chain area before termination (the `MQ_TERM_EXIT` function is a convenient point at which to do this). However, adding and removing chain areas from the chain must be done only by an exit function when it is invoked by the queue manager; this restriction is necessary to avoid serialization problems.

Exit chain areas are made available to all exit suites, and must not be used to hold private data. Use *ExitUserArea* in MQAXP to hold private data.

In general there is no correspondence between the chain of exit functions that are invoked for an API call, and the chain of exit chain areas:

- Some exit functions might not have chain areas.
- Other exit functions might each have multiple chain areas.
- The order of the chain areas might be different from the order of the exit functions that own those chain areas.

Fields

The MQACH structure contains the following fields:

StrucId (MQCHAR4)

Structure identifier.

The value is:

MQACH_STRUC_ID

Identifier for API exit chain header structure.

MQACH – API exit chain header

For the C programming language, the constant `MQACH_STRUC_ID_ARRAY` is also defined; this has the same value as `MQACH_STRUC_ID`, but is an array of characters instead of a string.

This initial value of this field is `MQACH_STRUC_ID`.

Version (MQLONG)

Structure version number.

The value is:

MQACH_VERSION_1

Version-1 API exit chain header structure.

The following constant specifies the version number of the current version:

MQACH_CURRENT_VERSION

Current version of API exit chain header structure.

Note: When a new version of the MQACH structure is introduced, the layout of the existing part is not changed. The exit function must therefore check that the version number is equal to or greater than the lowest version that contains the fields that the exit function needs to use.

The initial value of this field is `MQACH_CURRENT_VERSION`.

StrucLength (MQLONG)

Length of MQACH structure.

This is the length of the MQACH structure itself; this length *excludes* the exit-defined data that follows the MQACH structure (see the *ChainAreaLength* field).

- The exit function that creates the MQACH structure must set this field to the length of the MQACH.
- An exit function that wants to access the exit-defined data should use *StrucLength* as the offset of the exit-defined data from the start of the MQACH structure.

The following value is defined:

MQACH_LENGTH_1

Length of version-1 MQACH structure.

The following constant specifies the length of the current version:

MQACH_CURRENT_LENGTH

Length of current version of exit chain area header.

The initial value of this field is `MQACH_CURRENT_LENGTH`.

ChainAreaLength (MQLONG)

Total length of chain area.

This is the total length of the chain area. It is equal to the sum of the length of the MQACH plus the length of the exit-defined data that follows the MQACH.

The initial value of this field is zero.

ExitInfoName (MQCHAR48)

Exit information name.

This is a name that is used to identify the exit suite to which the chain area belongs.

The length of this field is given by `MQ_EXIT_INFO_NAME_LENGTH`. The initial value of this field is the null string in C.

NextChainAreaPtr (PMQACH)

Address of next MQACH structure in chain.

This is the address of the next chain area in the chain. If the current chain area is the last one in the chain, *NextChainAreaPtr* is the null pointer.

The initial value of this field is the null pointer.

C declaration

```
typedef struct tagMQACH MQACH;
struct tagMQACH {
    MQCHAR4  StrucId;           /* Structure identifier */
    MQLONG   Version;          /* Structure version number */
    MQLONG   StrucLength;      /* Length of MQACH structure */
    MQLONG   ChainAreaLength; /* Total length of chain area */
    MQCHAR48 ExitInfoName;     /* Exit information name */
    PMQACH   NextChainAreaPtr; /* Address of next MQACH structure in
                                chain */
};
```

MQAXC – API exit context

The following table summarizes the fields in the structure.

Table 33. Fields in MQAXC

Field	Description	Page
<i>StrucId</i>	Structure identifier	512
<i>Version</i>	Structure version number	512
<i>Environment</i>	Environment	513
<i>UserId</i>	User identifier	513
<i>SecurityId</i>	Security identifier	513
<i>ConnectionName</i>	Connection name	514
<i>LongMCAUserIdLength</i>	Length of long MCA user identifier	514
<i>LongRemoteUserIdLength</i>	Length of long remote user identifier	514
<i>LongMCAUserIdPtr</i>	Address of long MCA user identifier	514
<i>LongRemoteUserIdPtr</i>	Address of long remote user identifier	514
<i>AppIName</i>	Application name	514
<i>AppIType</i>	Application type	514
<i>ProcessId</i>	Process identifier	515
<i>ThreadId</i>	Thread identifier	515

The MQAXC structure describes the context information that is passed to an API exit. The context information relates to the environment in which the application is running.

Fields

The MQAXC structure contains the following fields:

StrucId (MQCHAR4)

Structure identifier.

The value is:

MQAXC_STRUC_ID

Identifier for API exit parameter structure.

For the C programming language, the constant MQAXC_STRUC_ID_ARRAY is also defined; this has the same value as MQAXC_STRUC_ID, but is an array of characters instead of a string.

This is an input field to the exit.

Version (MQLONG)

Structure version number.

The value is:

MQAXC_VERSION_1

Version-1 API exit parameter structure.

The following constant specifies the version number of the current version:

MQAXC_CURRENT_VERSION

Current version of API exit parameter structure.

Note: When a new version of the MQAXC structure is introduced, the layout of the existing part is not changed. The exit should therefore check that the version number is equal to or greater than the lowest version which contains the fields that the exit needs to use.

This is an input field to the exit.

Environment (MQLONG)

Environment.

This indicates the environment from which the API call was issued. The value is one of the following:

MQXE_COMMAND_SERVER

Command server.

MQXE_MQSC

The “runmqsc” command interpreter.

MQXE_MCA

Message channel agent.

MQXE_MCA_SVRCONN

Message channel agent acting on behalf of a client.

MQXE_OTHER

Environment not defined.

This is an input field to the exit.

UserId (MQCHAR12)

User identifier.

This is the user identifier associated with the program that issued the API call. For a client connection (MQXE_MCA_SVRCONN), *UserId* contains the user identifier of the adopted user, and not the user identifier of the MCA.

The length of this field is given by MQ_USER_ID_LENGTH. This is an input field to the exit.

SecurityId (MQBYTE40)

Security identifier.

This is the security identifier associated with the program that issued the API call. For a client connection (MQXE_MCA_SVRCONN), *SecurityId* contains the security identifier of the adopted user, and not the security identifier of the MCA. If the security identifier is not known, *SecurityId* has the value:

MQSID_NONE

No security identifier specified.

The value is binary zero for the length of the field.

For the C programming language, the constant MQSID_NONE_ARRAY is also defined; this has the same value as MQSID_NONE, but is an array of characters instead of a string.

The length of this field is given by MQ_SECURITY_ID_LENGTH. This is an input field to the exit.

ConnectionName (MQCHAR264)

Connection name.

For a client connection (MQXE_MCA_SVRCONN), this field contains the address of the client (for example, the TCP/IP address). In other cases, this field is blank.

The length of this field is given by MQ_CONN_NAME_LENGTH. This is an input field to the exit.

LongMCAUserIdLength (MQLONG)

Length of long MCA user identifier.

For MQXE_MCA and MQXE_MCA_SVRCONN, this is the length in bytes of the full MCA user identifier pointed to by *LongMCAUserIdPtr*. In other cases, this field is zero.

This is an input field to the exit.

LongRemoteUserIdLength (MQLONG)

Length of long remote user identifier.

For MQXE_MCA and MQXE_MCA_SVRCONN, this is the length in bytes of the full remote user identifier pointed to by *LongRemoteUserIdPtr*. In other cases, this field is zero.

This is an input field to the exit.

LongMCAUserIdPtr (MQPTR)

Address of long MCA user identifier.

For MQXE_MCA and MQXE_MCA_SVRCONN, this is the address of the full MCA user identifier. The length of the full identifier is given by *LongMCAUserIdLength*. In other cases, this field is the null pointer.

This is an input field to the exit.

LongRemoteUserIdPtr (MQPTR)

Address of long remote user identifier.

For MQXE_MCA and MQXE_MCA_SVRCONN, this is the address of the full remote user identifier. The length of the full identifier is given by *LongRemoteUserIdLength*. In other cases, this field is the null pointer.

This is an input field to the exit.

AppName (MQCHAR28)

Application name.

This is the name of the application that issued the API call. This name is obtained in the same way as the default value for the *PutAppName* field in MQMD.

The length of this field is given by MQ_APPL_NAME_LENGTH. This is an input field to the exit.

AppType (MQLONG)

Application type.

This is the type of the application that issued the API call. The value is the same as MQAT_DEFAULT for the environment for which the application was compiled.

This is an input field to the exit.

ProcessId (MQPID)

The WebSphere MQ process identifier.

This is the same identifier used in WebSphere MQ trace and FFST dumps, but might be different from the operating system process identifier. Where applicable, the exit handler sets this field on entry to each exit function.

This is an input field to the exit.

ThreadId (MQTID)

The WebSphere MQ thread identifier.

This is the same identifier used in WebSphere MQ trace and FFST dumps, but might be different from the operating system thread identifier. Where applicable, the exit handler sets this field on entry to each exit function.

This is an input field to the exit.

C declaration

```
typedef struct tagMQAXC MQAXC;
struct tagMQAXC {
    MQCHAR4    StrucId;           /* Structure identifier */
    MQLONG     Version;          /* Structure version number */
    MQLONG     Environment;      /* Environment */
    MQCHAR12   UserId;           /* User identifier */
    MQBYTE40   SecurityId;       /* Security identifier */
    MQCHAR264  ConnectionName;   /* Connection name */
    MQLONG     LongMCAUserIdLength; /* Length of long MCA user
                                   identifier */
    MQLONG     LongRemoteUserIdLength; /* Length of long remote user
                                   identifier */
    MQPTR      LongMCAUserIdPtr;  /* Address of long MCA user
                                   identifier */
    MQPTR      LongRemoteUserIdPtr; /* Address of long remote user
                                   identifier */
    MQCHAR28   ApplName;         /* Application name */
    MQLONG     ApplType;         /* Application type */
    MQPID      ProcessId;        /* Process identifier */
    MQTID      ThreadId;         /* Thread identifier */
};
```

MQAXP – API exit parameter

The following table summarizes the fields in the structure.

Table 34. Fields in MQAXP

Field	Description	Page
<i>StrucId</i>	Structure identifier	516
<i>Version</i>	Structure version number	516
<i>ExitId</i>	Type of exit	517
<i>ExitReason</i>	Reason for invoking exit	517
<i>ExitResponse</i>	Response from exit	518
<i>ExitResponse2</i>	Secondary response from exit	519
<i>Feedback</i>	Feedback code	520
<i>APICallerType</i>	API caller type	520
<i>ExitUserArea</i>	Exit user area	520
<i>ExitData</i>	Exit data	521
<i>ExitInfoName</i>	Exit information name	521
<i>ExitPDArea</i>	Problem determination area	521
<i>QMGrName</i>	Name of local queue manager	521
<i>ExitChainAreaPtr</i>	Address of first chain area	521
<i>Hconfig</i>	Configuration handle	522
<i>Function</i>	API function identifier	522

The MQAXP structure describes the information that is passed to an API exit.

Fields

The MQAXP structure contains the following fields:

StrucId (MQCHAR4)

Structure identifier.

The value is:

MQAXP_STRUC_ID

Identifier for API exit parameter structure.

For the C programming language, the constant MQAXP_STRUC_ID_ARRAY is also defined; this has the same value as MQAXP_STRUC_ID, but is an array of characters instead of a string.

This is an input field to the exit.

Version (MQLONG)

Structure version number.

The value is:

MQAXP_VERSION_1

Version-1 API exit parameter structure.

The following constant specifies the version number of the current version:

MQAXP_CURRENT_VERSION

Current version of API exit parameter structure.

Note: When a new version of the MQAXP structure is introduced, the layout of the existing part is not changed. The exit should therefore check that the version number is equal to or greater than the lowest version which contains the fields that the exit needs to use.

This is an input field to the exit.

ExitId (MQLONG)

Type of exit.

This indicates the type of exit being called. The value is:

MQXT_API_EXIT

API exit.

This is an input field to the exit.

ExitReason (MQLONG)

Reason for invoking exit.

This indicates the reason why the exit is being called. Possible values are:

MQXR_CONNECTION

Connection level processing.

The exit is invoked with this value twice for each connection:

- Before the MQCONN or MQCONNX call, so that the exit can perform connection-level initialization. The *Function* field has the value MQXF_INIT in this case.

The MQXF_INIT exit function should be used for general initialization of the exit suite, and the MQXF_CONN or MQXF_CONNX exit functions should be used specifically for processing the MQCONN or MQCONNX calls.

- After the MQDISC call, so that the exit can perform connection-level termination. The *Function* field has the value MQXF_TERM in this case.

The MQXF_TERM exit function should be used for general termination of the exit suite, and the MQXF_DISC exit function should be used specifically for processing the MQDISC call.

MQXR_BEFORE

Before API execution.

The *Function* field can have any of the MQXF_* values other than MQXF_INIT or MQXF_TERM.

For the MQGET call, this value occurs with the:

- MQXF_GET exit function before API execution
- MQXF_DATA_CONV_ON_GET exit function after API execution but before data conversion

MQXR_AFTER

After API execution.

The *Function* field can have any of the MQXF_* values other than MQXF_INIT, MQXF_TERM, or MQXF_DATA_CONV_ON_GET.

For the MQGET call, this value occurs with the:

MQAXP – API exit parameter

- MQXF_GET exit function after both API execution and data conversion have been completed

This is an input field to the exit.

ExitResponse (MQLONG)

Response from exit.

This is set by the exit function to indicate the outcome of the processing performed by the exit. It must be one of the following:

MQXCC_OK

Exit completed successfully.

This value can be set by all MQXR_* exit functions. The *ExitResponse2* field must be set by the exit function to indicate how processing should continue.

Note: Returning MQXCC_OK does *not* imply that the completion code for the API call is MQCC_OK, or that the reason code is MQRC_NONE.

MQXCC_FAILED

Exit failed.

This value can be set by all MQXR_* exit functions. It causes the queue manager to set the completion code for the API call to MQCC_FAILED, and the reason code to one of the following values:

Exit function	Reason code set by queue manager
MQXF_INIT	MQRC_API_EXIT_INIT_ERROR
MQXF_TERM	MQRC_API_EXIT_TERM_ERROR
All others	MQRC_API_EXIT_ERROR

However, the values set by the queue manager can be altered by an exit function later in the chain.

The *ExitResponse2* field is ignored; the queue manager continues processing as though MQXR2_SUPPRESS_CHAIN had been returned:

- For an MQXR_BEFORE exit function, processing continues with the MQXR_AFTER exit function that matches this MQXR_BEFORE exit function (that is, all intervening MQXR_BEFORE and MQXR_AFTER exit functions, plus the API call itself, are skipped).
- For an MQXR_AFTER exit function, processing continues with the next MQXR_AFTER exit function in the chain.

MQXCC_SUPPRESS_FUNCTION

Suppress function.

If an MQXR_BEFORE exit function returns this value, the queue manager sets the completion code for the API call to MQCC_FAILED, the reason code to MQRC_SUPPRESSED_BY_EXIT, and the API call is skipped. If returned by the MQXF_DATA_CONV_ON_GET exit function, data conversion is skipped.

The *ExitResponse2* field must be set by the exit function to indicate whether the remaining MQXR_BEFORE exit functions and their matching MQXR_AFTER exit functions should be invoked. Any of these exit functions can alter the completion code and reason code of the API call that were set by the queue manager.

If an MQXR_AFTER or MQXR_CONNECTION exit function returns this value, the queue manager continues processing as though the exit had returned MQXCC_FAILED.

MQXCC_SKIP_FUNCTION

Skip function.

This is the same as MQXCC_SUPPRESS_FUNCTION, except the exit function can set the completion code and reason code of the API call.

MQXCC_SUPPRESS_EXIT

Suppress exit.

If an MQXR_BEFORE or MQXR_AFTER exit function returns this value, the queue manager deregisters immediately all of the exit functions belonging to this exit suite. The only exception is the MQXF_TERM exit function, which will be invoked at termination of the connection if registered when MQXCC_SUPPRESS_EXIT is returned. Note that if an MQXR_BEFORE exit function returns this value, the matching MQXR_AFTER exit function will *not* be invoked after the API call, since that exit function will no longer be registered.

The *ExitResponse2* field must be set by the exit function to indicate whether the remaining MQXR_BEFORE exit functions and their matching MQXR_AFTER exit functions should be invoked.

If an MQXR_CONNECTION exit function returns this value, the queue manager continues processing as though the exit had returned MQXCC_FAILED.

If the exit function sets *ExitResponse* to a value that is not valid, the queue manager continues processing as though the exit had returned MQXCC_FAILED.

On entry to the exit function, *ExitResponse* has the value MQXCC_OK.

This is an output field from the exit.

ExitResponse2 (MQLONG)

Secondary response from exit.

This is the secondary exit response code that can be set by an MQXR_BEFORE exit function to provide additional information to the queue manager. If set by an MQXR_AFTER or MQXR_CONNECTION exit function, the value is ignored. The value must be one of the following:

MQXR2_DEFAULT_CONTINUATION

Default continuation.

Continuation with the next exit function in the chain depends on the value of the *ExitResponse* field:

- If *ExitResponse* is MQXCC_OK or MQXCC_SUPPRESS_EXIT, the next MQXR_BEFORE exit function in the chain is invoked.
- If *ExitResponse* is MQXCC_SUPPRESS_FUNCTION or MQXCC_SKIP_FUNCTION, no further MQXR_BEFORE exit functions are invoked for this particular API call.

MQXR2_CONTINUE_CHAIN

Continue with next MQXR_BEFORE exit function in chain.

MQXR2_SUPPRESS_CHAIN

Skip remaining MQXR_BEFORE exit functions in chain.

MQAXP – API exit parameter

All subsequent MQXR_BEFORE exit functions in the chain, and their matching MQXR_AFTER exit functions, are skipped for this particular API call. The MQXR_AFTER exit functions that match the current exit function and earlier MQXR_BEFORE exit functions are not skipped.

If the exit function sets *ExitResponse2* to a value that is not valid, the queue manager continues processing as though the exit had returned MQXR2_DEFAULT_CONTINUATION.

This is an output field from the exit.

Feedback (MQLONG)

Feedback.

This is a field that allows the exit functions belonging to an exit suite to communicate feedback codes both to each other, and to exit functions belonging to other exit suites. The field is initialized to MQFB_NONE before the first invocation of the first exit function in the first exit suite (the MQXF_INIT exit function), and thereafter any changes made to this field by exit functions are preserved across the invocations of the exit functions.

This is an input/output field to the exit.

APICallerType (MQLONG)

API caller type.

This indicates the type of program that issued the API call that caused the exit function to be invoked. The value is one of the following:

MQXACT_EXTERNAL

Caller is external to the queue manager.

MQXACT_INTERNAL

Caller is internal to the queue manager.

This is an input field to the exit.

ExitUserArea (MQBYTE16)

Exit user area.

This is a field that allows exit functions belonging to the same exit suite to share data with each other, but not with other exit suites. The field is initialized to MQXUA_NONE (binary zero) before the first invocation of the first exit function in the exit suite (the MQXF_INIT exit function), and thereafter any changes made to this field by exit functions are preserved across the invocations of the exit functions. The queue manager resets the field to MQXUA_NONE when control returns from the MQXF_TERM exit function to the queue manager.

The following value is defined:

MQXUA_NONE

No user information.

The value is binary zero for the length of the field.

For the C programming language, the constant MQXUA_NONE_ARRAY is also defined; this has the same value as MQXUA_NONE, but is an array of characters instead of a string.

The length of this field is given by `MQ_EXIT_USER_AREA_LENGTH`. This is an input/output field to the exit.

ExitData (MQCHAR32)

Exit data.

On input to each exit function, this field is set to the character data associated with the definition of the exit suite to which the exit function belongs. If no value has been defined for that data, *ExitData* is blank.

The length of this field is given by `MQ_EXIT_DATA_LENGTH`. This is an input field to the exit.

ExitInfoName (MQCHAR48)

Exit information name.

This is a name that is used to identify the exit suite to which the exit function belongs.

The length of this field is given by `MQ_EXIT_INFO_NAME_LENGTH`. This is an input field to the exit.

ExitPDArea (MQBYTE48)

Problem determination area.

This is a field that is available for the exit to use, to assist with problem determination. The field is initialized to `MQXPDA_NONE` (binary zero) before each invocation of the exit function. The exit function can set this field to any value it chooses. When the exit returns control to the queue manager, the contents of *ExitPDArea* are written to the trace file, if tracing is active.

The following value is defined:

MQXPDA_NONE

No problem-determination information.

The value is binary zero for the length of the field.

For the C programming language, the constant `MQXPDA_NONE_ARRAY` is also defined; this has the same value as `MQXPDA_NONE`, but is an array of characters instead of a string.

The length of this field is given by `MQ_EXIT_PD_AREA_LENGTH`. This is an input/output field to the exit.

QMgrName (MQCHAR48)

Name of local queue manager.

This is the name of the queue manager that invoked the exit function. *QMgrName* is never blank.

The length of this field is given by `MQ_Q_MGR_NAME_LENGTH`. This is an input field to the exit.

ExitChainAreaPtr (PMQACH)

Address of first MQACH structure in chain.

MQAXP – API exit parameter

The exit chain area allows exit functions belonging to one exit suite to share data with exit functions belonging to another exit suite. The exit chain area is a chain of MQACH structures that is made available to all exit functions. The address of the first MQACH structure in the chain is passed to each exit function in the *ExitChainAreaPtr* field. The exit function can scan the chain, and examine or alter the data contained within it. However, this should be done only with the prior agreement of the owner of the data.

If there is no current exit chain area, *ExitChainAreaPtr* is the NULL pointer. An exit function can at any time create an MQACH structure in storage obtained dynamically (for example, by using the C function `malloc`), and add it to the chain. The exit suite which creates an MQACH is responsible for freeing the storage associated with the MQACH before the exit suite terminates.

If data is to be shared between different exit functions belonging to the same exit suite, but that data is *not* to be made available to other exit suites, the *ExitUserArea* field should be used in preference to *ExitChainAreaPtr*.

This is an input/output field to the exit.

Hconfig (MQHCONFIG)

Configuration handle.

This handle represents the set of exit functions that belong to the exit suite whose name is given by the *ExitInfoName* field. The queue manager generates a new configuration handle when the MQXF_INIT exit function is invoked, and passes that handle to the other exit functions that belong to the exit suite. This handle must be specified on the MQXEP call in order to register the entry point for an exit function.

This is an input field to the exit.

Function (MQLONG)

API function identifier.

This is the identifier of the API call that is about to be executed (when *ExitReason* has the value MQXR_BEFORE), or the API call that has just been executed (when *ExitReason* has the value MQXR_AFTER). If *ExitReason* has the value MQXR_CONNECTION, *Function* indicates whether the exit should perform initialization or termination. The value is one of the following:

MQXF_INIT

Initialization of exit suite.

MQXF_TERM

Termination of exit suite.

MQXF_CONN

MQCONN call.

MQXF_CONNX

MQCONNX call.

MQXF_DISC

MQDISC call.

MQXF_OPEN

MQOPEN call.

MQXF_CLOSE

MQCLOSE call.

MQXF_PUT1

MQPUT1 call.

MQXF_PUT
 MQPUT call.
MQXF_GET
 MQGET call.
MQXF_DATA_CONV_ON_GET
 Data conversion on MQGET call.
MQXF_INQ
 MQINQ call.
MQXF_SET
 MQSET call.
MQXF_BEGIN
 MQBEGIN call.
MQXF_CMIT
 MQCMIT call.
MQXF_BACK
 MQBACK call.

This is an input field to the exit.

C declaration

```

typedef struct tagMQAXP MQAXP;
struct tagMQAXP {
    MQCHAR4    StrucId;           /* Structure identifier */
    MQLONG     Version;          /* Structure version number */
    MQLONG     ExitId;           /* Type of exit */
    MQLONG     ExitReason;       /* Reason for invoking exit */
    MQLONG     ExitResponse;     /* Response from exit */
    MQLONG     ExitResponse2;    /* Secondary response from exit */
    MQLONG     Feedback;         /* Feedback */
    MQLONG     APICallerType;    /* API caller type */
    MQBYTE16   ExitUserArea;     /* Exit user area */
    MQCHAR32   ExitData;         /* Exit data */
    MQCHAR48   ExitInfoName;     /* Exit information name */
    MQBYTE48   ExitPDArea;       /* Problem determination area */
    MQCHAR48   QMgrName;         /* Name of local queue manager */
    PMQACH     ExitChainAreaPtr; /* Address of first MQACH structure in
                                chain */

    MQHCONFIG  Hconfig;          /* Configuration handle */
    MQLONG     Function;         /* API function identifier */
};
  
```

MQXEP – Register entry point

This call is used by an exit function to register the entry points of other exit functions in the exit suite. This is usually done by the MQ_INIT_EXIT function, but can be done by any exit function in the exit suite.

The MQXEP call is also used to deregister entry points. This is usually done by the MQ_TERM_EXIT function, but can be done by any exit function in the exit suite.

Syntax

```
MQXEP (Hconfig, ExitReason, Function, EntryPoint, Reserved,
      pCompCode, pReason)
```

Parameters

The MQXEP call has the following parameters.

Hconfig (MQHCONFIG) – input

Configuration handle.

This handle represents the exit suite to which the current exit function belongs. The queue manager generates this configuration handle when the MQ_INIT_EXIT function is invoked, and uses the *Hconfig* field in the MQAXP structure to pass the handle to each exit function in the exit suite.

ExitReason (MQLONG) – input

Exit reason.

This specifies when to call the entry point being registered or deregistered. It must be one of the following:

MQXR_CONNECTION

Connection level processing.

The *Function* parameter must have the value MQXF_INIT or MQXF_TERM.

MQXR_BEFORE

Before API execution.

The *Function* parameter can have any of the MQXF_* values other than MQXF_INIT or MQXF_TERM.

MQXR_AFTER

After API execution.

The *Function* parameter can have any of the MQXF_* values other than MQXF_INIT, MQXF_TERM, or MQXF_DATA_CONV_ON_GET.

Function (MQLONG) – input

Function identifier.

This specifies the API call for which the entry point is being registered or deregistered. It must be one of the following:

MQXF_INIT

Initialization of exit suite.

MQXF_TERM

Termination of exit suite.

MQXF_CONN
 MQCONN call.
MQXF_CONNX
 MQCONNX call.
MQXF_DISC
 MQDISC call.
MQXF_OPEN
 MQOPEN call.
MQXF_CLOSE
 MQCLOSE call.
MQXF_PUT1
 MQPUT1 call.
MQXF_PUT
 MQPUT call.
MQXF_GET
 MQGET call.
MQXF_DATA_CONV_ON_GET
 Data conversion on MQGET call.
MQXF_INQ
 MQINQ call.
MQXF_SET
 MQSET call.
MQXF_BEGIN
 MQBEGIN call.
MQXF_CMIT
 MQCMIT call.
MQXF_BACK
 MQBACK call.

If the MQXEP call is used more than once to register different entry points for a particular combination of *Function* and *ExitReason*, the last call made provides the entry point that is used.

EntryPoint (PMQFUNC) – input

Exit function entry point.

This is the address of the entry point being registered.

If the value specified is the null pointer, it indicates either that the exit function is not provided, or that a previously-registered exit function is being deregistered. The null pointer is assumed for entry points which are not defined using MQXEP.

Reserved (MQPTR) – input

Reserved.

This is a reserved parameter. The value specified must be the null pointer.

pCompCode (PMQLONG) – output

Completion code.

The value returned is one of the following:

MQCC_OK
 Successful completion.
MQCC_FAILED
 Call failed.

pReason (PMQLONG) – output

Reason code qualifying *pCompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE

(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQRC_EXIT_REASON_ERROR

(2377, X'949') Exit reason not valid.

MQRC_FUNCTION_ERROR

(2281, X'8E9') Function identifier not valid.

MQRC_HCONFIG_ERROR

(2280, X'8E8') Configuration handle not valid.

MQRC_RESERVED_VALUE_ERROR

(2378, X'94A') Reserved value not valid.

MQRC_RESOURCE_PROBLEM

(2102, X'836') Insufficient system resources available.

MQRC_UNEXPECTED_ERROR

(2195, X'893') Unexpected error occurred.

For more information on these reason codes, see the *WebSphere MQ Application Programming Reference*.

C invocation

```
MQXEP (Hconfig, ExitReason, Function, EntryPoint, Reserved,  
      &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHCONFIG Hconfig;    /* Configuration handle */  
MQLONG    ExitReason; /* Exit reason */  
MQLONG    Function;   /* Function identifier */  
PMQFUNC   EntryPoint; /* Exit function entry point */  
MQPTR     Reserved;   /* Reserved */  
PMQLONG   pCompCode;  /* Completion code */  
PMQLONG   pReason;    /* Reason code qualifying CompCode */
```

MQ_BACK_EXIT – Back out changes

Exit providers can supply an MQ_BACK_EXIT function to intercept the MQBACK call. If the unit of work is being coordinated by an external unit-of-work manager, MQ_BACK_EXIT is also invoked in response to the application issuing the unit-of-work manager's back-out call.

Syntax

```
MQ_BACK_EXIT (pExitParms, pExitContext, pHconn, pCompCode,
              pReason)
```

Parameters

The MQ_BACK_EXIT call has the following parameters.

pExitParms (PMQAXP) – input/output

Exit parameter structure.

pExitContext (PMQAXC) – input/output

Exit context structure.

pHconn (PMQHCONN) – input/output

Connection handle.

pCompCode (PMQLONG) – input/output

Completion code.

pReason (PMQLONG) – input/output

Reason code qualifying *pCompCode*.

C invocation

```
MQ_BACK_EXIT (&ExitParms, &ExitContext, &Hconn,
              &CompCode, &Reason);
```

The parameters passed to the exit are declared as follows:

```
PMQAXP    pExitParms;    /* Exit parameter structure */
PMQAXC    pExitContext;  /* Exit context structure */
PMQHCONN  pHconn;       /* Connection handle */
PMQLONG   pCompCode;    /* Completion code */
PMQLONG   pReason;      /* Reason code qualifying CompCode */
```

MQ_BEGIN_EXIT – Begin unit of work

Exit providers can supply an MQ_BEGIN_EXIT function to intercept the MQBEGIN call.

Syntax

```
MQ_BEGIN_EXIT (pExitParms, pExitContext, pHconn, ppBeginOptions,
              pCompCode, pReason)
```

Parameters

The MQ_BEGIN_EXIT call has the following parameters.

pExitParms (PMQAXP) – input/output

Exit parameter structure.

pExitContext (PMQAXC) – input/output

Exit context structure.

pHconn (PMQHCONN) – input/output

Connection handle.

ppBeginOptions (PPMQBO) – input/output

Options that control the action of MQBEGIN.

pCompCode (PMQLONG) – input/output

Completion code.

pReason (PMQLONG) – input/output

Reason code qualifying *pCompCode*.

C invocation

```
MQ_BEGIN_EXIT (&ExitParms, &ExitContext, &Hconn,
              &pBeginOptions, &CompCode, &Reason);
```

The parameters passed to the exit are declared as follows:

```
PMQAXP  pExitParms;      /* Exit parameter structure */
PMQAXC  pExitContext;    /* Exit context structure */
PMQHCONN pHconn;        /* Connection handle */
PPMQBO  ppBeginOptions; /* Options that control the action of
                        MQBEGIN */
PMQLONG pCompCode;      /* Completion code */
PMQLONG pReason;        /* Reason code qualifying CompCode */
```

MQ_CLOSE_EXIT – Close object

Exit providers can supply an MQ_CLOSE_EXIT function to intercept the MQCLOSE call.

Syntax

```
MQ_CLOSE_EXIT (pExitParms, pExitContext, pHconn, ppHobj, pOptions,
              pCompCode, pReason)
```

Parameters

The MQ_CLOSE_EXIT call has the following parameters.

pExitParms (PMQAXP) – input/output

Exit parameter structure.

pExitContext (PMQAXC) – input/output

Exit context structure.

pHconn (PMQHCONN) – input/output

Connection handle.

ppHobj (PPMQHOBJ) – input/output

Object handle.

pOptions (PMQLONG) – input/output

Options that control the action of MQCLOSE.

pCompCode (PMQLONG) – input/output

Completion code.

pReason (PMQLONG) – input/output

Reason code qualifying *pCompCode*.

C invocation

```
MQ_CLOSE_EXIT (&ExitParms, &ExitContext, &Hconn, &pHobj,
              &Options, &CompCode, &Reason);
```

The parameters passed to the exit are declared as follows:

```
PMQAXP    pExitParms;    /* Exit parameter structure */
PMQAXC    pExitContext;  /* Exit context structure */
PMQHCONN  pHconn;       /* Connection handle */
PPMQHOBJ  ppHobj;       /* Object handle */
PMQLONG   pOptions;     /* Options that control the action of MQCLOSE */
PMQLONG   pCompCode;    /* Completion code */
PMQLONG   pReason;      /* Reason code qualifying CompCode */
```

MQ_CMITS_EXIT – Commit changes

Exit providers can supply an MQ_CMITS_EXIT function to intercept the MQCMITS call. If the unit of work is being coordinated by an external unit-of-work manager, MQ_CMITS_EXIT is also invoked in response to the application issuing the unit-of-work manager's commit call.

Syntax

```
MQ_CMITS_EXIT (pExitParms, pExitContext, pHconn, pCompCode,
              pReason)
```

Parameters

The MQ_CMITS_EXIT call has the following parameters.

pExitParms (PMQAXP) – input/output

Exit parameter structure.

pExitContext (PMQAXC) – input/output

Exit context structure.

pHconn (PMQHCONN) – input/output

Connection handle.

pCompCode (PMQLONG) – input/output

Completion code.

pReason (PMQLONG) – input/output

Reason code qualifying *pCompCode*.

C invocation

```
MQ_CMITS_EXIT (&ExitParms, &ExitContext, &Hconn,
              &CompCode, &Reason);
```

The parameters passed to the exit are declared as follows:

```
PMQAXP    pExitParms;    /* Exit parameter structure */
PMQAXC    pExitContext;  /* Exit context structure */
PMQHCONN  pHconn;       /* Connection handle */
PMQLONG   pCompCode;    /* Completion code */
PMQLONG   pReason;      /* Reason code qualifying CompCode */
```

MQ_CONNX_EXIT – Connect queue manager (extended)

Exit providers can supply an MQ_CONNX_EXIT function to intercept the MQCONN and MQCONNX calls.

Syntax

```
MQ_CONNX_EXIT (pExitParms, pExitContext, pQMgrName, ppConnectOpts,
              ppHconn, pCompCode, pReason)
```

Parameters

The MQ_CONNX_EXIT call has the following parameters.

pExitParms (PMQAXP) – input/output

Exit parameter structure.

pExitContext (PMQAXC) – input/output

Exit context structure.

pQMgrName (PMQCHAR48) – input/output

Name of queue manager.

ppConnectOpts (PPMQCNO) – input/output

Options that control the action of MQCONNX.

ppHconn (PPMQHCONN) – input/output

Connection handle.

pCompCode (PMQLONG) – input/output

Completion code.

pReason (PMQLONG) – input/output

Reason code qualifying *pCompCode*.

Usage notes

1. The MQ_CONNX_EXIT function interface described here is used for both the MQCONN call and the MQCONNX call. However, separate entry points are defined for these two calls. To intercept *both* calls, the MQXEP call must be used at least twice – once with function identifier MQXF_CONN, and again with MQXF_CONNX.

Because the MQ_CONNX_EXIT interface is the same for MQCONN and MQCONNX, a single exit function can be used for both calls; the *Function* field in the MQAXP structure indicates which call is in progress. Alternatively, the MQXEP call can be used to register different exit functions for the two calls.

2. When a message channel agent (MCA) responds to an inbound client connection, the MCA can issue a number of MQ calls before the client state is fully known. These MQ calls result in the API exit functions being invoked with the MQAXC structure containing data relating to the MCA, and not to the client (for example, user identifier and connection name). However, once the client state is fully known, subsequent MQ calls result in the API exit functions being invoked with the appropriate client data in the MQAXC structure.
3. All MQXR_BEFORE exit functions are invoked before any parameter validation is performed by the queue manager. The parameters may therefore be invalid (including invalid pointers for the addresses of parameters).

MQ_CONNX_EXIT – Usage notes

The MQ_CONNX_EXIT function is invoked before any authorization checks are performed by the queue manager.

4. The exit function must not change the name of the queue manager specified on the MQCONN or MQCONNX call. If the name is changed by the exit function, the results are undefined.
5. An MQXR_BEFORE exit function for the MQ_CONNX_EXIT cannot issue MQ calls other than MQXEP.

C invocation

```
MQ_CONNX_EXIT (&ExitParms, &ExitContext, QMgrName,  
               &pConnectOpts, &pHconn, &CompCode,  
               &Reason);
```

The parameters passed to the exit are declared as follows:

```
PMQAXP    pExitParms;    /* Exit parameter structure */  
PMQAXC    pExitContext;  /* Exit context structure */  
PMQCHAR48 pQMgrName;     /* Name of queue manager */  
PPMQCNO   ppConnectOpts; /* Options that control the action of  
                        MQCONNX */  
PPMQHCONN ppHconn;       /* Connection handle */  
PMQLONG   pCompCode;     /* Completion code */  
PMQLONG   pReason;       /* Reason code qualifying CompCode */
```

MQ_DISC_EXIT – Disconnect queue manager

Exit providers can supply an MQ_DISC_EXIT function to intercept the MQDISC call.

Syntax

```
MQ_DISC_EXIT (pExitParms, pExitContext, ppHconn, pCompCode,
             pReason)
```

Parameters

The MQ_DISC_EXIT call has the following parameters.

pExitParms (PMQAXP) – input/output

Exit parameter structure.

pExitContext (PMQAXC) – input/output

Exit context structure.

ppHconn (PPMQHCONN) – input/output

Connection handle.

pCompCode (PMQLONG) – input/output

Completion code.

pReason (PMQLONG) – input/output

Reason code qualifying *pCompCode*.

C invocation

```
MQ_DISC_EXIT (&ExitParms, &ExitContext, &pHconn,
             &CompCode, &Reason);
```

The parameters passed to the exit are declared as follows:

```
PMQAXP    pExitParms;    /* Exit parameter structure */
PMQAXC    pExitContext;  /* Exit context structure */
PPMQHCONN ppHconn;      /* Connection handle */
PMQLONG   pCompCode;    /* Completion code */
PMQLONG   pReason;      /* Reason code qualifying CompCode */
```

MQ_GET_EXIT – Get message

Exit providers can supply an MQ_GET_EXIT function to intercept the MQGET call. The same exit function interface is used for the MQXF_DATA_CONV_ON_GET exit function.

Syntax

```
MQ_GET_EXIT (pExitParms, pExitContext, pHconn, pHobj, ppMsgDesc,
            ppGetMsgOpts, pBufferLength, ppBuffer, ppDataLength, pCompCode, pReason)
```

Parameters

The MQ_GET_EXIT call has the following parameters.

pExitParms (PMQAXP) – input/output

Exit parameter structure.

pExitContext (PMQAXC) – input/output

Exit context structure.

pHconn (PMQHCONN) – input/output

Connection handle.

pHobj (PMQHOBJ) – input/output

Object handle.

ppMsgDesc (PPMQMD) – input/output

Message descriptor.

ppGetMsgOpts (PPMQGMO) – input/output

Options that control the action of MQGET.

pBufferLength (PMQLONG) – input/output

Length in bytes of the *ppBuffer* area.

ppBuffer (PPMQVOID) – input/output

Area to contain the message data.

ppDataLength (PPMQLONG) – input/output

Length of the message.

pCompCode (PMQLONG) – input/output

Completion code.

pReason (PMQLONG) – input/output

Reason code qualifying *pCompCode*.

Usage notes

1. The MQ_GET_EXIT function interface described here is used for both the MQXF_GET exit function and the MQXF_DATA_CONV_ON_GET exit function. However, separate entry points are defined for these two exit functions, so to intercept *both* the MQXEP call must be used twice – once with function identifier MQXF_GET, and again with MQXF_DATA_CONV_ON_GET. Because the MQ_GET_EXIT interface is the same for MQXF_GET and MQXF_DATA_CONV_ON_GET, a single exit function can be used for both; the

Function field in the MQAXP structure indicates which exit function has been invoked. Alternatively, the MQXEP call can be used to register different exit functions for the two cases.

2. There is no MQXR_AFTER exit function for MQXF_DATA_CONV_ON_GET; the MQXR_AFTER exit function for MQXF_GET provides the required capability for exit processing after data conversion.

C invocation

```
MQ_GET_EXIT (&ExitParms, &ExitContext, &Hconn, &Hobj,
             &pMsgDesc, &pGetMsgOpts, &BufferLength,
             &pBuffer, &pDataLength, &CompCode, &Reason);
```

The parameters passed to the exit are declared as follows:

```
PMQAXP    pExitParms;    /* Exit parameter structure */
PMQAXC    pExitContext;  /* Exit context structure */
PMQHCONN  pHconn;       /* Connection handle */
PMQHOBJ   pHobj;        /* Object handle */
PPMQMD    ppMsgDesc;    /* Message descriptor */
PPMQGMO   ppGetMsgOpts; /* Options that control the action of MQGET */
PMQLONG   pBufferLength; /* Length in bytes of the pBuffer area */
PPMVOID   ppBuffer;     /* Area to contain the message data */
PPMQLONG  ppDataLength; /* Length of the message */
PMQLONG   pCompCode;    /* Completion code */
PMQLONG   pReason;      /* Reason code qualifying CompCode */
```

MQ_INIT_EXIT – Initialize exit environment

Exit providers can supply an MQ_INIT_EXIT function to perform connection-level initialization.

Syntax

```
MQ_INIT_EXIT (pExitParms, pExitContext, pCompCode, pReason)
```

Parameters

The MQ_INIT_EXIT call has the following parameters.

pExitParms (PMQAXP) – input/output

Exit parameter structure.

pExitContext (PMQAXC) – input/output

Exit context structure.

pCompCode (PMQLONG) – input/output

Completion code.

pReason (PMQLONG) – input/output

Reason code qualifying *pCompCode*.

Usage notes

1. The MQ_INIT_EXIT function can issue the MQXEP call to register the addresses of the exit functions for the particular MQ calls to be intercepted. It is not necessary to intercept all MQ calls, or to intercept both MQXR_BEFORE and MQXR_AFTER calls. For example, an exit suite could choose to intercept only the MQXR_BEFORE call of MQPUT.
2. Storage that is to be used by exit functions in the exit suite can be acquired by the MQ_INIT_EXIT function. Alternatively, exit functions can acquire storage when they are invoked, as and when needed. However, all storage should be freed before the exit suite is terminated; the MQ_TERM_EXIT function can free the storage, or an exit function invoked earlier.
3. If MQ_INIT_EXIT returns MQXCC_FAILED in the *ExitResponse* field of MQAXP, or fails in some other way, the MQCONN or MQCONNX call that caused MQ_INIT_EXIT to be invoked also fails, with the *CompCode* and *Reason* parameters set to appropriate values.
4. An MQ_INIT_EXIT function cannot issue MQ calls other than MQXEP.

C invocation

```
MQ_INIT_EXIT (&ExitParms, &ExitContext, &CompCode,
              &Reason);
```

The parameters passed to the exit are declared as follows:

```
PMQAXP  pExitParms;    /* Exit parameter structure */
PMQAXC  pExitContext; /* Exit context structure */
PMQLONG pCompCode;    /* Completion code */
PMQLONG pReason;      /* Reason code qualifying CompCode */
```

MQ_INQ_EXIT – Inquire object attributes

Exit providers can supply an MQ_INQ_EXIT function to intercept the MQINQ call.

Syntax

```
MQ_INQ_EXIT (pExitParms, pExitContext, pHconn, pHobj, pSelectorCount,
            ppSelectors, pIntAttrCount, ppIntAttrs, pCharAttrLength, ppCharAttrs,
            pCompCode, pReason)
```

Parameters

The MQ_INQ_EXIT call has the following parameters.

pExitParms (PMQAXP) – input/output

Exit parameter structure.

pExitContext (PMQAXC) – input/output

Exit context structure.

pHconn (PMQHCONN) – input/output

Connection handle.

pHobj (PMQHOBJ) – input/output

Object handle.

pSelectorCount (PMQLONG) – input/output

Count of selectors.

ppSelectors (PPMQLONG) – input/output

Array of attribute selectors.

pIntAttrCount (PMQLONG) – input/output

Count of integer attributes.

ppIntAttrs (PPMQLONG) – input/output

Array of integer attributes.

pCharAttrLength (PMQLONG) – input/output

Length of character attributes buffer.

ppCharAttrs (PPMQCHAR) – input/output

Character attributes.

pCompCode (PMQLONG) – input/output

Completion code.

pReason (PMQLONG) – input/output

Reason code qualifying *pCompCode*.

C invocation

```
MQ_INQ_EXIT (&ExitParms, &ExitContext, &Hconn, &Hobj,
            &SelectorCount, &ppSelectors, &IntAttrCount,
            &ppIntAttrs, &CharAttrLength, &ppCharAttrs,
            &CompCode, &Reason);
```

The parameters passed to the exit are declared as follows:

MQ_INQ_EXIT

```
PMQAXP    pExitParms;        /* Exit parameter structure */
PMQAXC    pExitContext;     /* Exit context structure */
PMQHCONN  pHconn;          /* Connection handle */
PMQHOBJ   pHobj;           /* Object handle */
PMQLONG   pSelectorCount;   /* Count of selectors */
PPMQLONG  ppSelectors;      /* Array of attribute selectors */
PMQLONG   pIntAttrCount;    /* Count of integer attributes */
PPMQLONG  ppIntAttrs;       /* Array of integer attributes */
PMQLONG   pCharAttrLength;  /* Length of character attributes buffer */
PMQCHAR   ppCharAttrs;      /* Character attributes */
PMQLONG   pCompCode;        /* Completion code */
PMQLONG   pReason;          /* Reason code qualifying CompCode */
```

MQ_OPEN_EXIT – Open object

Exit providers can supply an MQ_OPEN_EXIT function to intercept the MQOPEN call.

Syntax

```
MQ_OPEN_EXIT (pExitParms, pExitContext, pHconn, ppObjDesc,
             pOptions, ppHobj, pCompCode, pReason)
```

Parameters

The MQ_OPEN_EXIT call has the following parameters.

pExitParms (PMQAXP) – input/output

Exit parameter structure.

pExitContext (PMQAXC) – input/output

Exit context structure.

pHconn (PMQHCONN) – input/output

Connection handle.

ppObjDesc (PPMQOD) – input/output

Object descriptor.

pOptions (PMQLONG) – input/output

Options that control the action of MQ_OPEN_EXIT.

ppHobj (PPMQHOBJ) – input/output

Object handle.

pCompCode (PMQLONG) – input/output

Completion code.

pReason (PMQLONG) – input/output

Reason code qualifying *pCompCode*.

C invocation

```
MQ_OPEN_EXIT (&ExitParms, &ExitContext, &Hconn,
             &pObjDesc, &Options, &pHobj, &CompCode,
             &Reason);
```

The parameters passed to the exit are declared as follows:

```
PMQAXP    pExitParms;    /* Exit parameter structure */
PMQAXC    pExitContext;  /* Exit context structure */
PMQHCONN  pHconn;       /* Connection handle */
PPMQOD    ppObjDesc;    /* Object descriptor */
PMQLONG   pOptions;     /* Options that control the action of
                        MQ_OPEN_EXIT */
PPMQHOBJ  ppHobj;       /* Object handle */
PMQLONG   pCompCode;    /* Completion code */
PMQLONG   pReason;      /* Reason code qualifying CompCode */
```

MQ_PUT_EXIT – Put message

Exit providers can supply an MQ_PUT_EXIT function to intercept the MQPUT call.

Syntax

```
MQ_PUT_EXIT (pExitParms, pExitContext, pHconn, pHobj, ppMsgDesc,
            ppPutMsgOpts, pBufferLength, ppBuffer, pCompCode, pReason)
```

Parameters

The MQ_PUT_EXIT call has the following parameters.

pExitParms (PMQAXP) – input/output

Exit parameter structure.

pExitContext (PMQAXC) – input/output

Exit context structure.

pHconn (PMQHCONN) – input/output

Connection handle.

pHobj (PMQHOBJ) – input/output

Object handle.

ppMsgDesc (PPMQMD) – input/output

Message descriptor.

ppPutMsgOpts (PPMQPMO) – input/output

Options that control the action of MQPUT.

pBufferLength (PMQLONG) – input/output

Length of the message in *pBuffer*.

ppBuffer (PPMQVOID) – input/output

Message data.

pCompCode (PMQLONG) – input/output

Completion code.

pReason (PMQLONG) – input/output

Reason code qualifying *pCompCode*.

Usage notes

- Report messages generated by the queue manager skip the normal call processing. As a result, such messages cannot be intercepted by the MQ_PUT_EXIT function or the MQPUT1 function. However, report messages generated by the message channel agent are processed normally, and hence can be intercepted by the MQ_PUT_EXIT function or the MQ_PUT1_EXIT function. To be sure to intercepting all of the report messages generated by the MCA, both MQ_PUT_EXIT and MQ_PUT1_EXIT should be used.

C invocation

```
MQ_PUT_EXIT (&ExitParms, &ExitContext, &Hconn, &Hobj,
            &pMsgDesc, &pPutMsgOpts, &BufferLength,
            &pBuffer, &CompCode, &Reason);
```

The parameters passed to the exit are declared as follows:

```
PMQAXP  pExitParms;      /* Exit parameter structure */
PMQAXC  pExitContext;   /* Exit context structure */
PMQHCONN pHConn;       /* Connection handle */
PMQHOBJ  pHObj;        /* Object handle */
PPMQMD  ppMsgDesc;     /* Message descriptor */
PPMQPMO  ppPutMsgOpts; /* Options that control the action of MQPUT */
PMQLONG  pBufferLength; /* Length of the message in pBuffer */
PPMQVOID ppBuffer;     /* Message data */
PMQLONG  pCompCode;    /* Completion code */
PMQLONG  pReason;      /* Reason code qualifying CompCode */
```

MQ_PUT1_EXIT – Put one message

Exit providers can supply an MQ_PUT1_EXIT function to intercept the MQPUT1 call.

Syntax

```
MQ_PUT1_EXIT (pExitParms, pExitContext, pHconn, ppObjDesc,
             ppMsgDesc, ppPutMsgOpts, pBufferLength, pBuffer, pCompCode, pReason)
```

Parameters

The MQ_PUT1_EXIT call has the following parameters.

pExitParms (PMQAXP) – input/output

Exit parameter structure.

pExitContext (PMQAXC) – input/output

Exit context structure.

pHconn (PMQHCONN) – input/output

Connection handle.

ppObjDesc (PPMQOD) – input/output

Object descriptor.

ppMsgDesc (PPMQMD) – input/output

Message descriptor.

ppPutMsgOpts (PPMQPMO) – input/output

Options that control the action of MQPUT1.

pBufferLength (PMQLONG) – input/output

Length of the message in *pBuffer*.

ppBuffer (PPMQVOID) – input/output

Message data.

pCompCode (PMQLONG) – input/output

Completion code.

pReason (PMQLONG) – input/output

Reason code qualifying *pCompCode*.

C invocation

```
MQ_PUT1_EXIT (&ExitParms, &ExitContext, &Hconn,
             &pObjDesc, &pMsgDesc, &pPutMsgOpts,
             &BufferLength, &pBuffer, &CompCode,
             &Reason);
```

The parameters passed to the exit are declared as follows:

```
PMQAXP    pExitParms;    /* Exit parameter structure */
PMQAXC    pExitContext;  /* Exit context structure */
PMQHCONN  pHconn;       /* Connection handle */
PPMQOD    ppObjDesc;    /* Object descriptor */
PPMQMD    ppMsgDesc;    /* Message descriptor */
PPMQPMO   ppPutMsgOpts; /* Options that control the action of MQPUT1 */
PMQLONG   pBufferLength; /* Length of the message in pBuffer */
```

```
PPMVOID ppBuffer;      /* Message data */  
PMQLONG pCompCode;    /* Completion code */  
PMQLONG pReason;      /* Reason code qualifying CompCode */
```

MQ_SET_EXIT – Set object attributes

Exit providers can supply an MQ_SET_EXIT function to intercept the MQSET call.

Syntax

```
MQ_SET_EXIT (pExitParms, pExitContext, pHconn, pHobj, pSelectorCount,
            ppSelectors, pIntAttrCount, ppIntAttrs, pCharAttrLength, ppCharAttrs,
            pCompCode, pReason)
```

Parameters

The MQ_SET_EXIT call has the following parameters.

pExitParms (PMQAXP) – input/output

Exit parameter structure.

pExitContext (PMQAXC) – input/output

Exit context structure.

pHconn (PMQHCONN) – input/output

Connection handle.

pHobj (PMQHOBJ) – input/output

Object handle.

pSelectorCount (PMQLONG) – input/output

Count of selectors.

ppSelectors (PPMQLONG) – input/output

Array of attribute selectors.

pIntAttrCount (PMQLONG) – input/output

Count of integer attributes.

ppIntAttrs (PPMQLONG) – input/output

Array of integer attributes.

pCharAttrLength (PMQLONG) – input/output

Length of character attributes buffer.

ppCharAttrs (PPMQCHAR) – input/output

Character attributes.

pCompCode (PMQLONG) – input/output

Completion code.

pReason (PMQLONG) – input/output

Reason code qualifying *pCompCode*.

C invocation

```
MQ_SET_EXIT (&ExitParms, &ExitContext, &Hconn, &Hobj,
            &SelectorCount, &ppSelectors, &IntAttrCount,
            &pIntAttrs, &CharAttrLength, &ppCharAttrs,
            &CompCode, &Reason);
```

The parameters passed to the exit are declared as follows:

```
PMQAXP    pExitParms;      /* Exit parameter structure */
PMQAXC    pExitContext;   /* Exit context structure */
PMQHCONN  pHconn;        /* Connection handle */
PMQHOBJ   pHobj;         /* Object handle */
PMQLONG   pSelectorCount; /* Count of selectors */
PPMQLONG  ppSelectors;    /* Array of attribute selectors */
PMQLONG   pIntAttrCount;  /* Count of integer attributes */
PPMQLONG  ppIntAttrs;     /* Array of integer attributes */
PMQLONG   pCharAttrLength; /* Length of character attributes buffer */
PPMQCHAR  ppCharAttrs;    /* Character attributes */
PMQLONG   pCompCode;      /* Completion code */
PMQLONG   pReason;        /* Reason code qualifying CompCode */
```

MQ_TERM_EXIT – Terminate exit environment

Exit providers can supply an MQ_INIT_EXIT function to perform connection-level termination.

Syntax

```
MQ_TERM_EXIT (pExitParms, pExitContext, pCompCode, pReason)
```

Parameters

The MQ_TERM_EXIT call has the following parameters.

pExitParms (PMQAXP) – input/output

Exit parameter structure.

pExitContext (PMQAXC) – input/output

Exit context structure.

pCompCode (PMQLONG) – input/output

Completion code.

pReason (PMQLONG) – input/output

Reason code qualifying *pCompCode*.

Usage notes

1. The MQ_TERM_EXIT function is optional. It is not necessary for an exit suite to register a termination exit if there is no termination processing to be done. If functions belonging to the exit suite acquire resources during the connection, an MQ_TERM_EXIT function is a convenient point at which to free those resources, for example, freeing storage obtained dynamically.
2. If an MQ_TERM_EXIT function is registered when the MQDISC call is issued, the exit function is invoked after all of the MQDISC exit functions have been invoked.
3. If MQ_TERM_EXIT returns MQXCC_FAILED in the *ExitResponse* field of MQAXP, or fails in some other way, the MQDISC call that caused MQ_TERM_EXIT to be invoked also fails, with the *CompCode* and *Reason* parameters set to appropriate values.

C invocation

```
MQ_TERM_EXIT (&ExitParms, &ExitContext, &CompCode,
              &Reason);
```

The parameters passed to the exit are declared as follows:

```
PMQAXP  pExitParms;    /* Exit parameter structure */
PMQAXC  pExitContext;  /* Exit context structure */
PMQLONG pCompCode;    /* Completion code */
PMQLONG pReason;      /* Reason code qualifying CompCode */
```

Part 8. Appendixes

Appendix A. System and default objects

When you create a queue manager using the **crtmqm** control command, the system objects and the default objects are created automatically.

- The system objects are those WebSphere MQ objects needed to operate a queue manager or channel.
- The default objects define all the attributes of an object. When you create an object, such as a local queue, any attributes that you do not specify explicitly are inherited from the default object.

The following tables list the system and default objects created by **crtmqm**:

- Table 35 lists the system and default queue objects.
- Table 36 on page 550 lists the system and default channel objects.
- Table 37 on page 550 lists the system and default authentication information objects.
- Table 38 on page 550 lists the system and default listener objects.
- Table 39 on page 551 lists the system and default namelist objects.
- Table 40 on page 551 lists the system and default process objects.
- Table 41 on page 551 lists the system and default service objects.

Table 35. System and default objects: 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.

Default objects

Table 35. System and default objects: queues (continued)

Object name	Description
SYSTEM.DEFAULT.ALIAS.QUEUE	Default alias queue.
SYSTEM.DEFAULT.INITIATION.QUEUE	Default initiation queue.
SYSTEM.DEFAULT.LOCAL.QUEUE	Default local queue.
SYSTEM.DEFAULT.MODEL.QUEUE	Default model queue.
SYSTEM.DEFAULT.REMOTE.QUEUE	Default remote queue.
SYSTEM.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 command reply-to queue. This is 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 36. System and default objects: channels

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

Table 37. System and default objects: authentication information objects

Object name	Description
SYSTEM.DEFAULT.AUTHINFO.CRLLDAP	Default authentication information object.

Table 38. System and default objects: listeners

Object name	Description
SYSTEM.DEFAULT.LISTENER.TCP	Default TCP listener.
SYSTEM.DEFAULT.LISTENER.LU62 (Windows only)	Default LU62 listener.
SYSTEM.DEFAULT.LISTENER.NETBIOS (Windows only)	Default NETBIOS listener.

Table 38. System and default objects: listeners (continued)

Object name	Description
SYSTEM.DEFAULT.LISTENER.SPX (Windows only)	Default SPX listener.

Table 39. System and default objects: namelists

Object name	Description
SYSTEM.DEFAULT.NAMELIST	Default namelist.

Table 40. System and default objects: processes

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

Table 41. System and default objects: services

Object name	Description
SYSTEM.DEFAULT.SERVICE	Default service.
SYSTEM.BROKER	Publish/subscribe broker

Windows default configuration objects

On Windows systems, you set up a default configuration using either the WebSphere MQ First Steps application or the WebSphere MQ Postcard application.

Note: You cannot set up a default configuration if other queue managers exist on your computer.

Many of the names used for the Windows default configuration objects involve the use of a short TCP/IP name. This is the TCP/IP name of the computer, without the domain part; for example the short TCP/IP name for the computer `mycomputer.hursley.ibm.com` is `mycomputer`. In all cases, where this name has to be truncated, if the last character is a period (`.`), it is removed.

Any characters within the short TCP/IP name that are not valid for WebSphere MQ object names (for example, hyphens) are replaced by an underscore character.

Valid characters for WebSphere MQ object names are: a to z, A to Z, 0 to 9, and the four special characters `/`, `%`, `.`, and `_`.

The cluster name for the Windows default configuration is `DEFAULT_CLUSTER`.

If the queue manager is not a repository queue manager, the objects listed in Table 42 are created.

Windows default configuration

Table 42. Objects created by the Windows default configuration application

Object	Name
Queue manager	<p>The short TCP/IP name prefixed with the characters QM_. The maximum length of the queue manager name is 48 characters. Names exceeding this limit are truncated at 48 characters. If the last character of the name is a period (.), this is replaced by a space ().</p> <p>The queue manager has a command server, a channel listener, and channel initiator associated with it. The channel listener listens on the standard WebSphere MQ port, port number 1414. Any other queue managers created on this machine must not use port 1414 while the default configuration queue manager still exists.</p>
Generic cluster receiver channel	<p>The short TCP/IP name prefixed with the characters TO_QM_. The maximum length of the generic cluster receiver name is 20 characters. Names exceeding this limit are truncated at 20 characters. If the last character of the name is a period (.), this is replaced by a space ().</p>
Cluster sender channel	<p>The cluster sender channel is initially created with the name TO_+QMNAME+. Once WebSphere MQ has established a connection to the repository queue manger for the default configuration cluster, this name is replaced with the name of the repository queue manager for the default configuration cluster, prefixed with the characters TO_. The maximum length of the cluster sender channel name is 20 characters. Names exceeding this limit are truncated at 20 characters. If the last character of the name is a period (.), this is replaced by a space ().</p>
Local message queue	<p>The local message queue is called default.</p>
Local message queue for use by the WebSphere MQ Postcard application	<p>The local message queue for use by the WebSphere MQ Postcard application is called postcard.</p>
Server connection channel	<p>The server connection channel allows clients to connect to the queue manager. Its name is the short TCP/IP name, prefixed with the characters S_. The maximum length of the server connection channel name is 20 characters. Names exceeding this limit are truncated at 20 characters. If the last character of the name is a period (.), this is replaced by a space ().</p>

If the queue manager is a repository queue manager, the default configuration is similar to that described in Table 42, but with the following differences:

- The queue manager is defined as a repository queue manager for the default configuration cluster.
- There is no cluster-sender channel defined.
- A local cluster queue that is the short TCP/IP name prefixed with the characters clq_default_ is created. The maximum length of this name is 48 characters. Names exceeding this length are truncated at 48 characters.

If you request remote administration facilities, the server connection channel, SYSTEM.ADMIN.SVRCONN is also created.

Appendix B. Directory structure (Windows systems)

Table 43 shows the directories found under the root `c:\Program Files\IBM\WebSphere MQ\`. If you have installed WebSphere MQ for Windows under a different directory, the root is modified appropriately.

Table 43. *WebSphere MQ for Windows directory structure*

\bin	Contains binary files (commands and DDLs).
\config	Contains configuration information.
\conv	Contains files for data conversion in folder <code>\table</code> .
\errors	<p>Contains the system error log files: AMQERR01.LOG AMQERR02.LOG AMQERR03.LOG</p> <p>These files contain information related errors that are not associated with a particular queue manager. AMQERR01.LOG contains the most recent error information.</p> <p>This folder also holds any FFST files that are produced.</p>
\exits	Contains channel exit programs.
\licenses	Contains a folder for each national language. Each folder contains license information.
\log	<p>Contains a folder for each queue manager. The following subdirectories and files will exist for each queue manager after you have been using that queue manager for some time.</p> <p>AMQHLCTL.LFH Log control file.</p> <p>Active This directory contains the log files numbered S0000000.LOG, S0000001.LOG, S0000002.LOG, and so on.</p>
\qmgrs	Contains a folder for each queue manager; the contents of these folders are described in Table 44 on page 554. Also contains the folder <code>\@SYSTEM\errors</code> ,
\tivoli	Contains the signature file used by Tivoli®.
\tools	Contains all the WebSphere MQ sample programs. These are described in <i>WebSphere MQ for Windows, V6.0 Quick Beginnings</i> .
\trace	Contains all trace files.
\uninst	Contains files necessary to uninstall WebSphere MQ.

Table 44 on page 554 shows the directory structure for each queue manager in the `c:\Program Files\IBM\WebSphere MQ\qmgrs\` folder. The queue manager might have been transformed as described in “Understanding WebSphere MQ file names” on page 20.

Directory structure (Windows systems)

Table 44. Content of a \queue-manager-name\ folder for WebSphere MQ for Windows

amqalchk.fil	Contains a checkpoint file containing information about the last checkpoint.
\authinfo	Contains a file for each authentication information object.
\channel	Contains a file for each channel object.
\clntconn	Contains a file for each client connection channel object.
\errors	Contains error log files associated with the queue manager: AMQERR01.LOG AMQERR02.LOG AMQERR03.LOG AMQERR01.LOG contains the most recent error information.
\listener	Contains a file for each listener object.
\namelist	Contains a file for each WebSphere MQ namelist.
\Plugcomp	Directory reserved for use by WebSphere MQ installable services.
\Procdef	Contains a file for each WebSphere MQ process definition. Where possible, the file name matches the associated process definition name, but some characters have to be altered. There might be a directory called @MANGLED here containing process definitions with transformed or mangled names.
\Qmanager	Contains the following files: Qmanager The queue manager object. QMOMOBJCAT The object catalogue containing the list of all WebSphere MQ objects, used internally. Note: If you are using a FAT system, this name is transformed and a subdirectory created containing the file with its name transformed. QAADMIN File used internally for controlling authorizations.
\Queues	Each queue has a directory here containing a single file called Q. Where possible, the directory name matches the associated queue name but some characters have to be altered. There might be a directory called @MANGLED here containing queues with transformed or mangled names.
\services	Contains a file for each service object.
\ssl	Contains SSL certificate stores.
\Startprm	Contains temporary files used internally.

Appendix C. Directory structure (UNIX systems)

Figure 36 on page 556 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. For information about the location of the product files, see one of the following:

- *WebSphere MQ for AIX, V6.0 Quick Beginnings*
- *WebSphere MQ for HP-UX, V6.0 Quick Beginnings*
- *WebSphere MQ for Solaris, V6.0 Quick Beginnings*
- *WebSphere MQ for Linux, V6.0 Quick Beginnings*

In Figure 36 on page 556, 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.

Directory structure (UNIX systems)

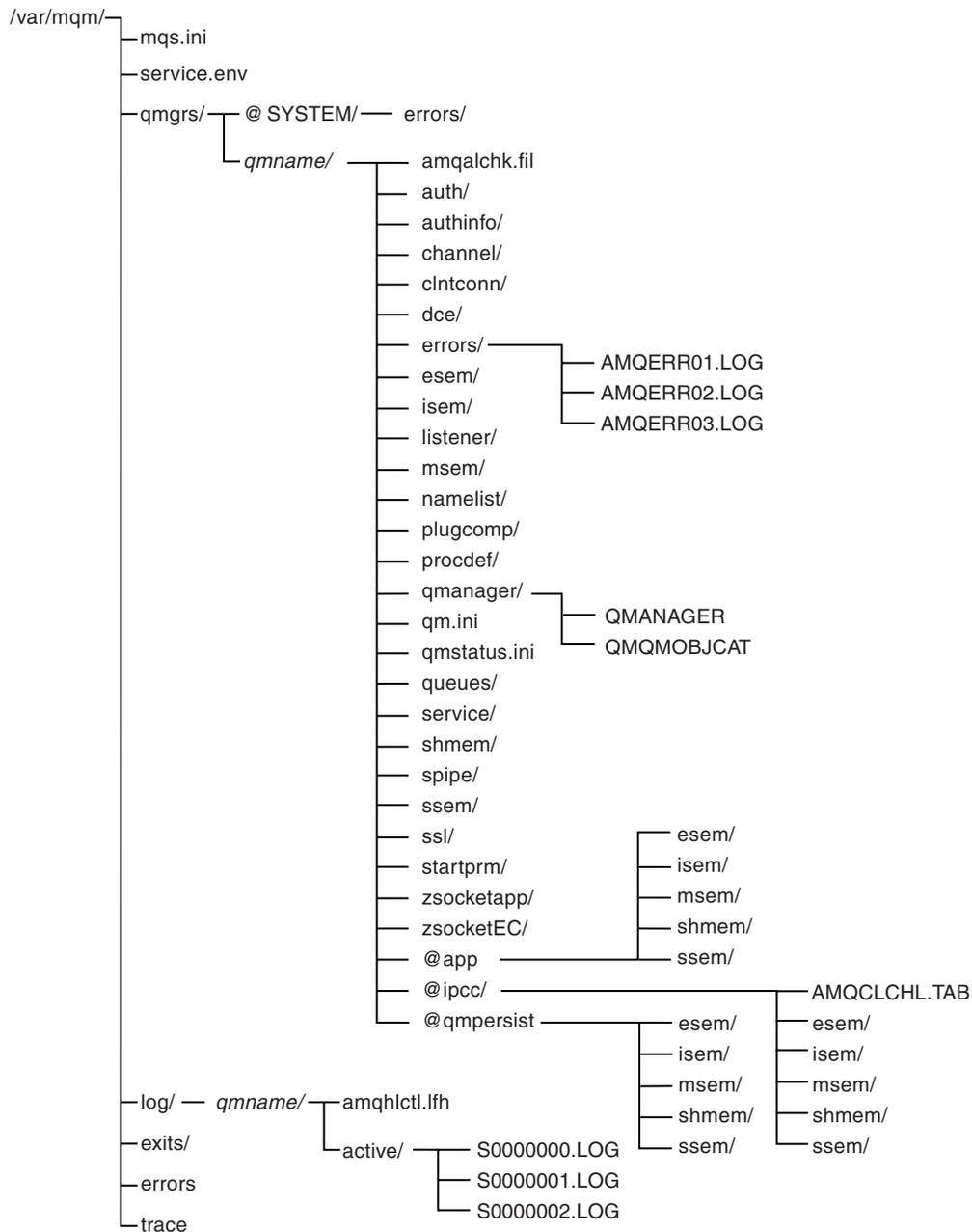


Figure 36. Default directory structure (UNIX systems) after a queue manager has been started

By default, the following directories and files are located in the directory `/var/mqm/qmgrs/qmname/` (where `qmname` is the name of the queue manager).

Directory structure (UNIX systems)

Table 45. Default content of a `/var/mqm/qmgrs/qmname/` directory on UNIX systems

amqalchk.fil	Checkpoint file containing information about the last checkpoint.
auth/	Contained subdirectories and files associated with authority in WebSphere MQ prior to Version 6.0.
authinfo/	Each WebSphere MQ authentication information definition is associated with a file in this directory. The file name matches the authentication information definition name—subject to certain restrictions; see “Understanding WebSphere MQ file names” on page 20.
channel/	Each WebSphere MQ channel definition is associated with a file in this directory. The file name matches the channel definition name—subject to certain restrictions; see “Understanding WebSphere MQ file names” on page 20.
clntconn/	Each WebSphere MQ client connection channel definition is associated with a file in this directory. The file name matches the client connection channel definition name—subject to certain restrictions; see “Understanding WebSphere MQ file names” on page 20.
dce/	Used for DCE support prior to WebSphere MQ Version 6.0.
errors/	Directory containing FFSTs, client application errors, and 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.
listener/	Each WebSphere MQ listener definition is associated with a file in this directory. The file name matches the listener definition name—subject to certain restrictions; see “Understanding WebSphere MQ file names” on page 20.
msem/	Directory containing files used internally.
namelist/	Each WebSphere MQ namelist definition is associated with a file in this directory. The file name matches the namelist definition name—subject to certain restrictions; see “Understanding WebSphere MQ file names” on page 20.
plugcomp/	Empty directory reserved for use by installable services.
procdef/	Each WebSphere MQ process definition is associated with a file in this directory. The file name matches the process definition name—subject to certain restrictions; see “Understanding WebSphere MQ file names” on page 20.
qmanager/	QMANAGER The queue manager object. QMOMBJCAT The object catalog containing the list of all WebSphere MQ objects; used internally.
qm.ini	Queue manager configuration file.
queues/	Each queue has a directory in here containing a single file called <code>q</code> . The file name matches the queue name, subject to certain restrictions; see “Understanding WebSphere MQ file names” on page 20.

Directory structure (UNIX systems)

Table 45. Default content of a `/var/mqm/qmgrs/qmname/` directory on UNIX systems (continued)

services/	Each WebSphere MQ service definition is associated with a file in this directory. The file name matches the service definition name—subject to certain restrictions; see “Understanding WebSphere MQ file names” on page 20.
shmem/	Directory containing files used internally.
spipe/	Used internally by channel processes.
ssem/	Directory containing files used internally.
ssl/	Directory for SSL key database files.
startprm/	Directory containing temporary files used internally.
zsocketapp/	Used internally for isolated bindings.
zsocketEC/	Used internally for isolated bindings.
@ipcc/	AMQCLCHL.TAB Client channel table file. esem/ Directory containing files used internally. isem/ Directory containing files used internally. msem/ Directory containing files used internally. shmem/ Directory containing files used internally. ssem/ Directory containing files used internally.
@qmpersist	esem/ Directory containing files used internally. isem/ Directory containing files used internally. msem/ Directory containing files used internally. shmem/ Directory containing files used internally. ssem/ Directory containing files used internally.
@qmpersist	esem/ Directory containing files used internally. isem/ Directory containing files used internally. msem/ Directory containing files used internally. shmem/ Directory containing files used internally. ssem/ Directory containing files used internally.
@app	esem/ Directory containing files used internally. isem/ Directory containing files used internally. msem/ Directory containing files used internally. shmem/ Directory containing files used internally. ssem/ Directory containing files used internally.

By default, the following directories and files are found in `/var/mqm/log/qmname/` (where *qmname* is the name of the queue manager).

The following subdirectories and files exist 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 S0000000.LOG, S0000001.LOG, S0000002.LOG, and so on.

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 Windows

To stop a queue manager running under WebSphere MQ for Windows:

1. List the names (IDs) of the processes currently running using the Windows Process Viewer (PView)
2. Stop the processes using PView in the following order (if they are running):

AMQZMUC0	Critical process manager
AMQZXMA0	Execution controller
AMQFUMA	OAM process
AMQZLAA0	LQM agents
AMQZLSA0	LQM agents
AMQZMUR0	Restartable process manager
AMQRMPPA	Process pooling process
AMQRRMFA	The repository process (for clusters)
AMQZDMAA	Deferred message processor
AMQPCSEA	The command server
AMQXSSVN	Shared memory servers
AMQZTRCN	Trace

3. Stop the WebSphere MQ service from Services on the Windows Control Panel.
4. If you have tried all methods and the queue manager has not stopped, reboot your system.

Stopping queue managers in WebSphere MQ for UNIX systems

To stop a queue manager running under WebSphere MQ for UNIX systems:

1. Find the process IDs of the queue manager programs that are still running using the **ps** command. For example, if the queue manager is called QMNAME, use the following command:

```
ps -ef | grep QMNAME
```

2. End any queue manager processes that are still running. Use the **kill** command, specifying the process IDs discovered using the **ps** command.

End the processes in the following order:

amqzmuc0	Critical process manager
amqzxma0	Execution controller

Stopping queue managers

amqfuma	OAM process
amqzlaa0	LQM agents
amqzlsa0	LQM agents
amqzmur0	Restartable process manager
amqrmppa	Process pooling process
amqrrmfa	The repository process (for clusters)
amqzdmaa	Deferred message processor
amqpcsea	The command server

Note: Processes that fail to stop can be ended using **kill -9**.

If you stop the queue manager manually, FFSTs might be taken, and FDC files placed in `/var/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. If, for some reason, this command fails to delete the queue manager, use the manual processes described here.

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*<QMgrName>*\QueueManager
5. Change the startup value to zero. (1 means automatic and 0 means manual.)
6. Close the Registry Editor.
7. Run `amqdain 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.

Stopping queue managers

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. File Transfer Application

This chapter explains how to use the File Transfer Application.

The chapter includes:

- “Introduction to the File Transfer Application”
- “Installing and configuring the File Transfer Application” on page 564
- “Using the File Transfer Application” on page 570

Introduction to the File Transfer Application

The File Transfer Application allows you to send and receive files in the form of WebSphere MQ messages. You can use the File Transfer Application to send and receive any type of file in any format, for example: ASCII Linux format (with line feed characters), ASCII file Windows format (with carriage return/line feed characters), binary (for example, image files, wordprocessor files, spreadsheet files, or zip files), also reports, letters, memos and charts.

The File Transfer Application is available on both WebSphere MQ for Windows, and WebSphere MQ for Linux (x86 platform) servers, and clients.

The File Transfer Application includes a graphical user interface (GUI), and a command-line interface. Both interfaces invoke the same underlying WebSphere MQ functionality and can be used to transfer the same types of file.

There are three principal users of the File Transfer Application:

System administrators

Set up the File Transfer Application for end-users, by defining queues as sources and destinations in the File Transfer Application GUI.

Non-experienced end-users

These are non-expert IT users working within the business, for example car showroom managers. Users in this group want to send and receive files such as daily sales figures and stock reports, using the File Transfer Application GUI.

Experienced end-users

These are expert IT users working within the business, who have an understanding of WebSphere MQ. Users in this group want to send and receive files from the command-line.

Advantages

- *Files of any type can be transferred.* Because the File Transfer Application does not distinguish between files of different types, you can send and receive files in any format (for example, spreadsheets, memos, letters). You can even send and receive image and sound files.
- *File transfer is technology independent.* Files can be transferred between dissimilar operating platforms (for example Windows, UNIX), using TCP/IP.
- *Transferred files cannot be accidentally duplicated.* Files are sent once-and-once-only to a specified destination.
- *Files are transferred securely.* High-level data security and integrity is provided if SSL (secure sockets layer) encrypted message channels are used.

File Transfer Application

- *The sender and receiver run independently.* The sender and receiver do not both have to be running at the same time.
If the receiver is currently unavailable or busy, the file is held on a queue. When the receiver becomes available, the file is then automatically transferred. The persistent option assures maximum reliability (non-persistent messages are automatically deleted from the queue on the receiving machine, when the queue manager restarts).
- *The system is scalable.* An administrator can add new sources and destinations to the File Transfer Application GUI, so that they become accessible to users.
- *The File Transfer Application GUI.* The File Transfer Application provides a GUI.

Components

The File Transfer Application consists of the following main components:

Sender

This is a program that puts a file stored in the local file system onto a queue, as one or more WebSphere MQ messages.

Receiver

This is a program that receives files and stores them in a local file system.

File Transfer Application GUI

The GUI allows non-experienced users to send files, receive files, and create a list of sent/received files in an intuitive way. Users of the GUI need no knowledge of how the underlying WebSphere MQ technology works.

Available when installed from a WebSphere MQ server CD.

A command line interface

This provides a way for experienced users to send and receive files by issuing commands from the command line. Additional functionality is available with the File Transfer Application GUI. Users of the command line interface need to have an understanding of how WebSphere MQ works.

Installing and configuring the File Transfer Application

The File Transfer Application is used to transfer files between remote machines. The File Transfer Application can be installed on a WebSphere MQ server or on a WebSphere MQ client. For a file to be transferred between two remote machines, the File Transfer Application must be installed on both the sender machine and the receiver machine. Having installed the File Transfer Application additional setup and configuration tasks must be performed.

Installing the File Transfer Application on a WebSphere MQ server

A WebSphere MQ installation server CD must be used to install the File Transfer Application on a WebSphere MQ server. To install the File Transfer Application on a WebSphere MQ server, do one of the following:

- Install the File Transfer Application during an initial installation of a WebSphere MQ server.
- Add the File Transfer Application by modifying a WebSphere MQ server installation after the initial installation has taken place.

During the initial installation

To install the File Transfer Application on a WebSphere MQ server during the initial installation of WebSphere MQ, do the following:

1. Follow the WebSphere MQ custom installation instructions on the appropriate platform to the point where you specify the components that will be installed:
 - For the WebSphere MQ server installation instructions on Windows, see the *WebSphere MQ for Windows, V6.0 Quick Beginnings*.
 - For the WebSphere MQ server installation instructions on Linux (x86 platform), see the *WebSphere MQ for Linux, V6.0 Quick Beginnings*.
2. Ensure that **Server File Transfer** is selected.
3. Follow the WebSphere MQ server installation instructions to completion.

The File Transfer Application is now installed.

Modifying the installation

To install the File Transfer Application on a WebSphere MQ server by modifying the installation after the initial installation has taken place, do the following:

1. Follow the instructions for modifying an installation on the appropriate platform to the point where you specify the components that are to be added:
 - For the instructions for modifying an installation on Windows, see the *WebSphere MQ for Windows, V6.0 Quick Beginnings*.
 - For the instructions for modifying an installation on Linux (x86 platform), see the *WebSphere MQ for Linux, V6.0 Quick Beginnings*.
2. Ensure that **Server File Transfer** is selected.
3. Follow the instructions for modifying the installation to completion.

The File Transfer Application is now installed.

Installing the File Transfer Application on a WebSphere MQ client

Either a WebSphere MQ installation server CD, or a WebSphere MQ installation client CD, can be used to install the File Transfer Application on a WebSphere MQ client.

Note: If you require the File Transfer Application GUI, use a WebSphere MQ installation server CD.

To install the File Transfer Application on a WebSphere MQ client, do one of the following:

- Install the File Transfer Application during an initial installation of a WebSphere MQ client.
- Add the File Transfer Application by modifying an installation after the initial installation has taken place.

During the initial installation

To install the File Transfer Application on a WebSphere MQ client during the initial installation, do the following:

1. Follow the WebSphere MQ custom installation instructions on the appropriate platform to the point where you specify the components that will be installed:
 - For the WebSphere MQ client installation instructions on Windows, see the *WebSphere MQ for Windows, V6.0 Quick Beginnings*.

File Transfer Application

- For the WebSphere MQ client installation instructions on Linux (x86 platform), see the *WebSphere MQ for Linux, V6.0 Quick Beginnings*.
2. Ensure that **Client File Transfer** is selected.
 3. Follow the WebSphere MQ installation instructions to completion.

The File Transfer Application is now installed.

Modifying the installation

To install the File Transfer Application on a WebSphere MQ client by modifying the installation after the initial installation has taken place, do the following:

1. Follow the instructions for modifying an installation on the appropriate platform to the point where you specify the components that are to be added:
 - For the instructions for modifying an installation on Windows, see the *WebSphere MQ for Windows, V6.0 Quick Beginnings*.
 - For the instructions for modifying an installation on Linux (x86 platform), see the *WebSphere MQ for Linux, V6.0 Quick Beginnings*.
2. Ensure that **Client File Transfer** is selected.
3. Follow the instructions for modifying an installations to completion.

The File Transfer Application is now installed

Setup tasks

The setup tasks are required when the File Transfer Application is used to send files between a queue manager and a remote WebSphere MQ client, or between two remote queue managers. If files are sent between local queue managers or clients, the following setup tasks are not required.

Sending files between remote queue managers

This section shows how to setup two queue managers to allow the File Transfer Application to be used to send and receive files between them. For illustration, queue managers HEAD.OFFICE.QM, and SHOWROOM.QM are used. Figure 37 summarizes the configuration that the following instructions form.

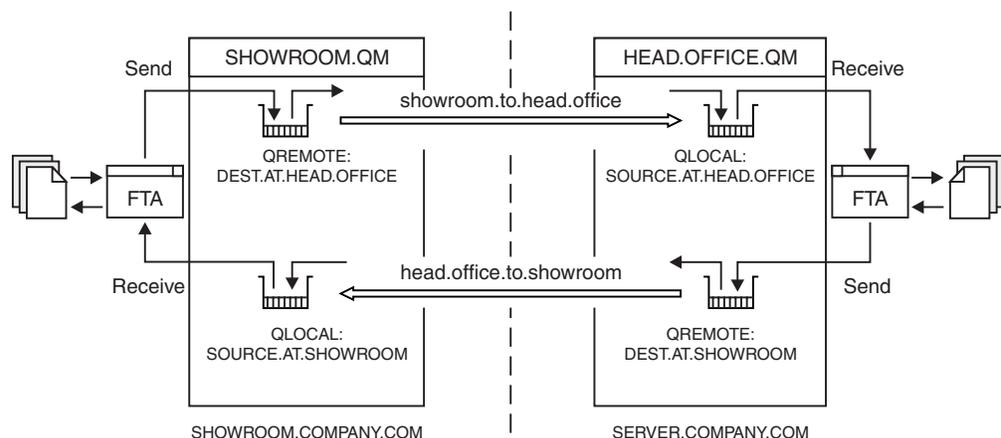


Figure 37. Using the File Transfer Application to send files between remote queue managers

These instruction outline how to setup sender and receiver channels, however other channel configurations can be used, see *WebSphere MQ Intercommunication*.

1. Issue the following commands on the queue manager HEAD.OFFICE.QM, to create the channels, listener, and the transmission queue:
 - a. Define the sender channel:

```
DEFINE CHANNEL (HEAD.OFFICE.TO.SHOWROOM) +
  CHLTYPE(SDR) +
  CONNAME (SERVER.COMPANY.COM) +
  XMITQ (SHOWROOM.QM) +
  TRPTYPE(TCP)
```

- b. Define the receiver channel:

```
DEFINE CHANNEL (SHOWROOM.TO.HEAD.OFFICE) +
  CHLTYPE(RCVR) +
  TRPTYPE(TCP)
```

- c. Define the listener:

```
DEFINE LISTENER (HEAD.OFFICE) +
  TRPTYPE (TCP) +
  PORT (1414)
```

- d. Define the transmission queue:

```
DEFINE QLOCAL (SHOWROOM.QM) +
  USAGE (XMITQ)
```

Notes:

- a. 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.
 - b. Sender and receiver channels have been used however other channel configurations are available, see the *WebSphere MQ Intercommunication* manual.
2. Issue the following commands on the queue manager SHOWROOM.QM, to create the channels, listener, and the transmission queue:

- a. Define the sender channel:

```
DEFINE CHANNEL (SHOWROOM.TO.HEAD.OFFICE) +
  CHLTYPE(SDR) +
  CONNAME (SHOWROOM.COMPANY.COM) +
  XMITQ (HEAD.OFFICE.QM) +
  TRPTYPE(TCP)
```

- b. Define the receiver channel:

```
DEFINE CHANNEL (HEAD.OFFICE.TO.SHOWROOM) +
  CHLTYPE(RCVR) +
  TRPTYPE(TCP)
```

- c. Define the listener:

```
DEFINE LISTENER (SHOWROOM) +
  TRPTYPE (TCP) +
  PORT (1414)
```

- d. Define the transmission queue:

```
DEFINE QLOCAL (HEAD.OFFICE.QM) +
  USAGE (XMITQ)
```

3. Start the listener and sender channel on the queue manager HEAD.OFFICE.QM by using the following MQSC commands:

- a. Start the listener:

```
START LISTENER (HEAD.OFFICE)
```

- b. Start the sender channel:

```
START CHANNEL (HEAD.OFFICE.TO.SHOWROOM)
```

Note: The receiver channels do not need to be started because it is the sender channels that initiate the delivery of messages.

4. Start the listener and sender channel on the queue manager SHOWROOM.QM by using the following MQSC commands:

File Transfer Application

- a. Start the listener:
`START LISTENER (SHOWROOM)`
- b. Start the sender channel:
`START CHANNEL (SHOWROOM.TO.HEAD.OFFICE)`
5. Define a remote queue definition for the destination queue and a source queue on the queue manager HEAD.OFFICE.QM, using the following MQSC commands:
 - a. Define a remote queue definition for the destination queue (the queue where files will be sent):
`DEFINE QREMOTE (DEST.AT.SHOWROOM) +
RNAME (SOURCE.AT.SHOWROOM) +
RQMNAME (SHOWROOM.QM)`
 - b. Define a source queue (the queue where files will be received):
`DEFINE QLOCAL (SOURCE.AT.HEAD.OFFICE)`

It is recommended that local queues are dedicated to the File Transfer Application.

6. Define a remote queue definition for the destination queue and a source queue on the queue manager SHOWROOM.QM, using the following MQSC commands:
 - a. Define a remote queue definition for the destination queue (the queue where files will be sent):
`DEFINE QREMOTE (DEST.AT.HEAD.OFFICE) +
RNAME (SOURCE.AT.HEAD.OFFICE) +
RQMNAME (HEAD.OFFICE.QM)`
 - b. Define a source queue (the queue where files will be received):
`DEFINE QLOCAL (SOURCE.AT.SHOWROOM)`

It is recommended that local queues are dedicated to the File Transfer Application.

7. Ensure that all the users of the File Transfer Application are members of the mqm group, or alternatively the local Administrators group on Windows.

You have now setup both queue managers for use with the File Transfer Application.

Sending files between a queue manager and a remote WebSphere MQ client

This section shows how to setup a queue manager and a remote WebSphere MQ client to allow the File Transfer Application to be used to send and receive files between them. For illustration, the queue manager HEAD.OFFICE.QM, and the WebSphere MQ client CARSHOWROOM are used. Figure 38 on page 569 summarizes the configuration that the following instructions form.

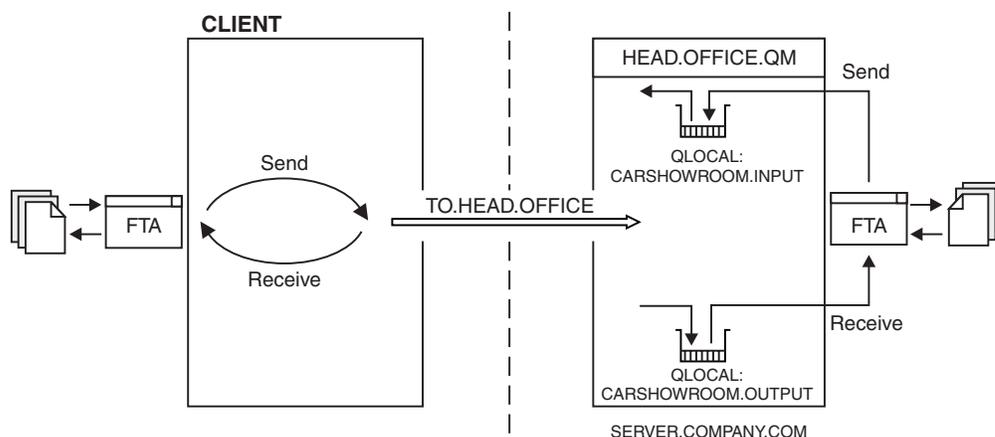


Figure 38. Using the File Transfer Application to send files between a queue manager and a remote client

To configure the queue manager HEAD.OFFICE.QM, and the remote WebSphere MQ client, do the following:

1. Define a server communication channel on the queue manager HEAD.OFFICE.QM, using the following MQSC command:

- a. Define a server communication channel:

```
DEFINE CHANNEL (TO.HEAD.OFFICE) +
  CHLTYPE(SVRCONN) +
  TRPTYPE(TCP) +
  MCAUSER (string)
```

Note: For MCAUSER (*string*), specify *string* as a user from the mqm group, or administrators group on the queue manager HEAD.OFFICE.

2. Define and start a listener on the queue manager HEAD.OFFICE.QM, using the following MQSC command:

- a. Define a listener:

```
DEFINE LISTENER (HEAD.OFFICE) +
  TRPTYPE (TCP) +
  PORT (1414)
```

- b. Start the listener:

```
START LISTENER (HEAD.OFFICE)
```

3. Define a source queue, and a destination queue on the queue manager HEAD.OFFICE.QM to be used by the WebSphere MQ client, using the following MQSC commands:

- a. Define a destination queue (the queue where the WebSphere MQ client will send files):

```
DEFINE QLOCAL (CARSHOWROOM.OUTPUT)
```

- b. Define a source queue (the queue from which the WebSphere MQ client will receive files):

```
DEFINE QLOCAL (CARSHOWROOM.INPUT)
```

It is recommended that local queues are dedicated to the File Transfer Application.

4. On the WebSphere MQ client, create an MQI channel by defining the MQSERVER environment variable as follows:

```
TO.HEAD.OFFICE/TCP/SERVER.COMPANY.COM(1414)
```

File Transfer Application

For more information on specifying the environment variable MQSERVER, see the *WebSphere MQ Clients* book.

Note: If you intend to implement SSL security, you must establish the MQI channel using a client channel definition table, and not by specifying the environment variable MQSERVER.

For more information on establishing MQI channels, see the *WebSphere MQ Clients* book.

5. Ensure that all the users of the File Transfer Application are members of the mqm group, or alternatively the local Administrators group on Windows.

Configuring the GUI

For every WebSphere MQ server or client system on which you intend to run the File Transfer Application GUI, you must perform some initial configuration tasks on the GUI. Before you can configure the File Transfer Application GUI, you must have setup up your system, see “Setup tasks” on page 566.

For information on what the File Transfer Application GUI provides, see “Components” on page 564.

To configure the File Transfer Application GUI, do the following:

1. Start the File Transfer Application.
The first time you start File Transfer Application, a message prompts you to complete the initial setup.
2. Click **OK**. The Queue Manager panel is displayed.
3. In the Name field, type the name of the queue manager to which the File Transfer Application is connecting.

Note: Ensure the queue manager is running.

4. Select either the **Local** or **Remote** radio button.
If connecting to a local queue manager, click the **Local** radio button.
If connecting to a remote queue manager, click the **Remote** radio button.
5. Click **OK**. The File Transfer panel is displayed, showing a list of queues.
6. Check the tick box next to every queue that is to be used as a destination queue.
7. Select the **Sources** tab.
8. Check the tick box next to every queue that is to be used as a source queue.
9. Click **OK**.

The File Transfer Application GUI is now configured.

File Transfer Application channel security

When using the File Transfer Application, a file is transferred over a message channel or an MQI channel. To ensure channels are secure you can use the Secure Sockets Layer (SSL), or channel exits. For information on channel security, see “Channel security” on page 146.

Using the File Transfer Application

Once a system administrator has set up the File Transfer Application for use, files can be sent and received using WebSphere MQ messaging. This section explains how a sender of a file, or receiver of a file, uses the File Transfer Application.

Sending a file

This task shows you how to send a file to another destination using the File Transfer Application GUI. You can also send files from the command line, or shell, by using **mqftsnd** on a WebSphere MQ server, or **mqftsndc** on a WebSphere MQ client, see “Using the command line” on page 572.

1. Start the File Transfer Application by issuing the control command **mqftapp**, or by selecting it through the start menu.
2. Click the **Send** tab.
3. Click the **Browse** button.
4. In the browser dialog, select the file to transfer, then click **OK**.
5. In the **Comments** field, type any accompanying comments. This field can be used to help identify different versions of the same file, for example: Week 3 sales - version 2.
6. In the **Destination** pane, click the required destination, for example: DESTINATION.AT.HEAD.OFFICE.
7. Click **Send**.
8. Look for confirmation that the file was sent in the **Show files sent and received** list.

The File Transfer Application transfers the specified file to the selected location. You can check that the file was sent by looking in the **Session log**.

Receiving a file

This task shows how to receive a file from another destination using the File Transfer Application GUI. You can also receive files from the command line, or shell, by using **mqftrcv** on a WebSphere MQ server, or **mqftrcvc** on a WebSphere MQ client, see “Using the command line” on page 572.

1. Start the File Transfer Application by issuing the control command **mqftapp**, or by selecting it through the start menu.
2. Click the **Receive** tab.
3. Click the **Files from** drop-down list to display the source where the file to be received is held, for example: SOURCE.AT.SHOWROOM.
4. Select the source to display the files stored there.
5. Select the file to receive, for example: Stocks 14 Aug 03.doc.
6. Look for confirmation that the file was received in the **Show files sent and received** list.

The selected file is received, and an icon representing the file is automatically displayed on the desktop. Double-click the icon to view the file. You can also check that the file was received by looking in the **Session log**.

Listing all sent and received files

This task shows how to create a list of all files sent and received using the File Transfer Application GUI.

1. Start the File Transfer Application by issuing the control command **mqftapp**, or by selecting it through the start menu.
2. In the Session Log pane, click **History Log...**
3. View the list of all files sent or received. The list shows all files sent or received since the File Transfer Application was first used, or since the log was last cleared.
4. Click **Save as**.

File Transfer Application

5. Navigate to a folder on the hard drive (c:) on your machine where you want to save the list.
6. In the **File name** field, type a meaningful file name, for example: File log 9 Apr 03.
7. In the **Save as type** field, type .txt.
8. Click the **Save** button.
9. Click the **Done** button.

File status

When a File Transfer Application receiver receives files, they can be listed using the File Transfer Application GUI, or by using either of the **receive file** control commands. For information on the **receive file** control commands see “mqftrcv (receive file on server)” on page 339, and “mqftrvc (receive file on client)” on page 342. Information about files in the source queue are returned as one of the following:

Complete file

When a file is too large to be transferred as a single WebSphere MQ message, the file is segmented into a number of smaller messages, known as segments. These segments are then transmitted.

A complete file is a file where every message that forms the file has been transferred to the destination queue.

Incomplete file

An incomplete file, is a file where a subset of the messages that form the file have been transferred to the queue. If you are using the File Transfer Application GUI, by clicking **Receive** again, you will see the file's percentage complete figure increase.

Other message

Other messages, are messages that have not been sent using the File Transfer Application. They are not related to a file.

If a dead letter queue has been associated with the queue manager and a **receive file** command is issued, then other messages are put on the dead letter queue. If a dead letter queue has not been associated with the queue manager, then other messages are left on the queue.

Using the command line

This section describes how to use the File Transfer Application commands directly, without the use of the File Transfer Application GUI. If you want to use the File Transfer Application, your user ID must be a member of the mqm group. For more information about this, see “Setup tasks” on page 566. In addition, note the following environment-specific information:

WebSphere MQ for Windows

All File Transfer Application commands can be issued from a command line. Command names and their flags are not case sensitive: you can enter them in uppercase, lowercase, or a combination of uppercase and lowercase. However, arguments to specify objects (such as queue names) are case sensitive. In the syntax descriptions, the hyphen (-) is used as a flag indicator. You can use the forward slash (/) instead of the hyphen.

WebSphere MQ for Linux (x86 platform)

All File Transfer Application commands can be issued from a shell. All commands are case-sensitive.

The control commands available with the File Transfer Application follow:

Table 46. File Transfer Application command files

Command name	Purpose
mqftapp	Run the File Transfer Application GUI. For more information, see “mqftapp (run File Transfer Application GUI)” on page 338.
mqftsnd	Send a file from a WebSphere MQ server. For more information, see “mqftsnd (send file from server)” on page 345.
mqftrcv	Receive a file on an WebSphere MQ server. For more information, see “mqftrcv (receive file on server)” on page 339.
mqftsndc	Send a file from a WebSphere MQ client. For more information, see “mqftsndc (send file from client)” on page 347.
mqftrcvc	Receive a file on an WebSphere MQ client. For more information, see “mqftrcvc (receive file on client)” on page 342.

The command line interface, sender, and receiver components are included as part of the WebSphere MQ clients for Windows and Linux (x86 platform). The clients are available as free downloads:

www-3.ibm.com/software/info1/websphere/index.jsp

File Transfer Application

Appendix F. Comparing command sets

The tables in this appendix compare the facilities available from the different administration command sets, and state whether you can perform each function from within the WebSphere MQ Explorer.

Queue manager commands

Table 47. Queue manager commands

Description	PCF command	MQSC command	Control command	WebSphere MQ Explorer equivalent?
Change Queue Manager	Change Queue Manager	ALTER QMGR	No equivalent	Yes
Create queue manager	No equivalent	No equivalent	crtmqm	Yes
Delete queue manager	No equivalent	No equivalent	dltmqm	Yes
Inquire Queue Manager	Inquire Queue Manager	DISPLAY QMGR	No equivalent	Yes
Inquire Queue Manager Status	Inquire Queue Manager Status	DISPLAY QMSTATUS	dspmq	Yes
Ping Queue Manager	Ping Queue Manager	PING QMGR	No equivalent	No
Refresh Queue Manager	No equivalent	REFRESH QMGR	No equivalent	No
Reset Queue Manager	Reset Queue Manager	RESET QMGR	No equivalent	No
Start queue manager	No equivalent	No equivalent	strmqm	Yes
Stop queue manager	No equivalent	No equivalent	endmqm	Yes

Command server commands

Table 48. Commands for command server administration

Description	PCF command	MQSC command	Control command	WebSphere MQ Explorer equivalent?
Display command server	Inquire Queue Manager Status	DISPLAY QMSTATUS	dspmqcsv	Yes
Start command server	Change Queue Manager	ALTER QMGR	strmqcsv	Yes
Stop command server	No equivalent	No equivalent	endmqcsv	Yes

Comparing command sets

Authority commands

Table 49. Commands for authority administration

PCF command	MQSC command	Control command	WebSphere MQ Explorer equivalent?
Delete authority record	No equivalent	setmqaut	No
Inquire authority records	No equivalent	dmpmqaut	No
Inquire entity authority	No equivalent	dspmqa	No
Refresh Security	REFRESH SECURITY	No equivalent	No
Set authority record	No equivalent	setmqaut	No

Cluster commands

Table 50. Cluster commands

PCF command	MQSC command	Control command	WebSphere MQ Explorer equivalent?
Inquire Cluster Queue Manager	DISPLAY CLUSQMGR	No equivalent	Yes
Refresh Cluster	REFRESH CLUSTER	No equivalent	Yes
Reset Cluster	RESET CLUSTER	No equivalent	No
Resume Queue Manager Cluster	RESUME QMGR	No equivalent	No
Suspend Queue Manager Cluster	SUSPEND QMGR	No equivalent	No

Authentication information commands

Table 51. Authentication information commands

PCF command	MQSC command	Control command	WebSphere MQ Explorer equivalent?
Change Authentication Information Object	ALTER AUTHINFO	No equivalent	Yes
Copy Authentication Information Object	DEFINE AUTHINFO(x) LIKE(y)	No equivalent	Yes
Create Authentication Information Object	DEFINE AUTHINFO	No equivalent	No
Delete Authentication Information Object	DELETE AUTHINFO	No equivalent	Yes
Inquire Authentication Information Object	DISPLAY AUTHINFO	No equivalent	Yes

Channel commands

Table 52. Channel commands

PCF command	MQSC command	Control command	WebSphere MQ Explorer equivalent?
Change Channel	ALTER CHANNEL	No equivalent	Yes
Copy Channel	DEFINE CHANNEL(x) LIKE(y)	No equivalent	Yes
Create Channel	DEFINE CHANNEL	No equivalent	No
Delete Channel	DELETE CHANNEL	No equivalent	Yes
Inquire Channel	DISPLAY CHANNEL	No equivalent	Yes
Inquire Channel Names	DISPLAY CHANNEL	No equivalent	Yes
Inquire Channel Status	DISPLAY CHSTATUS	No equivalent	Yes
Ping Channel	PING CHANNEL	No equivalent	Yes
Reset Channel	RESET CHANNEL	No equivalent	Yes
Resolve Channel	RESOLVE CHANNEL	No equivalent	Yes
Start Channel	START CHANNEL	runmqchl	Yes
Start Channel Initiator	START CHINIT	runmqchi	No
Stop Channel	STOP CHANNEL	No equivalent	Yes

Listener commands

Table 53. Listener commands

PCF command	MQSC command	Control command	WebSphere MQ Explorer equivalent?
Change Listener	ALTER LISTENER	No equivalent	Yes
Copy Listener	DEFINE LISTENER(x) LIKE(y)	No equivalent	Yes
Create Listener	DEFINE LISTENER	No equivalent	Yes
Delete Listener	DELETE LISTENER	No equivalent	Yes
Inquire Listener	DISPLAY LISTENER	No equivalent	Yes
Inquire Listener Status	DISPLAY LSSTATUS	No equivalent	Yes
Start Channel Listener	START LISTENER ¹	runmqlsr	Yes
Stop Listener	STOP LISTENER	endmqlsr ²	Yes
Notes:			
1. Used with listener objects only			
2. Stops all active listeners			

Comparing command sets

Namelist commands

Table 54. Namelist commands

PCF command	MQSC command	Control command	WebSphere MQ Explorer equivalent?
Change Namelist	ALTER NAMELIST	No equivalent	Yes
Copy Namelist	DEFINE NAMELIST(x) LIKE(y)	No equivalent	Yes
Create Namelist	DEFINE NAMELIST	No equivalent	Yes
Delete Namelist	DELETE NAMELIST	No equivalent	Yes
Inquire Namelist	DISPLAY NAMELIST	No equivalent	Yes
Inquire Namelist Names	DISPLAY NAMELIST	No equivalent	Yes

Process commands

Table 55. Process commands

PCF command	MQSC command	Control command	WebSphere MQ Explorer equivalent?
Change Process	ALTER PROCESS	No equivalent	Yes
Copy Process	DEFINE PROCESS(x) LIKE(y)	No equivalent	Yes
Create Process	DEFINE PROCESS	No equivalent	Yes
Delete Process	DELETE PROCESS	No equivalent	Yes
Inquire Process	DISPLAY PROCESS	No equivalent	Yes
Inquire Process Names	DISPLAY PROCESS	No equivalent	Yes

Queue commands

Table 56. Queue commands

PCF command	MQSC command	Control command	WebSphere MQ Explorer equivalent?
Change Queue	ALTER QLOCAL ALTER QALIAS ALTER QMODEL ALTER QREMOTE	No equivalent	Yes
Clear Queue	CLEAR QLOCAL	No equivalent	Yes
Copy Queue	DEFINE QLOCAL(x) LIKE(y) DEFINE QALIAS(x) LIKE(y) DEFINE QMODEL(x) LIKE(y) DEFINE QREMOTE(x) LIKE(y)	No equivalent	Yes
Create Queue	DEFINE QLOCAL DEFINE QALIAS DEFINE QMODEL DEFINE QREMOTE	No equivalent	No
Delete Queue	DELETE QLOCAL DELETE QALIAS DELETE QMODEL DELETE QREMOTE	No equivalent	Yes
Inquire Queue	DISPLAY QUEUE	No equivalent	Yes
Inquire Queue Names	DISPLAY QUEUE	No equivalent	Yes
Inquire Queue Status	DISPLAY QSTATUS	No equivalent	No
Reset Queue Statistics	No equivalent	No equivalent	No

Service commands

Table 57. Service commands

PCF command	MQSC command	Control command	WebSphere MQ Explorer equivalent?
Change Service	ALTER SERVICE	No equivalent	Yes
Copy Service	DEFINE SERVICE(x) LIKE(y)	No equivalent	Yes
Create Service	DEFINE SERVICE	No equivalent	No
Delete Service	DELETE SERVICE	No equivalent	Yes
Inquire Service	DISPLAY SERVICE	No equivalent	Yes
Inquire Service Status	DISPLAY SVSTATUS	No equivalent	Yes
Start Service	START SERVICE	No equivalent	Yes
Stop Service	STOP SERVICE	No equivalent	Yes

Comparing command sets

Other commands

Table 58. Other commands

Description	PCF command	MQSC command	Control command	WebSphere MQ Explorer equivalent?
Create conversion exit	No equivalent	No equivalent	<code>crtmqcvx</code>	No
Display files used by objects	No equivalent	No equivalent	<code>dspmqfls</code>	No
Display formatted trace	No equivalent	No equivalent	<code>dspmqtrc</code> ¹	No
Display version information	No equivalent	No equivalent	<code>dspmqver</code>	No
Display transactions	No equivalent	No equivalent	<code>dspmqtrn</code>	No
Dump log	No equivalent	No equivalent	<code>dmpmqlog</code>	No
End trace	No equivalent	No equivalent	<code>endmqtrc</code>	Yes
Escape	Escape	No equivalent	No equivalent	No
Record media image	No equivalent	No equivalent	<code>rcdmqimg</code>	No
Recreate media object	No equivalent	No equivalent	<code>rcrmqobj</code>	No
Resolve transactions	No equivalent	No equivalent	<code>rsvmqtrn</code>	No
Run client trigger monitor	No equivalent	No equivalent	<code>runmqtrmc</code>	No
Run dead-letter queue handler	No equivalent	No equivalent	<code>runmqdlq</code>	No
Run MQSC commands	No equivalent	No equivalent	<code>runmqsc</code>	No
Run trigger monitor	No equivalent	No equivalent	<code>runmqtrm</code>	No
Set service connection points	No equivalent	No equivalent	<code>setmqscp</code> ²	No
Start WebSphere MQ trace	No equivalent	No equivalent	<code>strmqtrc</code>	Yes
WebSphere MQ Services control	No equivalent	No equivalent	<code>amqmdain</code> ²	No

Notes:

1. Not supported on WebSphere MQ for Windows.
2. Supported by WebSphere MQ for Windows only.

Appendix G. WebSphere MQ and UNIX System V IPC resources

This information applies to WebSphere MQ running on UNIX systems only.

WebSphere MQ uses System V interprocess communication (IPC) resources (*semaphores* and *shared memory segments*) to store and pass data between system components. These resource are used by queue manager processes and applications that connect to the queue manager. WebSphere MQ clients do not use System V IPC resources. Use the UNIX command `ipcs` to view the number and size of these resources that the system uses.

Clearing WebSphere MQ shared memory resources

When a WebSphere MQ queue manager stops, it releases the IPC resources that it was using back to the system. There are two situations in which this might not happen automatically:

- If applications are connected to the queue manager when it stops (perhaps because the queue manager was shut down using `endmqm -i` or `endmqm -p`), the IPC resources used by these applications might not be released.
- If the queue manager ends abnormally (for example, if an operator issues the system `kill` command), some IPC resources might be left allocated after all queue manager processes have terminated.

In these cases, the IPC resources are not release back to the system until you manually remove them, or restart (`strmqm`) or delete (`dltmqm`) the queue manager.

WebSphere MQ provides a utility to release System V IPC resources allocated by the queue manager but not freed, for one of the above reasons, back to the system.

To check and free any unused System V IPC resources do the following

1. Log on as user `mqm`
2. End the queue manager
3. Type the following:

On Solaris, HP-UX, and Linux:

```
/opt/mqm/bin/amqiclen -x -m QMGR
```

On AIX:

```
/usr/mqm/bin/amqiclen -x -m QMGR
```

This command does not report any status. However, if some WebSphere MQ-allocated resources could not be freed because they were still in use, the return code is nonzero.

Shared memory on AIX

The AIX model for System V shared memory differs from other UNIX platforms, in that a 32-bit process can only attach to 10 WebSphere MQ memory segments concurrently.

A typical 32-bit WebSphere MQ application requires two WebSphere MQ memory segments attached for every connected queue manager. Every additional connected queue manager requires one further WebSphere MQ memory segment attached.

Note: During the MQCONN operation an additional shared memory segment is required. In a threaded process where multiple threads are connecting to the same queue manager, you must ensure an additional memory segment is available for every connected queue manager.

A 64-bit process is not limited to attaching to only 10 WebSphere MQ memory segments concurrently. A typical 64-bit WebSphere MQ application requires three WebSphere MQ memory segments for every connected queue manager. The connection of additional queue managers typically requires two further WebSphere MQ memory segments for every connected queue manager. Applications that connect to heavily loaded queue managers can require additional memory segments.

WebSphere MQ Version 5.3 recommended the use of the environment variable EXTSHM to allow 32-bit applications to connect to more than 10 WebSphere MQ memory segments at a time. With WebSphere MQ Version 6, the setting of the EXTSHM variable has no effect on the shared memory used by WebSphere MQ.

Appendix H. Common Criteria

Common Criteria is a scheme for independent assessment, analysis, and testing of IT products to a set of security requirements. The Common Criteria Scheme provides consumers with an impartial security assurance of a product to predefined levels. These levels range from EAL0 to EAL7, each assurance level places increased demands on the developer for evidence of testing, and provides increased assurance within the product.

WebSphere MQ V5.3.0.2 with Corrective Service Diskette (CSD) 6, has been evaluated to Common Criteria EAL2. This provides assurance that the product has been structurally tested.

Under the Common Criteria Recognition Arrangement (CCRA), countries agree to recognize Common Criteria certificates that have been produced by any certificate authorizing participant, in accordance with the terms laid out in the CCRA. Currently, the CCRA is comprised of nineteen member nations: Australia, Austria, Canada, Finland, France, Germany, Greece, Hungary, Israel, Italy, Japan, the Netherlands, New Zealand, Norway, Spain, Sweden, Turkey, the United Kingdom, and the United States. New members are expected to join in the near future.

You can find further information on the Common Criteria scheme at the following Web site: <http://www.csrc.nist.gov/cc>

Environmental Considerations

In order that WebSphere MQ operates in accordance with its Common Criteria certificate, the environmental requirements defined in this section need to be met.

- There must be one or more competent individuals that are assigned to manage WebSphere MQ and the security of the information that it contains. Such personnel are assumed not to be careless, wilfully negligent or hostile.
- The operating system must be configured in accordance with the manufacturer's installation guides and where applicable, in its evaluated configuration. It must be securely configured such that the operating system protects WebSphere MQ from any unauthorized users or processes.

The following operating systems are supported within the evaluation:

- AIX 5L
- HP-UX 11i
- SUSE Linux Enterprise Server 8 (Linux (x86 platform) and Linux (zSeries platform))
- RedHat Enterprise Linux AS 2.1 (Linux (x86 platform))
- Solaris Version 8
- Solaris Version 9
- Windows 2000 (including all combinations of Advanced Server, Server, Professional, Service Packs and hotfixes)
- Windows 2003 (including all combinations of Standard, Enterprise, Service Packs, and hotfixes)

Common Criteria

WebSphere MQ relies on the operating system to provide user/group IDs and time and date information. In addition, you need an application to read the event logs so that the audit records produced by WebSphere MQ can be read.

The evaluation of WebSphere MQ does not include the following aspects:

- The operating system
- Remote administration
- WebSphere MQ Explorer
- Windows Default Configuration application
- Remote queue management. This excludes WebSphere MQ clients, channels, and Message Channel Agents from the evaluation
- Third-party or user-written authorization services not supplied with the WebSphere MQ product.

Configuration Requirements

In order that auditing of authority events is implemented, execute the following MQSC command:

```
ALTER QMGR AUTHOREV (ENABLED)
```

If **AUTHOREV** is disabled, auditing will no longer be performed and WebSphere MQ will not operate in accordance with the evaluated configuration. To confirm whether auditing of the authority events is enabled, execute the following MQSC command:

```
DISPLAY QMGR
```

Appendix I. Notices

This information was developed for products and services offered in the United States. IBM may not offer the products, services, or features discussed in this information in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this information. The furnishing of this information does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the information. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this information at any time without notice.

Any references in this information to non-IBM 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.

Notices

Licenses of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM United Kingdom Laboratories,
Mail Point 151,
Hursley Park,
Winchester,
Hampshire,
England
SO21 2JN.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Programming License Agreement, or any equivalent agreement between us.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

Trademarks

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

AIX	CICS	DB2
Encina	FFST	First Failure Support Technology
HACMP	IBM	Informix
iSeries	Lotus	MQSeries
Notes	RACF	Tivoli
TXSeries	WebSphere	z/OS
zSeries		

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

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

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

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

Index

Special characters

.ini files

See configuration files

A

access control 131, 141

access settings 144, 145

accidental deletion of default queue manager 301

AccountingToken field

MQZIC structure 486

ACPI (Advanced Configuration and Power Interface) 114

ACTION keyword, rules table 199

Active Directory Service Interfaces (ADSI)

See ADSI (Active Directory Service Interfaces)

administration

authority 129

control commands 25

description of 19

for database managers 183

introduction to 17

local, definition of 17

MQAI, using 62

MQSC commands 18, 36

object name transformation 21

PCF commands 61

queue manager name

transformation 20

remote administration, definition of 17

remote objects 65

understanding WebSphere MQ file names 20

using control commands 17

using PCF commands 18

using the WebSphere MQ

Explorer 81

writing Eclipse plug-ins 93

ADSI (Active Directory Service Interfaces)

description of 63

IBMMQSeries namespace 63

Advanced Configuration and Power

Interface (ACPI) 114

AIX

trace data, sample 264

AIX operating system

DB2 switch load file, creating 174

Informix switch load file,

creating 179

MQAI support 62

Oracle switch load file, creating 176

performance of nonpersistent

messages 254

security 139

start client trigger monitor

(runmqtm) command 366

AIX operating system (*continued*)

sybswit, creating the Sybase switch

load file 181

trace data, example 267

tracing 260, 265

alert monitor application, using 87

alias queues

DEFINE QALIAS command 50

defining alias queues 50

remote queues as queue manager

aliases 76

reply-to queues 76

working with alias queues 50

aliases

queue manager aliases 76

working with alias queues 50

AllQueueManagers stanza, mqs.ini 109

alternate-user authority 134

amqccert command (check certificate chains) 283

amqmdain (WebSphere MQ Services

control) command

format 285

keywords 286

parameters 286

purpose 285

return codes 284, 289, 295

AMQMSRVN

changing the password 90

amqsdlq, the sample DLQ handler 196

amqtcert command (transfer certificates) 291

API exit

MQXEP 524

API exits 13

APICallerType field

MQAXP structure 520

ApiExitCommon stanza, mqs.ini 115

ApiExitLocal stanza, qm.ini 127

ApiExitTemplate stanza, mqs.ini 115

application programs

design considerations 254

message length, effects on

performance 254

MQI local administration, support

for 35

persistent messages, effect on

performance 255

programming errors, examples

of 247

receiving messages 4

retrieving messages from queues 5

searching for messages, effect on

performance 255

sending messages 4

threads, application design 255

time-independent applications 3

application queues

defining application queues for

triggering 58

ApplicationContext parameter

authenticate user call 423

APPLIDAT keyword, rules table 198

ApplIdentityData field

MQZIC structure 486

ApplName field

MQAXC structure 514

MQZAC structure 477

APPLNAME keyword, rules table 198

ApplType field

MQAXC structure 515

APPLTYPE keyword, rules table 198

attributes

changing local queue attributes 47

LIKE attribute, DEFINE command 46

queue manager 43, 44

queues 7

WebSphere MQ and PCF commands,

a comparison 62

authentication information objects

description of 10

AuthenticationType field

MQZAC structure 478

authority

administration 129

alternate-user 134

context 134

set/reset command 368

Authority field

MQZAD structure 480

Authority parameter

check authority (extended) call 432

check authority call 427

get authority (extended) call 451

get authority call 448

get explicit authority (extended)

call 457

get explicit authority call 454

set authority (extended) call 472

set authority call 469

AuthorityBuffer parameter

enumerate authority data call 443

AuthorityBufferLength parameter

enumerate authority data call 443

AuthorityDataLength parameter

enumerate authority data call 443

authorization

migrating data from MQSeries Version

5.1 410

refreshing the OAM after changing a

user's 409

authorization service 13, 401

component 409

defining to WebSphere MQ for UNIX

systems 409

defining to WebSphere MQ for

Windows 409

stanza, UNIX systems 410

stanza, Windows 412

user interface 412

- authorizations
 - MQI 150
 - specification tables 150
- automatic definition of channels 70

B

- backing up queue manager data 235
- BindType field
 - MQZAC structure 478
- browsing queues 48
- built-in formats, data conversion 76

C

- calculating the size of logs 228
- CallerType field
 - MQZAC structure 477
- calls
 - detailed description
 - MQ_BACK_EXIT 527
 - MQ_BEGIN_EXIT 528
 - MQ_CLOSE_EXIT 529
 - MQ_CMIT_EXIT 530
 - MQ_CONNX_EXIT 531
 - MQ_DISC_EXIT 533
 - MQ_GET_EXIT 534
 - MQ_INIT_EXIT 536
 - MQ_INQ_EXIT 537
 - MQ_OPEN_EXIT 539
 - MQ_PUT_EXIT 540
 - MQ_PUTI_EXIT 542
 - MQ_SET_EXIT 544
 - MQ_TERM_EXIT 546
- case-sensitive control commands 25
- ccsid.tbl, data conversion 77
- certificates, checking chains with
 - amqccert 283
- certificates, migrating with amqtcert 291
- certificates, transferring with
 - amqtcert 291
- ChainAreaLength field
 - MQACH structure 510
- changing
 - CCSID 78
 - local queue attributes 47
 - queue manager attributes 44
 - the default queue manager 30
- channel exits
 - security 148
- channels
 - administering a remote queue
 - manager from a local one 67
 - auto-definition of 70
 - channel commands 577
 - CHANNELS stanza, qm.ini 122
 - defining channels for remote
 - administration 69
 - description of 11, 65
 - escape command authorizations 153
 - exits 13, 148
 - preparing channels for remote
 - administration 68
 - remote queuing 65
 - security 146
 - starting 70

- channels (*continued*)
 - using the run channel (runmqchl)
 - command 356
 - using the run initiator (runmqchi)
 - command 355
- CHANNELS stanza, qm.ini 122
- character code sets, updating 77
- CharAttrCount parameter
 - inquire authorization service call 463
- CharAttrs parameter
 - inquire authorization service call 463
- check certificate chains, amqccert
 - command 283
- CICS
 - enabling the two-phase commit
 - process 193
 - requirements, two-phase commit
 - process 192
 - task termination exit, UE014015 193
 - two-phase commit process 192
 - user exits, enabling 193
 - XA-compliance 192
- circular logging 224
- clearing a local queue 47
- clearing WebSphere MQ shared memory
 - resources 581
- client connection channels
 - description of 11
- ClientExitPath stanza, mqs.ini 110
- clients and servers
 - definitions 12
 - problem determination 271
 - start client trigger monitor
 - (runmqmtmc) command 366
- clusters
 - cluster membership, the WebSphere
 - MQ Explorer 83
 - cluster transmission queues 8
 - description of 10, 66
 - ExitProperties stanza attributes 111
 - remote queuing 65
 - showing and hiding, WebSphere MQ
 - Explorer 86
- coded character sets, specifying 77
- command files 39
- command queues
 - command server status 71
 - description of 9
 - mandatory for remote
 - administration 68
- command server
 - authentication information
 - commands 576
 - cluster commands 576
 - command server commands 575
 - commands for authority
 - administration 576
 - display command server (dspmqcsv)
 - command 316
 - displaying status 71
 - end command server (endmqcsv)
 - command 331
 - listener commands 577
 - namelist commands 578
 - remote administration 70
 - service commands 579
 - starting a command server 71

- command server (*continued*)
 - starting the command server
 - (strmqcsv) command 380
 - stopping a command server 71
- command sets
 - comparison of sets 575
 - control commands 25
 - MQSC commands 36
 - PCF commands 61
- commands
 - check certificate chains (amqccert)
 - command 283
 - comparison of command sets 575
 - control commands 25
 - create queue manager (crtmqm)
 - command 299
 - data conversion (crtmqcvx)
 - command 297
 - delete queue manager (dltmqm)
 - command 303
 - display authority (dspmqaut)
 - command 312
 - display command server (dspmqcsv)
 - command 316
 - display version information
 - (dspmqver) 329
 - display WebSphere MQ files
 - (dspmqfls) command 317
 - display WebSphere MQ formatted
 - trace (dspmqtrc) command 327
 - display WebSphere MQ queue
 - managers (dspmq) command 311
 - display WebSphere MQ transactions
 - (dspmqtrn) command 328
 - dmpmqaut 144
 - dspmqaut 145
 - dump authority (dmpmqaut)
 - command 305
 - dump log (dmpmqlog)
 - command 309
 - end .NET monitor (endmqdnm) 334
 - end command server (endmqcsv)
 - command 331
 - end listener (endmqslr)
 - command 333
 - end queue manager (endmqm)
 - command 335
 - end WebSphere MQ trace (endmqtrc)
 - command 337
 - for authentication information
 - objects 576
 - for authority administration 576
 - for channel objects 577
 - for clusters 576
 - for command server
 - administration 575
 - for listeners 577
 - for namelist objects 578
 - for process objects 578
 - for queue objects 579
 - for service objects 579
 - gsk7cmd 387
 - help with syntax 280
 - issuing MQSC commands using an
 - ASCII file 36
 - other commands 580
 - PCF commands 61

- commands (*continued*)
 - queue manager objects 575
 - receive file on client (mqftrcv) 342
 - receive file on server (mqftrcv) 339
 - record media image (rcdmqimg)
 - command 349
 - recreate object (rcrmqobj)
 - command 351
 - resolve WebSphere MQ transactions (rsvmqtrn) command 353
 - run .NET monitor (runmqdnm) 358
 - command 356
 - run channel initiator (runmqchi) 355
 - run dead-letter queue handler 357
 - run DLQ handler (runmqdlq)
 - command 195
 - run File Transfer Application (mqftapp) 338
 - run listener (runmqlsr)
 - command 361
 - run MQSC commands (runmqsc) 363
 - runmqckm 387
 - runmqsc command, to issue MQSC commands 36
 - send file from client (mqftsndc) 347
 - send file from server (mqftsnd) 345
 - services control (amqmdain)
 - command 285
 - set CRL LDAP server definitions 375
 - set service connection points (setmqscp) 377
 - set/reset authority (setmqaut) 368
 - setmqaut 142
 - shell, WebSphere MQ for UNIX systems 26
 - start client trigger monitor (runmqtm) command 366
 - start command server (strmqcsv) 380
 - start queue manager (strmqm) 381
 - start trigger monitor (runmqtrm) 367
 - start WebSphere MQ Explorer (strmqcfg) 379
 - start WebSphere MQ trace (strmqtrc) 383
 - transfer certificates (amqtcert)
 - command 291
 - verifying MQSC commands 41
 - WebSphere MQ display route application (dspmqrte) 319
- common criteria 583
- CompCode parameter
 - authenticate user call 424
 - check authority (extended) call 434
 - check authority call 429
 - copy all authority call 437
 - delete authority call 440
 - enumerate authority data call 444
 - free user call 445
 - get authority (extended) call 452
 - get authority call 449
 - get explicit authority (extended) call 458
 - get explicit authority call 455
 - initialize authorization service call 460
 - initialize name service call 492
- CompCode parameter (*continued*)
 - inquire authorization service call 464
 - insert name call 495
 - lookup name call 497
 - MQ_GET_EXIT call 534
 - MQZ_DELETE_NAME call 490
 - MQZEP call 421
 - set authority (extended) call 473
 - set authority call 470
 - terminate authorization service call 475
 - terminate name service call 500
- ComponentData parameter
 - authenticate user call 424
 - check authority (extended) call 434
 - check authority call 429
 - copy all authority call 437
 - delete authority call 440
 - enumerate authority data call 443
 - free user call 445
 - get authority (extended) call 451
 - get authority call 448
 - get explicit authority (extended) call 457
 - get explicit authority call 454
 - initialize authorization service call 459
 - initialize name service call 491
 - inquire authorization service call 464
 - insert name call 494
 - lookup name call 496
 - MQZ_DELETE_NAME call 489
 - set authority (extended) call 472
 - set authority call 469
 - terminate authorization service call 474
 - terminate name service call 499
- ComponentDataLength parameter
 - initialize authorization service call 459
 - initialize name service call 491
- components, File Transfer Application 563
- components, installable services 401
- configuration file
 - authorization service 409
- configuration files
 - AllQueueManagers stanza, mqs.ini 109
 - ApiExitCommon, mqs.ini 115
 - ApiExitLocal, qm.ini 127
 - ApiExitTemplate, mqs.ini 115
 - backing up of 30
 - CHANNELS stanza, qm.ini 122
 - ClientExitPath stanza, mqs.ini 110
 - databases, qm.ini 121
 - DefaultQueueManager stanza, mqs.ini 110
 - editing 104
 - example mqs.ini file, MQSeries for UNIX systems 105
 - example qm.ini file, WebSphere MQ for UNIX systems 107
 - ExitPath stanza, qm.ini 126
 - ExitProperties stanza, mqs.ini 111
 - Log stanza, qm.ini 118
 - LogDefaults stanza, mqs.ini 111
- configuration files (*continued*)
 - LU62 stanza, qm.ini 124
 - mqs.ini, description of 105
 - NETBIOS stanza, qm.ini 124
 - priorities 105
 - queue manager configuration file, qm.ini 107
 - QueueManager stanza, mqs.ini 115
 - RestrictedMode stanza, qm.ini 120
 - Service stanza, qm.ini 116
 - ServiceComponent stanza, qm.ini 117
 - SPX stanza, qm.ini 124
 - TCP stanza, qm.ini 124
 - XAResourceManager stanza, qm.ini 121
- configuration information 103
- configuring
 - database products 170
 - DB2 173
 - Informix 178
 - logs 118
 - multiple databases 182
 - Oracle 176
 - Sybase 180
- configuring your system for database coordination 170
- ConnectionName field
 - MQAXC structure 514
- context authority 134
- Continuation parameter
 - authenticate user call 424
 - check authority (extended) call 434
 - check authority call 429
 - copy all authority call 437
 - delete authority call 440
 - enumerate authority data call 443
 - free user call 445
 - get authority (extended) call 451
 - get authority call 448
 - get explicit authority (extended) call 457
 - get explicit authority call 454
 - inquire authorization service call 464
 - insert name call 495
 - lookup name call 497
 - MQZ_DELETE_NAME call 489
 - set authority (extended) call 472
 - set authority call 469
- control commands
 - case sensitivity of 25
 - categories of 25
 - changing the default queue manager 30
 - controlled shutdown 31
 - creating a default queue manager 29
 - creating a queue manager 26
 - crtmqm, creating a default queue manager 29
 - deleting a queue manager, dltnmqm 33
 - dltnmqm, deleting a queue manager 33
 - endmqm, stopping a queue manager 31
 - for WebSphere MQ for Windows systems 25

- control commands (*continued*)
 - for WebSphere MQ for UNIX systems 26
 - immediate shutdown 32
 - preemptive shutdown 32
 - quiesced shutdown 32
 - restarting a queue manager, strmqm 33
 - runmqsc, using interactively 37
 - starting a queue manager 31
 - stopping a queue manager, endmqm 31
 - strmqm, restarting a queue manager 33
 - strmqm, starting a queue manager, 31
 - using 25
- controlled shutdown of a queue manager 31, 32
- CorrelationPtr field
 - MQZED structure 484
 - MQZFP structure 487
- CorrelationPtr parameter
 - authenticate user call 424
- CorrelId, performance considerations 255
- create queue manager command (crtmqm)
 - See crtmqm (create queue manager) command
- creating
 - a default queue manager 29
 - a dynamic (temporary) queue 5
 - a model queue 5
 - a predefined (permanent) queue 5
 - a process definition 60
 - a queue manager 26, 299
 - a transmission queue 75
- creating service components 406
- crtmqcvx (data conversion) command
 - examples 297
 - format 297
 - parameters 297
 - purpose 297
 - return codes 297
- crtmqm (create queue manager) command
 - examples 302
 - format 299
 - parameters 299
 - purpose 299
 - related commands 302
 - return codes 302
- CURDEPTH, current queue depth 46
- current queue depth, CURDEPTH 46

D

- data conversion
 - built-in formats 76
 - ccsid.tbl, uses for 77
 - ConvEBCDICNewline attribute, AllQueueManagers stanza 109
 - converting user-defined message formats 78
 - data conversion (crtmqcvx) command 297

- data conversion (*continued*)
 - data conversion for the WebSphere MQ Explorer 86
 - default data conversion 77
 - EBCDIC NL character conversion to ASCII 109
 - introduction 76
 - updating coded character sets 77
- data conversion command (crtmqcvx)
 - See crtmqcvx (data conversion) command
- data types, detailed description
 - elementary
 - MQHCONFIG 422
 - PMQFUNC 422
 - structure
 - MQACH 509
 - MQAXC 512
 - MQAXP 516
 - MQZAC 476
 - MQZAD 478
 - MQZED 483
 - MQZFP 487
 - MQZIC 485
- database managers
 - changing the configuration information 186
 - connections to 170
 - coordination
 - application program crashes 168
 - configuring database product 170
 - configuring for 170
 - database crashes 167
 - installing database product 170
 - introduction 166
 - restrictions 168
 - switch function pointers 169
 - switch load files 169
 - database manager instances, removing 186
 - dspmqrtn command, checking outstanding units of work 184
 - in-doubt units of work 184
 - multiple databases, configuring 182
 - restrictions, database coordination support 168
 - rsvmqtrn command, explicit resynchronization of units of work 185
 - security considerations 183
 - server crashes 167
 - switch load files, creating 170
 - syncpoint coordination 190
- database products
 - configuring 170
 - installing 170
- DB2
 - adding XAResourceManager stanza 174
 - configuring 173
 - DB2 configuration parameters, changing 175
 - environment variable settings 174
 - explicit resynchronization of units of work 185
 - security considerations 183
 - switch load file, creating 174

- DB2 (*continued*)
 - switch load file, creating on UNIX 174
 - switch load file, creating on Windows systems 174
- DCE Generic Security Service (GSS)
 - name service, installable service 13
- DCOMCNFG.EXE, WebSphere MQ Explorer 89
- dead-letter header, MQDLH 195
- dead-letter queue handler
 - ACTION keyword, rules table 199
 - action keywords, rules table 199
 - APPLIDAT keyword, rules table 198
 - APPLNAME keyword, rules table 198
 - APPLTYPE keyword, rules table 198
 - control data 196
 - DESTQ keyword, rules table 198
 - DESTQM keyword, rules table 198
 - example of a rules table 204
 - FEEDBACK keyword, rules table 198
 - FORMAT keyword, rules table 198
 - FWDQ keyword, rules table 199
 - FWDQM keyword, rules table 200
 - HEADER keyword, rules table 200
 - INPUTQ, rules table 196
 - INPUTQM keyword, rules table 197
 - invoking the DLQ handler 195
 - MSGTYPE keyword, rules table 199
 - pattern-matching keywords, rules table 198
 - patterns and actions (rules) 198
 - PERSIST keyword, rules table 199
 - processing all DLQ messages 203
 - processing rules, rules table 202
 - PUTAUT keyword, rules table 200
 - REASON keyword, rules table 199
 - REPLYQ keyword, rules table 199
 - REPLYQM keyword, rules table 199
 - RETRY keyword, rules table 200
 - RETRYINT, rules table 197
 - rule table conventions 200
 - rules table, description of 196
 - sample, amqsdlq 196
 - syntax rules, rules table 201
 - USERID keyword, rules table 199
 - WAIT keyword, rules table 197
- dead-letter queues
 - defining a dead-letter queue 45
 - description of 8
 - DLQ handler 357
 - MQDLH, dead-letter header 195
 - specifying 28
- debugging
 - command syntax errors 248
 - common command errors 248
 - common programming errors 247
 - further checks 249
 - preliminary checks 245
- default configuration, Windows systems 19
- default data conversion 77
- default transmission queues 75
- DefaultQueueManager stanza, mqs.ini 110

- defaults
 - changing the default queue manager 30
 - creating a default queue manager 29
 - objects 12, 549
 - queue manager 27
 - reverting to the original default queue manager 30
 - transmission queue 28
 - defining
 - a model queue 52
 - an alias queue 50
 - an initiation queue 59
 - WebSphere MQ queues 6
 - delete queue manager command (dlmqm)
 - See* dlmqm (delete queue manager) command
 - deleting
 - a local queue 47
 - a queue manager 33
 - a queue manager using the dlmqm command 303
 - queue managers, WebSphere MQ for UNIX systems 561
 - Windows queue managers 560
 - Windows queue managers, automatic startup list 561
 - DESTQ keyword, rules table 198
 - DESTQM keyword, rules table 198
 - determining current queue depth 46
 - diagnostics
 - Java 271
 - directories
 - directory structure (UNIX) 555
 - directory structure, Windows systems 553
 - display
 - current authorizations (dmpmqaut) command 305
 - current authorizations (dspmqaut) command 312
 - default object attributes 46
 - file system name (dspmqfls) command 317
 - process definitions 60
 - queue manager attributes 43
 - queue managers (dspmq) command 311
 - status of command server 71
 - status of command server (dspmqcsv) command 316
 - WebSphere MQ formatted trace (dspmqtrc) command 327
 - WebSphere MQ transactions (dspmqtrn) command 328
 - display authority command (dspmqaut)
 - See* dspmqaut (display authority) command
 - display command server command (dspmqcsv)
 - See* dspmqcsv (display command server) command
 - display version information, dspmqver command 329
 - display WebSphere MQ files command (dspmqfls)
 - See* dspmqfls (display WebSphere MQ files) command
 - display WebSphere MQ formatted trace output command (dspmqtrc)
 - See* dspmqtrc (display WebSphere MQ formatted trace) command
 - display WebSphere MQ queue managers (dspmq)
 - See* dspmq (display WebSphere MQ queue managers) command
 - display WebSphere MQ transactions command (dspmqtrn)
 - See* dspmqtrn (display WebSphere MQ transactions) command
 - distributed queuing, incorrect output 251
 - dlmqm (delete queue manager) command
 - examples 303
 - format 303
 - parameters 303
 - purpose 303
 - related commands 284, 296, 303
 - return codes 303
 - dlmqm control command 33
 - dmpmqlog (dump log) command
 - format 309
 - parameters 309
 - purpose 309
 - domain controller
 - security 162
 - dspmq (display WebSphere MQ queue managers) command
 - format 311
 - parameters 311
 - purpose 311
 - return codes 311
 - dspmqaut (display authority) command
 - dspmqaut command 314
 - examples 306, 315
 - format 312
 - parameters 312
 - purpose 305, 312
 - related commands 315
 - results 313
 - return codes 314
 - dspmqcsv (display command server) command
 - examples 316
 - format 316
 - parameters 316
 - purpose 316
 - related commands 316
 - return codes 316
 - dspmqfls (display WebSphere MQ files) command
 - examples 318
 - format 317
 - parameters 317
 - purpose 317
 - return codes 318
 - dspmqrte
 - format 319
 - parameters 319
 - dspmqtrc (display WebSphere MQ formatted trace) command
 - format 327
 - parameters 327
 - purpose 327
 - related commands 327
 - dspmqtrn (display WebSphere MQ transactions) command
 - format 328
 - parameters 328
 - purpose 328
 - related commands 328
 - return codes 328
 - dspmqver
 - examples 330
 - format 329
 - parameters 329
 - dump
 - dumping log records (dmpmqlog command) 240
 - dumping the contents of a recovery log 240
 - formatted system log (dmpmqlog) command 309
 - dump log command (dmpmqlog)
 - See* dmpmqlog (dump log) command
 - dynamic binding 405
 - dynamic definition of channels 70
 - dynamic queues
 - description of 5
- ## E
- EBCDIC NL character conversion to ASCII 109
 - EffectiveUserID field
 - MQZAC structure 477
 - end command server command (endmqcsv)
 - See* endmqcsv (end command server) command
 - end listener command (endmqslr)
 - See* endmqslr (end listener) command
 - end queue manager command (endmqm)
 - See* endmqm (end queue manager) command
 - end WebSphere MQ trace command (endmqtr)
 - See* endmqtr (end WebSphere MQ trace) command
 - ending
 - a queue manager 31
 - interactive MQSC commands 38
 - endmqcsv (end command server) command
 - examples 331
 - format 331
 - parameters 331
 - purpose 331
 - related commands 332
 - return codes 331
 - endmqdnm
 - format 334
 - parameters 334
 - endmqslr (end listener) command
 - format 333
 - parameters 333

- endmqlsr (end listener) command
 - (continued)
 - purpose 333
 - return codes 333
 - endmqm (end queue manager) command
 - examples 336
 - format 335
 - parameters 335
 - purpose 335
 - related commands 336
 - return codes 336
 - endmqtr (end WebSphere MQ trace)
 - command
 - examples 337
 - format of 337
 - parameters 337
 - purpose of 337
 - related commands 337
 - return codes 337
 - syntax of 337
 - EntityData parameter
 - check authority (extended) call 431
 - get authority (extended) call 450
 - get explicit authority (extended) call 456
 - set authority (extended) call 471
 - EntityDataPtr field
 - MQZAD structure 480
 - EntityDomainPtr field
 - MQZED structure 484
 - EntityName parameter
 - check authority call 426
 - get authority call 447
 - get explicit authority call 453
 - set authority call 468
 - EntityNamePtr field
 - MQZED structure 483
 - EntityType field
 - MQZAD structure 481
 - EntityType parameter
 - check authority (extended) call 431
 - check authority call 426
 - get authority (extended) call 450
 - get authority call 447
 - get explicit authority (extended) call 456
 - get explicit authority call 453
 - set authority (extended) call 471
 - set authority call 468
 - EntryPoint parameter
 - MQXEP call 525
 - MQZEP call 421
 - Environment field
 - MQAXC structure 513
 - MQZAC structure 477
 - environment variables
 - DB2INSTANCE 174
 - INFORMIXDIR 178
 - INFORMIXSERVER 178
 - MQS_TRACE_OPTIONS 266
 - MQSPREFIX 109
 - ONCONFIG 178
 - ORACLE_HOME, Oracle 176
 - ORACLE_SID, Oracle 176
 - SYBASE_OCS, Sybase 180
 - SYBASE, Sybase 180
 - error codes, ignoring under UNIX
 - systems 258
 - error codes, ignoring under Windows
 - systems 258
 - error logs
 - description of 255
 - errors occurring before log established 257
 - log files 256
 - error messages, MQSC commands 38
 - escape PCFs 62
 - event queues
 - description of 9
 - examples
 - amqccert command 283
 - amqmdain command 289
 - amqtcert command 294
 - creating a transmission queue 75
 - crtmqcvx command 297
 - crtmqm command 302
 - dltnmqm command 303
 - dmpmqaut command 306
 - dspmqaout command 315
 - dspmqcvs command 316
 - dspmqls command 318
 - dspmqrte command 326
 - dspmqrver command 330
 - endmqcvs command 331
 - endmqm command 336
 - endmqtrc command 337
 - mqftrvc command 341
 - mqftrvcv command 344
 - mqftsnd command 346
 - mqftsndc command 348
 - mqs.ini file, MQSeries for UNIX systems 105
 - programming errors 247
 - qm.ini file, WebSphere MQ for UNIX systems 107
 - rcdmqing command 350
 - rcrmqobj command 352
 - runmqlsr command 362
 - runmqsc command 364
 - runmqtmc command 366
 - setmqaut command 373
 - setmqscp command 375, 377
 - strmqcvs command 380
 - strmqm command 382
 - strmqtrc command 386
 - trace data (AIX) 267
 - ExitChainAreaPtr field
 - MQAXP structure 522
 - ExitContext parameter
 - MQ_INIT_EXIT call 531
 - ExitData field
 - MQAXP structure 521
 - ExitId field
 - MQAXP structure 517
 - ExitInfoName field
 - MQACH structure 511
 - MQAXP structure 521
 - ExitPath stanza, qm.ini 126
 - ExitPDArea field
 - MQAXP structure 521
 - ExitProperties stanza, mqs.ini 111
 - ExitReason field
 - MQAXP structure 517
 - ExitReason parameter
 - MQXEP call 524
 - ExitResponse field
 - MQAXP structure 518
 - ExitResponse2 field
 - MQAXP structure 519
 - ExitUserArea field
 - MQAXP structure 520
 - extending queue manager facilities 13
 - EXTSHM, using 581
- ## F
- Feedback field
 - MQAXP structure 520
 - FEEDBACK keyword, rules table 198
 - feedback, MQSC commands 38
 - FFST (first-failure support technology)
 - UNIX systems 269
 - Windows NT 267
 - file names 20
 - file sizes, for logs 228
 - File Transfer Application 563
 - files
 - log control file 224
 - log files, in problem determination 256
 - logs 223
 - names 20
 - queue manager configuration 107
 - sizes, for logs 228
 - understanding names 20
 - WebSphere MQ configuration 105
 - XA switch load files 191
 - Filter parameter
 - enumerate authority data call 442
 - first-failure support technology (FFST)
 - See FFST (first-failure support technology)
 - FORMAT keyword, rules table 198
 - FreeParms parameter
 - free user call 445
 - function
 - MQZ_REFRESH_CACHE 466
 - Function field
 - MQAXP structure 522
 - Function parameter
 - MQXEP call 524
 - MQZEP call 421
 - FWDQ keyword, rules table 199
 - FWDQM keyword, rules table 200
- ## G
- generic profiles, OAM 142
 - global units of work
 - adding XAResourceManager stanza to qm.ini, Informix 179
 - adding XAResourceManager stanza to qm.ini, Oracle 177
 - adding XAResourceManager stanza, DB2 174
 - definition of 15, 166
 - groups
 - creating 136
 - managing 136

- groups (*continued*)
 - security 132
- gsk7cmd
 - commands 387
 - options 392
 - preparing 387
- GSS (DCE Generic Security Service)
 - See* DCE Generic Security Service (GSS)
- guidelines for creating queue managers 27

H

- Hconfig field
 - MQAXP structure 522
- Hconfig parameter
 - initialize authorization service call 459
 - initialize name service call 491
 - MQXEP call 524
 - MQZEP call 421
 - terminate authorization service call 474
 - terminate name service call 499
- HEADER keyword, rules table 200
- help with command syntax 280
- HP-UX
 - MQAI support for 62
 - security 138
 - sybswit, creating the Sybase switch load file 181
 - trace 260
 - trace data, sample 261

I

- i5/OS
 - levels supported by the WebSphere MQ Explorer 82
- IBM MQSeries namespace, ADSI support 63
- IdentityContext parameter
 - authenticate user call 424
- ignoring error codes under UNIX systems 258
- ignoring error codes under Windows systems 258
- indirect mode, runmqsc command 72
- indoubt transactions
 - database managers 184
 - display WebSphere MQ transactions (dspmqtrn) command 328
 - using the resolve WebSphere MQ (rsvmqtrn) command 353
- Informix
 - configuration 178
 - database, creation 178
 - environment variable settings, checking 178
 - INFORMIXDIR, environment variable 178
 - ONCONFIG, environment variable 178
 - switch load file, creating 179

- Informix (*continued*)
 - switch load file, creating on UNIX 179
 - switch load file, creating on Windows systems 179
 - XAResourceManager stanza, adding to qm.ini 179
- initialization 404
- initiation queues
 - defining 59
 - description of 8
- input, standard 37
- installable service
 - authorization service 409
 - component
 - authenticate user 423
 - check authority 426
 - check authority (extended) 431
 - copy all authority 436
 - delete authority 439
 - enumerate authority data 442
 - free user 445
 - get authority 447
 - get authority (extended) 450
 - get explicit authority 453
 - get explicit authority (extended) 456
 - initialize authorization service 459
 - initialize name service 491
 - inquire authorization service 462
 - insert name 494
 - lookup name 496
 - MQZ_DELETE_NAME 489
 - MQZEP 421
 - set authority 468
 - set authority (extended) 471
 - terminate authorization service 474
 - terminate name service 499
 - Component data 403
 - component entry-points 403
 - components 402
 - configuring services 404
 - functions 402
 - initialization 404
 - interface to 419
 - multiple components 406
 - name service 415
 - name service interface 416
 - return information 403
- installable services 466
 - authorization service 13
 - definition of 13
 - installable services, list of 13
 - name service 13
 - service component 13
- installing
 - database products 170
- Installing multiple queue managers 29
- IntAttrCount parameter
 - inquire authorization service call 463
- IntAttrs parameter
 - inquire authorization service call 463
- interprocess communication resources 581

- IPC resources
 - clearing WebSphere MQ shared memory resources 581
 - EXTSHM 581
 - shared memory on AIX 581
- issuing
 - MQSC commands remotely 71
 - MQSC commands using an ASCII file 36
 - MQSC commands using runmqsc command 36

J

- Java diagnostics 271
- Java tracing 271

L

- LIKE attribute, DEFINE command 46
- linear logging 225
- Linux
 - security 140
 - trace data, sample 263
- listener
 - end listener (endmqslr) command 333
 - starting 70
 - using the run listener (runmqslr) command 361
- listener objects
 - description of 11
- listeners
 - defining listeners for remote administration 69
- local administration
 - creating a queue manager 26
 - definition of 17
 - issuing MQSC commands using an ASCII file 36
 - runmqsc command, to issue MQSC commands 36
 - support for application programs 35
 - using the WebSphere MQ Explorer 81
 - writing Eclipse plug-ins 93
- local queues 45
 - changing queue attributes, commands to use 47
 - clearing 47
 - copying a local queue definition 46
 - defining 45
 - defining application queues for triggering 58
 - deleting 47
 - description of 9
 - monitoring performance of WebSphere MQ for Windows queues 49
 - specific queues used by WebSphere MQ 8
 - working with local queues 45
- local unit of work
 - definition of 15, 165
- Log stanza, qm.ini 118
- LogDefaults stanza, mqsc.ini 111

- logging
 - calculating the size of logs 228
 - checkpoint records 226
 - checkpoints 225, 226
 - circular logging 224
 - contents of logs 223
 - linear logging 225
 - locations for log files 232
 - log file reuse 226
 - media recovery 233
 - parameters 28
 - types of 224
 - what happens when a disk fills up? 230
- logs
 - calculating the size of logs 228
 - checkpoints 225, 226
 - configuring 118
 - dumping log records (dmpmqlog command) 240
 - dumping the contents of 240
 - error logs 255
 - errors occurring before error log established 257
 - format of a log 223
 - log control file 224
 - log files, in problem determination 256
 - Log stanza, qm.ini 118
 - logging parameters 28
 - managing 229, 230
 - media recovery, linear logs 232
 - oldest required for recovery and restart 350
 - output from the dmpmqlog command 241
 - overheads 228
 - parameters 28
 - persistent messages, effect upon log sizes 228
 - primary log files 224
 - protecting 234
 - recreating objects (rcrmqobj) command 351
 - reuse of 226
 - secondary log files 224
 - types of logging 224
 - types of logs 223
 - using logs for recovery 232
 - what happens when a disk fills up? 230
- LongMCAUserIdLength field
 - MQAXC structure 514
- LongMCAUserIdPtr field
 - MQAXC structure 514
- LongRemoteUserIdLength field
 - MQAXC structure 514
- LongRemoteUserIdPtr field
 - MQAXC structure 514
- LU62 stanza, qm.ini 124

M

- managing objects for triggering 58
- manual removal of a queue manager 560

- manually stopping a queue manager 559
- maximum line length, MQSC commands 39
- MCA (message channel agent) 195
- media images
 - automatic media recovery failure, scenario 240
 - description of 232
 - oldest log required for recovery 350
 - record media image (rcdmqimg) command 349
 - recording media images 233
 - recovering damaged objects during start up 234
 - recovering media images 233
- message channel agent (MCA) 195
- message length, decreasing 47
- message queuing 3
- message-driven processing 3
- message-queuing interface (MQI)
 - See MQI (message-queuing interface)
- messages
 - application data 4
 - containing unexpected information 250
 - converting user-defined message formats 78
 - definition of 3
 - message descriptor 4
 - message length, effects on performance 254
 - message lengths 4
 - message-driven processing 3
 - not appearing on queues 249
 - operator messages 258
 - persistent messages, effect on performance 255
 - persistent messages, when determining log sizes 228
 - queuing 3
 - retrieval algorithms 5
 - retrieving messages from queues 5
 - sending and receiving 4
 - undelivered 258
 - variable length 255
- Microsoft Cluster Server (MSCS)
 - See MSCS (Microsoft Cluster Server)
- Microsoft Transaction Server (MTS)
 - See MTS (Microsoft Transaction Server)
- migrate certificates, amqtcert command 291
- migrating authorization data from MQSeries Version 5.1 410
- model queues
 - creating a model queue 5
 - DEFINE QMODEL command 52
 - defining 52
 - working with 52
- monitoring
 - performance of WebSphere MQ for Windows queues 49
 - start client trigger monitor (runmqtrmc) command 366
 - starting a trigger monitor (runmqtrm command) 367

- MQ_BACK_EXIT call 527
- MQ_BEGIN_EXIT call 528
- MQ_CLOSE_EXIT call 529
- MQ_COMMIT_EXIT call 530
- MQ_CONNX_EXIT call 531
- MQ_DISC_EXIT call 533
- MQ_GET_EXIT call 534
- MQ_INIT_EXIT call 536
- MQ_INQ_EXIT call 537
- MQ_OPEN_EXIT call 539
- MQ_PUT_EXIT call 540
- MQ_PUT1_EXIT call 542
- MQ_SET_EXIT call 544
- MQ_TERM_EXIT call 546
- MQACH structure 509
- MQACH_* values 509
- MQAI (WebSphere MQ administrative interface)
 - description of 62
- MQAXC structure 512
- MQAXC_* values 512
- MQAXP structure 516
- MQAXP_* values 516
- MQDLH, dead-letter header 195
- mqftapp
 - format 338
 - related commands 338
- mqftrcv
 - examples 341
 - format 339
 - parameters 339
 - related commands 341
 - return codes 341
- mqftrvcv
 - examples 344
 - format 342
 - parameters 342
 - related commands 344
 - return codes 344
- mqftsnd
 - examples 346
 - format 345
 - parameters 345
 - related commands 346
 - return codes 345
- mqftsndc
 - examples 348
 - format 347
 - parameters 347
 - related commands 348
 - return codes 347
- MQHCONFIG 422
- MQI (message-queuing interface)
 - authorization specification tables 150
 - authorizations 150
 - definition of 3
 - local administration support 35
 - queue manager calls 10
 - receiving messages 4
 - sending messages 4
- MQI authorizations 150
- mqm group 130
- MQOPEN authorizations 150
- MQOT_* values 480
- MQPUT and MQPUT1, performance considerations 255
- MQPUT authorizations 150

MQS_TRACE_OPTIONS, environment variable 266

mqs.ini configuration file

- AllQueueManagers stanza 109
- ApiExitCommon stanza 115
- ApiExitTemplate 115
- ClientExitPath stanza 110
- DefaultQueueManager stanza 110
- definition of 104
- editing 104
- ExitProperties stanza 111
- LogDefaults stanza 111
- path to 42
- priorities 105
- QueueManager stanza 115

MQSID_* values 513

MQSPREFIX, environment variable 109

MQXACT_* values 520

MQXCC_* values 518

MQXEP call 524

MQXPDA_* values 521

MQXR_* values 517

MQXR2_* values 519

MQXUA_* values 520

MQZ_AUTHENTICATE_USER call 423

MQZ_CHECK_AUTHORITY call 426

MQZ_CHECK_AUTHORITY_2 call 431

MQZ_COPY_ALL_AUTHORITY call 436

MQZ_DELETE_AUTHORITY call 439

MQZ_DELETE_NAME call 489

MQZ_ENUMERATE_AUTHORITY_DATA call 442

MQZ_FREE_USER call 445

MQZ_GET_AUTHORITY call 447

MQZ_GET_AUTHORITY_2 call 450

MQZ_GET_EXPLICIT_AUTHORITY call 453

MQZ_GET_EXPLICIT_AUTHORITY_2 call 456

MQZ_INIT_AUTHORITY call 459

MQZ_INIT_NAME call 491

MQZ_INQUIRE call 462

MQZ_INSERT_NAME call 494

MQZ_LOOKUP_NAME call 496

MQZ_REFRESH_CACHE function 466

MQZ_SET_AUTHORITY call 468

MQZ_SET_AUTHORITY_2 call 471

MQZ_TERM_AUTHORITY call 474

MQZ_TERM_NAME call 499

MQZAC structure 476

MQZAC_* values 476

MQZAD structure 478

MQZAD_* values 479

MQZAET_* values 481

MQZAO_* values 480

MQZAO, constants and authority 151

MQZED structure 483

MQZED_* values 483

MQZEP call 421

MQZFP structure 487

MQZFP_* values 487

MQZIC structure 485

MQZIC_* values 485

MQZSE_* values 442

MSCS (Microsoft Cluster Server) introduction 19

MsgId, performance considerations when using 255

MSGTYPE keyword, rules table 199

MTS (Microsoft Transaction Server) introduction 194

services 194

multiple queue managers, installing 29

multiple service components 406

MUSR_MQADMIN

- changing the password 90
- changing user name 89

N

name service 13, 401

- interface (NSI) 415

name transformations 20

namelists

- description of 10

naming conventions

- national language support 277
- object names 6
- queue manager name transformation 20

national language support

- data conversion 76
- EBCDIC NL character conversion to ASCII 109
- naming conventions for 277
- operator messages 258

nested groups 164

NETBIOS stanza, qm.ini 124

new function xix

NextChainAreaPtr field

- MQACH structure 511

NL character, EBCDIC conversion to ASCII 109

Nonpersistent messages, tuning in AIX 254

NSI (WebSphere MQ name service interface) 415

O

OAM 141

- migrating authorization data from MQSeries Version 5.1 410
- refreshing after changing a user's authorization 409

OAM (Object Authority Manager) authorization service, installable service 13

- overview 14
- using the set and reset authority (setmqaut) command 368

OAM generic profiles 142

object authority manager 409

object authority manager (OAM) 141

Object Authority Manager (OAM)

- See OAM (Object Authority Manager)

object name transformation 21

ObjectName parameter

- check authority (extended) call 432
- check authority call 426
- copy all authority call 436
- delete authority call 439

ObjectName parameter (*continued*)

- get authority (extended) call 450
- get authority call 447
- get explicit authority (extended) call 456
- get explicit authority call 453
- set authority (extended) call 471
- set authority call 468

objects

- access to 129
- administration of 17
- attributes of 6
- automation of administration tasks 18
- default configuration, Windows systems 19
- default object attributes, displaying 46
- description of 10, 11, 66
- display file system name (dspmqfls) command 317
- local queues 9
- managing objects for triggering 58
- media images 233
- multiple queues 10
- name transformation 21
- naming conventions 6, 277
- object name transformation 21
- process definitions 10
- queue manager objects used by MQI calls 10
- queue managers 9
- recovering damaged objects during start up 234
- recovering from media images 233
- recreate (rcrmqobj) command 351
- remote administration 65
- remote queue objects 76
- remote queues 9
- system default objects 12
- types of 5
- using MQSC commands to administer 18

ObjectType field

- MQZAD structure 480

ObjectType parameter

- check authority (extended) call 432
- check authority call 427
- copy all authority call 436
- delete authority call 439
- get authority (extended) call 451
- get authority call 448
- get explicit authority (extended) call 457
- get explicit authority call 454
- set authority (extended) call 472
- set authority call 469

operator

- commands, no response from 251
- messages 258

options

- gsk7cmd 392
- runmqckm 392

Options field

- MQZAD structure 481

- Options parameter
 - initialize authorization service call 459
 - initialize name service call 491
 - terminate authorization service call 474
 - terminate name service call 499
- Oracle
 - configuration parameters, changing 177
 - configuring 176
 - environment variable settings, checking 176
 - ORACLE_HOME, environment variable 176
 - ORACLE_SID, environment variable 176
 - security considerations 183
 - switch load file, creating 176
 - switch load file, creating on UNIX 176
 - switch load file, creating on Windows systems 176
 - XAResourceManager stanza, adding to qm.ini 177
- output, standard 37
- overheads, for logs 228

P

- pBufferLength parameter
 - MQ_GET_EXIT call 534
 - MQ_PUT_EXIT call 540
 - MQ_PUT1_EXIT call 542
- PCF (programmable command format)
 - Active Directory Service Interfaces (ADSI) 63
 - administration tasks 18
 - attributes in MQSC commands and PCF 62
 - authorization specification tables 150
 - automating administrative tasks using PCF 61
 - escape PCFs 62
 - MQAI, using to simplify use of 62
 - object attribute names 6
- pCharAttrLength parameter
 - MQ_INQ_EXIT call 537
 - MQ_SET_EXIT call 544
- pCompCode parameter
 - MQ_BACK_EXIT call 527
 - MQ_BEGIN_EXIT call 528
 - MQ_CLOSE_EXIT call 529
 - MQ_CONNX_EXIT call 531
 - MQ_DISC_EXIT call 533
 - MQ_INIT_EXIT call 536
 - MQ_INQ_EXIT call 537
 - MQ_OPEN_EXIT call 539
 - MQ_PUT_EXIT call 540
 - MQ_PUT1_EXIT call 542
 - MQ_SET_EXIT call 544
 - MQ_TERM_EXIT call 546
 - MQXEP call 525
- performance
 - advantages of using MQPUT1 255
 - application design, impact on 254
 - CorrelId, effect on 255
 - performance (*continued*)
 - message length, effects on 254
 - message persistence, effect on 255
 - MsgId, effect on 255
 - nonpersistent messages in AIX 254
 - Performance Monitor 49
 - syncpoints, effects on 255
 - threads, effect on 255
 - trace 260, 265
 - tracing Windows, performance considerations 259
 - Performance Monitor 49
 - permanent (predefined) queues 5
 - PERSIST keyword, rules table 199
 - persistent messages, effect on performance 255
 - pExitContext parameter
 - MQ_GET_EXIT call 534
 - MQ_INIT_EXIT call 527, 528, 529, 530, 533, 536, 537
 - MQ_OPEN_EXIT call 539
 - MQ_PUT_EXIT call 540
 - MQ_PUT1_EXIT call 542
 - MQ_SET_EXIT call 544
 - MQ_TERM_EXIT call 546
 - pExitParms parameter
 - MQ_GET_EXIT call 534
 - MQ_INIT_EXIT call 527, 528, 529, 530, 531, 533, 536, 537
 - MQ_OPEN_EXIT call 539
 - MQ_PUT_EXIT call 540
 - MQ_PUT1_EXIT call 542
 - MQ_SET_EXIT call 544
 - MQ_TERM_EXIT call 546
 - pHconn parameter
 - MQ_BACK_EXIT call 527
 - MQ_BEGIN_EXIT call 528
 - MQ_CLOSE_EXIT call 529
 - MQ_CMIT_EXIT call 530
 - MQ_GET_EXIT call 534
 - MQ_INQ_EXIT call 537
 - MQ_OPEN_EXIT call 539
 - MQ_PUT_EXIT call 540
 - MQ_PUT1_EXIT call 542
 - MQ_SET_EXIT call 544
 - pHobj parameter
 - MQ_GET_EXIT call 534
 - MQ_INQ_EXIT call 537
 - MQ_PUT_EXIT call 540
 - MQ_SET_EXIT call 544
 - pIntAttrCount parameter
 - MQ_INQ_EXIT call 537
 - MQ_SET_EXIT call 544
- PKCS #11
 - devices, gsk7cmd commands for 387
- plug-ins
 - enabling and disabling 96
 - writing 94
- PMQFUNC 422
- pOptions parameter
 - MQ_CLOSE_EXIT call 529
 - MQ_OPEN_EXIT call 539
- ppBeginOptions parameter 528
- ppBuffer parameter
 - MQ_GET_EXIT call 534
 - MQ_PUT_EXIT call 540
 - MQ_PUT1_EXIT call 542
- ppCharAttrs parameter
 - MQ_INQ_EXIT call 537
 - MQ_SET_EXIT call 544
- ppConnectOpts parameter 531
- ppDataLength parameter
 - MQ_GET_EXIT call 534
- ppGetMsgOpts parameter 534
- ppHconn parameter
 - MQ_CONNX_EXIT call 531
 - MQ_DISC_EXIT call 533
- ppHobj parameter
 - MQ_CLOSE_EXIT call 529
 - MQ_OPEN_EXIT call 539
- ppIntAttrs parameter
 - MQ_INQ_EXIT call 537
 - MQ_SET_EXIT call 544
- ppMsgDesc parameter
 - MQ_GET_EXIT call 534
 - MQ_PUT_EXIT call 540
 - MQ_PUT1_EXIT call 542
- ppObjDesc parameter
 - MQ_OPEN_EXIT call 539
 - MQ_PUT1_EXIT call 542
- ppPutMsgOpts parameter
 - MQ_PUT_EXIT call 540
 - MQ_PUT1_EXIT call 542
- ppSelectors parameter
 - MQ_INQ_EXIT call 537
 - MQ_SET_EXIT call 544
- pQMgrName parameter
 - MQ_CONNX_EXIT call 531
- pReason parameter
 - MQ_BACK_EXIT call 527
 - MQ_BEGIN_EXIT call 528
 - MQ_CLOSE_EXIT call 529
 - MQ_CMIT_EXIT call 530
 - MQ_CONNX_EXIT call 531
 - MQ_DISC_EXIT call 533
 - MQ_GET_EXIT call 534
 - MQ_INIT_EXIT call 536
 - MQ_INQ_EXIT call 537
 - MQ_OPEN_EXIT call 539
 - MQ_PUT_EXIT call 540
 - MQ_PUT1_EXIT call 542
 - MQ_SET_EXIT call 544
 - MQ_TERM_EXIT call 546
 - MQXEP call 526
- predefined (permanent) queues 5
- preemptive shutdown of a queue manager 32
- preparing
 - gsk7cmd 387
- primary initialization 404
- primary termination 404
- principals 132
- problem determination
 - application design considerations 254
 - applications or systems running slowly 253
 - clients 271
 - command errors 248
 - common programming errors 247
 - configuration files 259
 - has the application run successfully before? 246
 - incorrect output, definition of 249

- problem determination (*continued*)
 - incorrect output, distributed queuing 251
 - intermittent problems 248
 - introduction 245
 - log files 256
 - no response from operator
 - commands 251
 - preliminary checks 245
 - problems affecting parts of a network 248
 - problems caused by service updates 248
 - problems that occur at specific times in the day 248
 - problems with shutdown 32
 - questions to ask 245
 - queue failures, problems caused by 252
 - remote queues, problems
 - affecting 253
 - reproducing the problem 246
 - return codes 246, 247
 - searching for messages, performance effects 255
 - things to check first 245
 - trace 260, 265
 - undelivered messages 258
 - WebSphere MQ error messages 246
 - what is different since the last successful run? 246
- process definitions
 - creating 60
 - description of 10
 - displaying 60
 - process commands 578
- ProcessId field
 - MQAXC structure 515
 - MQZAC structure 477
- processing, message-driven 3
- ProfileName field
 - MQZAD structure 479
- profiles, OAM generic 142
- programmable command format (PCF)
 - See PCF (programmable command format)
- programming errors, examples of 247
 - further checks 249, 254
 - secondary checks 249, 254
- pSelectorCount parameter
 - MQ_INQ_EXIT call 537
 - MQ_SET_EXIT call 544
- PUTAUT keyword, rules table 200

Q

- qm.ini configuration file
 - ApiExitLocal stanza 127
 - CHANNELS stanza 122
 - definition of 107
 - editing 104
 - ExitPath stanza 126
 - Log stanza 118
 - LU62 stanza 124
 - NETBIOS stanza 124
 - priorities 105
 - RestrictedMode stanza 120

- qm.ini configuration file (*continued*)
 - Service stanza 116
 - ServiceComponent stanza 117
 - SPX stanza 124
 - TCP stanza 124
 - XAResourceManager stanza 121
- QMGrName field
 - MQAXP structure 521
- QMGrName parameter
 - authenticate user call 423
 - check authority (extended) call 431
 - check authority call 426
 - copy all authority call 436
 - delete authority call 439
 - enumerate authority data call 442
 - free user call 445
 - get authority (extended) call 450
 - get authority call 447
 - get explicit authority (extended) call 456
 - get explicit authority call 453
 - initialize authorization service call 459
 - initialize name service call 491
 - inquire authorization service call 462
 - insert name call 494
 - lookup name call 496
 - MQZ_DELETE_NAME call 489
 - set authority (extended) call 471
 - set authority call 468
 - terminate authorization service call 474
 - terminate name service call 499
- QName parameter
 - insert name call 494
 - lookup name call 496
 - MQZ_DELETE_NAME call 489
- queue browser, sample 48
- queue depth, current 46
- queue manager
 - ini file
 - authorization service 409
- queue manager ini file 409
- queue managers
 - accidental deletion of default 301
 - activating a backup queue manager 238
 - attributes, changing 44
 - attributes, displaying 43
 - backing up queue manager data 235
 - CCSID, changing 78
 - changing the CCSID 78
 - changing the default queue manager 30
 - checking certificates chains (amqccert) command 283
 - command server 70
 - configuration files, backing up 30
 - configuration information 103
 - controlled shutdown 31
 - creating a default queue manager 29
 - creating a queue manager 26, 299
 - default configuration, Windows systems 19
 - default for each node 27
 - deleting a queue manager 33

- queue managers (*continued*)
 - deleting a queue manager (dlmqm) command 303
 - description of 9
 - display queue managers (dsmpmq) command 311
 - dumping formatted system log (dmpmqlog) command 309
 - dumping the contents of a recovery log 240
 - end queue manager (endmqm) command 335
 - extending queue manager facilities 13
 - guidelines for creating a queue manager 27
 - immediate shutdown 32
 - limiting the numbers of 27
 - linear logging 225
 - log maintenance, recovery 223
 - name transformation 20
 - objects used in MQI calls 10
 - oldest log required to restart 350
 - preemptive shutdown 32
 - preparing for remote administration 67
 - qm.ini files 107
 - queue manager aliases 76
 - queue manager commands 575
 - quiesced shutdown 32
 - recording media images 233
 - remote administration 65
 - removing a queue manager manually 560
 - restoring queue manager data 235, 236
 - reverting to the original default 30
 - showing and hiding, using the WebSphere MQ Explorer 86
 - specifying unique names for 27
 - starting a queue manager 31
 - starting a queue manager automatically 31
 - starting a queue manager, strmqm command 381
 - stopping a queue manager 31
 - stopping a queue manager manually 559
 - transferring certificates (amqtcert) command 291
 - WebSphere MQ services control (amqmdain) command 285
 - z/OS queue manager 72
- QueueManager stanza, mqz.ini 115
- queues
 - alias 50
 - application queues 58
 - attributes 7
 - browsing 48
 - changing queue attributes 47
 - clearing local queues 47
 - current queue depth, determining 46
 - dead-letter, defining 45
 - defaults, transmission queues 28
 - defining WebSphere MQ queues 6
 - definition of 4
 - deleting a local queue 47

- queues (*continued*)
 - distributed, incorrect output from 251
 - dynamic (temporary) queues 5
 - extending queue manager facilities 13
 - for MQSeries applications 35
 - initiation queues 59
 - local definition of a remote queue 73
 - local queues 9
 - local, working with 45
 - model queues 5, 52
 - multiple queues 10
 - predefined (permanent) queues 5
 - preparing transmission queues for remote administration 68
 - queue commands 579
 - queue manager aliases 76
 - queue managers, description of 9
 - remote queue objects 76
 - reply-to queues 76
 - retrieving messages from 5
 - specific local queues used by WebSphere MQ 8
 - specifying dead-letter queues 28
 - specifying undelivered-message 28
- quiesced shutdown of a queue manager 32
- preemptive shutdown 32

R

- rcdmqimg (record media image) command
 - examples 350
 - format 349
 - parameters 349
 - purpose 349
 - related commands 350
 - return codes 350
- rcrmqobj (recreate object) command
 - examples 352
 - format 351
 - parameters 351
 - purpose 351
 - related commands 352
 - return codes 352
- REASON keyword, rules table 199
- Reason parameter
 - authenticate user call 424
 - check authority (extended) call 435
 - check authority call 429
 - copy all authority call 437
 - delete authority call 440
 - enumerate authority data call 444
 - free user call 446
 - get authority (extended) call 452
 - get authority call 449
 - get explicit authority (extended) call 458
 - get explicit authority call 455
 - initialize authorization service call 460
 - initialize name service call 492
 - inquire authorization service call 464
 - insert name call 495
 - lookup name call 497

- Reason parameter (*continued*)
 - MQZ_DELETE_NAME call 490
 - MQZEP call 421
 - set authority (extended) call 473
 - set authority call 470
 - terminate authorization service call 475
 - terminate name service call 500
- receiver channel, automatic definition of 70
- receiving a file 563
- record media image command (rcdmqimg)
 - See* rcdmqimg (record media image) command
- recovery
 - activating a backup queue manager 238
 - automatic media recovery failure, scenario 240
 - backing up queue manager data 235
 - backing up WebSphere MQ 235
 - checkpoints, recovery logs 226
 - disk drive failure, scenario 239
 - making sure messages are not lost using logs 223
 - media images, recovering 232, 233
 - recovering a damaged queue manager object, scenario 240
 - recovering a damaged single object, scenario 240
 - recovering damaged objects at other times 234
 - recovering damaged objects during start up 234
 - recovering from problems 232
 - restoring queue manager data 236
 - scenarios 239
 - using the log for recovery 232
- recreate object command (rcrmqobj)
 - See* rcrmqobj (recreate object) command
- redirecting input and output, MQSC commands 38, 41
- RefObjectName parameter
 - copy all authority call 436
- refreshing the OAM after changing a user's authorization 409
- remote administration
 - administering a remote queue manager from a local one 67
 - command server 70
 - defining channels, listeners, and transmission queues 69
 - definition of remote administration 17
 - initial problems 72
 - of objects 65
 - preparing channels for 68
 - preparing queue managers for 67
 - preparing transmission queues for security, connecting remote queue managers, the WebSphere MQ Explorer 85
 - using the WebSphere MQ Explorer 81
 - writing Eclipse plug-ins 93

- remote issuing of MQSC commands 71
- remote queue objects 76
- remote queues
 - as reply-to queue aliases 76
 - defining remote queues 73
 - recommendations for remote queuing 72
- remote queuing 65
- removing a queue manager manually 560
- reply-to queue aliases 76
- reply-to queues
 - description of 9
 - reply-to queue aliases 76
- REPLYQ keyword, rules table 199
- REPLYQM keyword, rules table 199
- Reserved field
 - MQZFP structure 487
- Reserved parameter
 - MQXEP call 525
- resolve WebSphere MQ transactions command (rsvmqtrn)
 - See* rsvmqtrn (resolve WebSphere MQ transactions) command
- ResolvedQMgrName parameter
 - insert name call 494
 - lookup name call 496
- resources
 - updating under syncpoint control 14
- resources, IPC 581
- restarting a queue manager 33
 - oldest logs required 350
- restoring queue manager data 235
- RestrictedMode stanza, qm.ini 120
- restrictions
 - access to MQM objects 129
 - database coordination support 168
 - on object names 277
- retrieval algorithms for messages 5
- RETRY keyword, rules table 200
- RETRYINT keyword, rules tables 197
- return codes
 - amqccert command 284
 - amqmdain command 289
 - amqtcert command 295
 - crtmqcvx command 297
 - crtmqm command 302
 - dltmqm command 303
 - dspmq command 311
 - dspmqcvs command 316
 - dspmqls command 318
 - dspmqrte command 326
 - dspmqrtn command 328
 - dspmqrver command 329
 - endmqcvs command 331
 - endmqslr command 333
 - endmqm command 336
 - endmqtrc command 337
 - mqftrvc command 341
 - mqftrvcv command 344
 - mqftsnd command 346
 - mqftsndc command 348
 - problem determination 247
 - rcdmqimg command 350
 - rcrmqobj command 352
 - rsvmqtrn command 353
 - runmqchi command 355

- return codes (*continued*)
 - runmqchl command 356
 - runmqdnm command 334, 360
 - runmqslr command 362
 - runmqsc command 364
 - runmqtmc command 366
 - runmqtrm command 367
 - setmqaut command 373
 - strmqcsv command 380
 - strmqm command 382
 - strmqtrc command 385
- rsvmqtrn (resolve WebSphere MQ transactions) command
 - format 353
 - parameters 353
 - purpose 353
 - related commands 354
 - return codes 353
- rules table (DLQ handler)
 - ACTION keyword 199
 - action keywords 199
 - APPLIDAT keyword 198
 - APPLNAME keyword 198
 - APPLTYPE keyword 198
 - control-data entry 196
 - conventions 200
 - description of 196
 - DESTQ keyword 198
 - DESTQM keyword 198
 - example of a rules table 204
 - FEEDBACK keyword 198
 - FORMAT keyword 198
 - FWDQ keyword 199
 - FWDQM keyword 200
 - HEADER keyword 200
 - INPUTQ keyword 196
 - INPUTQM keyword 197
 - MSGTYPE keyword 199
 - pattern-matching keywords 198
 - patterns and actions 198
 - PERSIST keyword 199
 - processing rules 202
 - PUTAUT keyword 200
 - REASON keyword 199
 - REPLYQ keyword 199
 - REPLYQM keyword 199
 - RETRY keyword 200
 - RETRYINT keyword 197
 - syntax rules 201
 - USERID keyword 199
 - WAIT keyword 197
- run channel command (runmqchl)
 - See runmqchl (run channel) command
- run channel initiator command (runmqchi)
 - See runmqchi (run channel initiator) command
- run dead-letter queue handler command (runmqdlq)
 - See runmqdlq (run DLQ handler) command
- run listener command (runmqslr)
 - See runmqslr (run listener) command
- run WebSphere MQ command (runmqsc)
 - See runmqsc (run WebSphere MQ commands) command

- runmqchi (run channel initiator) command
 - format 355
 - parameters 355
 - purpose 355
 - return codes 355
- runmqchl (run channel) command
 - format 356
 - parameters 356
 - purpose 356
 - return codes 356
- runmqckm
 - commands 387
 - options 392
- runmqdlq (run DLQ handler) command
 - format 357
 - parameters 357
 - purpose 357
 - run DLQ handler (runmqdlq) command 195
 - usage 357
- runmqdnm
 - format 358
 - parameters 358
 - return codes 334, 360
- runmqslr (run listener) command
 - example 362
 - format 361
 - parameters 361
 - purpose 361
 - return codes 362
- runmqsc (run WebSphere MQ commands) command
 - ending 38
 - examples 364
 - feedback 38
 - format 363
 - indirect mode 72
 - parameters 363
 - problems, resolving 42
 - purpose 363
 - redirecting input and output 38, 41
 - return codes 364
 - usage 363
 - using 38, 41
 - using interactively 37
 - verifying 41
- runmqtmc (start client trigger monitor) command
 - examples 366
 - format 366
 - parameters 366
 - purpose 366
 - return codes 366
- runmqtrm (start trigger monitor) command
 - format 367
 - parameters 367
 - purpose 367
 - return codes 367

S

- samples
 - trace data (AIX) 264
 - trace data (HP-UX) 261
 - trace data (Linux) 263

- samples (*continued*)
 - trace data (Solaris) 262
 - Windows trace data, sample 260
- secondary initialization 404
- secondary termination 404
- secure sockets layer (SSL)
 - channel parameters 149
 - MQSC commands 148
 - overview 14
 - protecting channels 148
 - queue manager parameters 148
- security
 - access control 131, 141
 - access settings 144, 145
 - administration authority 129
 - AIX 139
 - alternate-user authority 134
 - authority, alternate-user 134
 - authority, context 134
 - authorizations to use the WebSphere MQ Explorer 84
 - channel exits 148
 - channel security 14
 - channels 146
 - checks 131
 - checks, preventing 146
 - connecting to remote queue managers, the WebSphere MQ Explorer 85
 - considerations for transactional support 183
 - context authority 134
 - dmpmqaut command 144
 - domain controller 162
 - dspmqaout command 145
 - groups 132, 136
 - HP-UX 138
 - identifiers 133
 - Linux 140
 - MQI authorizations 150
 - mqm group 130
 - name service security, overview 14
 - nested groups 164
 - OAM 14, 141
 - object authority manager (OAM) 14, 141
 - principals 132
 - protecting log files 234
 - restoring queue manager data 235
 - security for the WebSphere MQ Explorer 84, 87
 - SecurityPolicy attribute, Service stanza 116
 - setmqaut command 142
 - Solaris 139
 - SSL 14
 - template files 163
 - transmission queues 147
 - user ID 132
 - using the set and reset authority (setmqaut) command 368
 - WebSphere MQ objects 130
 - WebSphere MQ Services 163
 - Windows 2000 136, 161
 - Windows 2003 136, 161
 - Windows systems 133
 - Windows XP 136
- security enabling interface (SEI) 409

- SecurityId field
 - MQAXC structure 513
 - MQZED structure 484
- SecurityParms parameter
 - authenticate user call 423
- SEI (WebSphere MQ security enabling interface) 409
- SelectorCount parameter
 - inquire authorization service call 462
- SelectorReturned parameter
 - inquire authorization service call 463
- Selectors parameter
 - inquire authorization service call 462
- sending a file 563
- server-connection channel, automatic
 - definition of 70
- servers 12
 - See clients and servers
- service component 13
 - authorization 409
 - creating your own 406
 - multiple 406
 - stanza 405
- service objects
 - description of 11
- service stanza 404
- Service stanza, qm.ini 116
- ServiceComponent stanza, qm.ini 117
- services 53
- Services snap-in
 - WebSphere MQ services control (amqmdain) command 285
- set CRL server definitions (setmqcrl)
 - See setmqcrl (set CRL server definitions) command
- set service connection points command (setmqscp)
 - See setmqscp (set service connection points) command
- Set WebSphere MQ CRL definitions 375
- Set WebSphere MQ Service Connection Points 377
- set/reset authority command (setmquat)
 - See setmquat (set/reset authority) command
- setmqcrl (set CRL server definitions) command
 - purpose 375
- setmqscp (set service connection points) command
 - examples 375, 377
 - format 375, 377
 - purpose 377
- setmquat (set/reset authority) command
 - examples 326, 373
 - format 368
 - parameters 320, 370
 - purpose 319, 368
 - related commands 374
 - return codes 326, 373
 - usage 369
- shared memory on AIX 581
- shared memory resources, clearing
 - WebSphere MQ 581
- shell commands, WebSphere MQ for UNIX systems 26
- shutting down a queue manager 31
 - shutting down a queue manager (continued)
 - a queue manager, quiesced 32
 - controlled 31
 - immediate 32
 - preemptive 32
 - SIDs (security identifiers) 133
 - Solaris
 - MQAI support for 62
 - security 139
 - sybswit, creating the Sybase switch load file 181
 - trace 260
 - trace data, sample 262
 - specifying coded character sets 77
 - SPX stanza, qm.ini 124
 - SSL
 - amqccert command 283
 - amqtcert command 291
 - stanza
 - authorization service, UNIX systems 410
 - authorization service, Windows 412
 - stanzas
 - AllQueueManagers, mqs.ini 109
 - ApiExitCommon, mqs.ini 115
 - ApiExitLocal, qm.ini 127
 - ApiExitTemplate, mqs.ini 115
 - CHANNELS, qm.ini 122
 - CICS XAD resource definition stanza 193
 - ClientExitPath, mqs.ini 110
 - DefaultQueueManager, mqs.ini 110
 - ExitPath, qm.ini 126
 - ExitProperties, mqs.ini 111
 - Log, qm.ini 118
 - LogDefaults, mqs.ini 111
 - LU62, qm.ini 124
 - NETBIOS, qm.ini 124
 - QueueManager, mqs.ini 115
 - RestrictedMode stanza, qm.ini 120
 - Service, qm.ini 116
 - ServiceComponent, qm.ini 117
 - SPX, qm.ini 124
 - TCP, qm.ini 124
 - XAResourceManager, qm.ini 121
 - start client trigger monitor command (runmqtrm)
 - See runmqtrm (start client trigger monitor) command
 - start command server command (strmqscsv)
 - See strmqscsv (start command server) command
 - start queue manager command (strmqm)
 - See strmqm (start queue manager) command
 - start trigger monitor command (runmqtrm)
 - See runmqtrm (start trigger monitor) command
 - start WebSphere MQ trace command (strmqtrc)
 - See strmqtrc (start WebSphere MQ trace) command
 - StartEnumeration parameter
 - enumerate authority data call 442
- starting
 - a channel 70
 - a command server 71
 - a listener 70
 - a queue manager 31
 - a queue manager automatically 31
- stdin, on runmqsc 38
- stdout, on runmqsc 38
- stopping
 - a queue manager manually 559
 - command server 71
- strmqcfcfg
 - format 379
- strmqscsv (start command server)
 - command
 - examples 380
 - format 380
 - parameters 380
 - purpose 380
 - related commands 380
 - return codes 380
- strmqm (start queue manager) command
 - examples 382
 - format 381
 - parameters 381
 - purpose 329, 381
 - related commands 382
 - return codes 329, 382
- strmqm control command 33
- strmqtrc (start WebSphere MQ trace)
 - command
 - examples 386
 - format 383
 - parameters 384
 - purpose 338, 379, 383
 - related commands 386
 - return codes 385
 - usage 383
- Strucld field
 - MQACH structure 509
 - MQAXC structure 512
 - MQAXP structure 516
 - MQZAC structure 476
 - MQZAD structure 479
 - MQZED structure 483
 - MQZFP structure 487
 - MQZIC structure 485
- StructLength field
 - MQACH structure 510
- switch load files
 - introduction 169
- switch load files, creating 170
- Sybase
 - configuring 180
 - environment variable settings, checking 180
 - security considerations 183
 - switch load file, creating 181
 - Sybase XA support, enabling 180
 - SYBASE_OCS, environment variable 180
 - SYBASE, environment variable 180
 - sybswit, creating the switch load file on UNIX 181
 - sybswit, creating the switch load file on Windows systems 181

Sybase (*continued*)
 XAResourceManager stanza,
 adding 181
 syncpoint coordination 190
 WebSphere MQ 191
 syncpoint, performance
 considerations 255
 syntax, help with 280
 system default objects 12
 system objects 549

T

task termination exit, CICS 193
 TCP stanza, qm.ini 124
 template files, security 163
 temporary (dynamic) queues 5
 termination 404
 ThreadId field
 MQAXC structure 515
 ThreadID field
 MQZAC structure 477
 time-independent applications 3
 timed out responses from MQSC
 commands 72
 trace
 AIX 260
 data sample (AIX) 264, 267
 data sample (HP-UX) 261
 data sample (Linux) 263
 data sample (Solaris) 262
 data sample (Windows) 260
 display WebSphere MQ formatted
 trace (dspmqtrc) command 327
 HP-UX 260
 performance considerations 260, 265
 Solaris 260
 starting WebSphere MQ trace
 (strmqtrc command) 383
 Windows, performance
 considerations 259
 tracing
 Java 271
 transactional support
 syncpoint coordination 190
 transactional support 165
 updating under syncpoint control 14
 WebSphere MQ XA switch
 structure 191
 transactions
 display WebSphere MQ transactions
 (dspmqtrn) command 328
 security considerations 183
 using the resolve WebSphere MQ
 (rsvmqtrn command) 353
 transfer certificates, amqtcert
 command 291
 transferring a file as an MQ
 message 563
 transmission queues
 cluster transmission queues 8
 creating 75
 default 28
 default transmission queues 75
 defining transmission queues remote
 administration 69
 description of 8

transmission queues (*continued*)
 preparing transmission queues for
 remote administration 68
 security 147
 triggering
 defining an application queue for
 triggering 58
 managing objects for triggering 58
 message-driven processing 3
 start client trigger monitor
 (runmqtrm) command 366
 start trigger monitor (runmqtrm)
 command 367
 Tuning nonpersistent messages in
 AIX 254
 two-phase commit process, CICS 192
 types of logging 224

U

undelivered message queue
See dead-letter queues
 units of work
 explicit resynchronization of
 (rsvmqtrn command) 185
 global 166
 introduction 165
 local 165
 mixed outcomes 186
 UNIX
 IPC resources 581
 UNIX operating system
 DB2 switch load file, creating 174
 directory structure 555
 example mq5.ini file 105
 example qm.ini file 107
 Informix switch load file,
 creating 179
 issuing control commands 26
 levels supported by the WebSphere
 MQ Explorer 82
 object authority manager (OAM) 14
 Oracle switch load file, creating 176
 queue managers, deleting 561
 switch load structures, library
 names 191
 sybswit, creating the Sybase switch
 load file 181
 UNIX, definition of term xvi
 updating coded character sets 77
 user exits
 channel exits 13
 CICS task termination exit,
 UE014015 193
 data conversion exits 13
 enabling CICS user exits 193
 user ID 132
 user-defined message formats 78
 UserId field
 MQAXC structure 513
 UserID field
 MQZAC structure 477
 USERID keyword, rules table 199
 UserIdentifier field
 MQZIC structure 486
 using EXTSHM 581

V

verifying MQSC commands 41
 Version field
 MQACH structure 510
 MQAXC structure 512
 MQAXP structure 516
 MQZAC structure 476
 MQZAD structure 479
 MQZED structure 483
 MQZFP structure 487
 MQZIC structure 485
 Version parameter
 initialize authorization service
 call 460
 initialize name service call 492

W

WAIT keyword, rules table 197
 WebSphere MQ 329
 attributes of MQSC commands 62
 commands 36
 configuration information 103
 issuing MQSC commands using an
 ASCII file 36
 name service interface (NSI) 415
 runmqsc command, to issue MQSC
 commands 36
 security enabling interface (SEI) 409
 WebSphere MQ administrative interface
 (MQAI)
 See MQAI (WebSphere MQ
 administrative interface)
 WebSphere MQ alert monitor, using 87
 WebSphere MQ command files
 input 39
 output reports 40
 running 40
 WebSphere MQ commands
 See also WebSphere MQ commands
 attributes of 62
 authorization 153
 command files, input 39
 command files, output reports 40
 command files, running 40
 ending interactive input 38
 escape PCFs 62
 issuing interactively 37
 issuing MQSC commands
 remotely 71
 maximum line length 39
 object attribute names 6
 overview 18, 36
 problems using MQSC commands
 remotely 72
 problems, list 42
 problems, resolving 42
 redirecting input and output 38, 41
 runmqsc control command,
 modes 18, 36
 syntax errors 38
 timed out command responses 72
 using 38, 41
 verifying 41
 WebSphere MQ Explorer
 alert monitor application 87

- WebSphere MQ Explorer (*continued*)
 - AMQMSRVN
 - changing the password 90
 - authorizations to use 84
 - cluster membership 83
 - connecting to remote queue managers, security 85
 - controlling access 89
 - data conversion 86
 - DCOMCNFG.EXE, using 89
 - description of 19
 - introduction 18
 - MUSR_MQADMIN
 - changing the password 90
 - changing the user name 89
 - performance considerations 82
 - Prepare WebSphere MQ Wizard 87, 89
 - prerequisite software 83
 - required resource definitions 83
 - security exits, the WebSphere MQ Explorer 85
 - security exits, using 85
 - security implications 84, 87
 - showing and hiding queue managers and clusters 86
 - SSL security, the WebSphere MQ Explorer 85
 - SSL security, using 85
 - user rights for AMQMSRVN 88
 - using 87
- WebSphere MQ queues, defining 6
- WebSphere MQ Services
 - security 163
- WebSphere MQ Services control (amqmdain)
 - See* amqmdain (WebSphere MQ Services control) command
- WebSphere MQ Services snap-in AMQMSRVN
 - changing the password 90
 - controlling access 89
 - DCOMCNFG.EXE, using 89
 - MUSR_MQADMIN
 - changing the password 90
 - security implications 87
 - WebSphere MQ services control (amqmdain) command 285
- WebSphere MQ Taskbar application
 - using 87
- What's new for this release xix
- Windows 2000
 - security 136
- Windows 2000, security 161
- Windows 2003
 - security 136
- Windows 2003, security 161
- Windows operating system
 - adding a queue manager to 31
 - adding XAResourceManager
 - information for DB2 174
 - control commands for 25
 - db2swit.dll, creating 174
 - default configuration 19
 - default configuration objects, list of 551
 - deleting queue managers 560

- Windows operating system (*continued*)
 - deletions from automatic startup
 - list 561
 - directory structure 553
 - editing configuration
 - information 103
 - Event Viewer application, problem
 - determination 256
 - FFST, examining 267
 - Informix switch load file,
 - creating 179
 - levels supported by the WebSphere MQ Explorer 82
 - MQAI support for 62
 - object authority manager (OAM) 14
 - oraswit.dll, creating 176
 - Performance Monitor 49
 - registry 103
 - security 133
 - SecurityPolicy attribute, Service stanza 116
 - switch load structures, library
 - names 191
 - sybswit, creating the Sybase switch load file 181
 - tracing, considerations 259
 - using the WebSphere MQ Explorer 81
 - viewing configuration
 - information 103
 - Windows trace data, sample 260
 - writing Eclipse plug-ins 93
- Windows Registry
 - deleting queue managers in
 - Windows 560
 - deletions from automatic startup
 - list 561
 - description of 103
 - using in problem determination 256
- Windows XP
 - security 136
- windows, definition of term xvi
- working with 53

X

- XA switch load files
 - description of 191
- XAD resource definition stanza, CICS 193
- XAResourceManager stanza, qm.ini 121

Z

- z/OS
 - levels supported by the WebSphere MQ Explorer 82
- z/OS queue manager 72

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.



SC34-6584-00



Spine information:



WebSphere MQ

System Administration Guide

Version 6.0