Note!

Before using this information and the product it supports, be sure to read the general information under "Notices," on page 485.
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 ........................... xv
How to use this book ................................. xv

What’s new in WebSphere MQ for HP NonStop Server, V5.3 ............... xvi

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
WebSphere MQ objects .................................................... 5
  Object names ......................................................... 5
  Managing objects ...................................................... 6
  Object attributes ...................................................... 6
  WebSphere MQ queues .................................................. 6
WebSphere MQ queue managers ........................................... 9
  Process definitions .................................................... 10
Channels ..................................................... 10
  Queue manager clusters ............................................... 10
Namelists .................................................... 10
Authentication information objects ..................................... 11
System and default objects ............................................. 11
Clients and servers ............................................... 11
  WebSphere MQ applications in a client/server environment ......... 11
Extending queue manager facilities ....................................... 12
  User exits ......................................................... 12
  Installable services .................................................. 12
Security .................................................... 13
  Object Authority Manager (OAM) facility .......................... 13
  Channel security using SSL ........................................... 13
Transactional support ............................................... 13

Chapter 2. An introduction to WebSphere MQ administration ............... 15
Local and remote administration ......................................... 15
Performing administration tasks using commands ....................... 15
  Control commands ................................................... 15
  WebSphere MQ Script (MQSC) commands .......................... 16
  PCF commands ....................................................... 16
Administration on WebSphere MQ for HP NonStop Server ............... 16
  The Pathway control program (PATHCOM) ......................... 17
  The process management rules configuration file ................... 17
  The TMF configuration utility (TMFCOM) .......................... 17
  The Subsystem Control Facility (SCF) .............................. 17
Understanding WebSphere MQ file names ................................ 18
  Transforming a queue manager name ................................ 18
  Transforming the name of a WebSphere MQ object ................ 19

Part 2. Administering WebSphere MQ for HP NonStop Server ............... 21

Chapter 3. Administering queue managers ........................................... 23
Using control commands .................................................. 23
  Using control commands on NonStop OS ............................ 23
Creating a queue manager ................................................ 24
  Guidelines for creating queue managers ............................ 24
  Creating a default queue manager ................................... 27
Making an existing queue manager the default ......................... 27
Back ing up configuration files after creating a queue manager ....... 28
  Creating entries in the principal database of a queue manager .... 28
Starting a queue manager ............................................... 29
Stopping a queue manager .............................................. 29
  Quesed shutdown .................................................... 29
  Immediate shutdown .................................................. 29
  Preemptive shutdown .................................................. 30
Deleting a queue manager .............................................. 30

Chapter 4. Administering local WebSphere MQ objects ......................... 31
Supporting application programs that use the MQI ....................... 31
  Performing local administration tasks using MQSC commands ....... 32
  WebSphere MQ object names ........................................ 32
  Standard input and output ........................................... 33
  Running MQSC commands interactively .............................. 33
  Running MQSC commands from text files .......................... 35
  Resolving problems with MQSC commands ........................... 37
Working with queue managers .......................................... 39
  Displaying queue manager attributes ................................ 39
  Altering queue manager attributes ................................... 40
Working with local queues ............................................ 40
  Defining a local queue ............................................... 40
  Displaying the attributes of a queue ............................... 41
  Copying a local queue .............................................. 42
  Changing the attributes of a local queue .......................... 42
  Clearing a local queue ............................................. 42
Deleting a local queue .............................................. 43
  Browsing queues ..................................................... 43

© Copyright IBM Corp. 2006
Interleaved application transactions 104
WebSphere MQ critical database files 105
Critical processes 105
Queue manager clusters 110
Configuration considerations for availability 111
Configuration considerations for data integrity 111

Part 4. Configuring WebSphere MQ for HP NonStop Server 113

Chapter 9. Configuring WebSphere MQ 115
Changing configuration information in Pathway 115
The server classes of a queue manager 116
The attributes of server classes 116
Modifying the attributes of server classes 121
Adding new server classes 124
Removing server classes 126
Modifying the PATHWAY attributes of the queue manager’s Pathway configuration 126
Changing configuration information in configuration files 130
Editing configuration files 130
The WebSphere MQ configuration file, mqs.ini 131
The default process management rules configuration file, proc.ini 132
The queue manager configuration file, qm.ini 134
The process management rules configuration file, qmproc.ini 134
The contents of a WebSphere MQ configuration file 134
All queue managers 135
Default queue manager 136
Exit properties 136
API exits 137
Queue managers 137
The contents of a queue manager configuration file 137
Installable services 138
Restricted mode 138
Channels 139
TCP 140
Exit path 141

Chapter 10. WebSphere MQ security 143
Authority to administer WebSphere MQ 143
Managing the MQM group 144
Authority to work with WebSphere MQ objects 144
When security checks are made 144
How access control is implemented by WebSphere MQ 145
Identifying the user 146
Alternate user authority 147
Context authority 148
Creating and managing groups 148
Creating a group 148
Adding a user to a group 149
Displaying who is in a group 149
Removing a user from a group 149
Using the OAM to control access to objects 149
Giving access to a WebSphere MQ object 149
Using OAM generic profiles 151
Displaying access settings 153
Changing and preventing access to a WebSphere MQ object 153
Preventing access control checks 153
Channel security 153
Operating on channels, channel initiators, and listeners 155
Transmission queues 155
Channel exit programs 155
Protecting channels with SSL 156
How authorizations work 157
Authorizations for MQI calls 157
Authorizations for MQSC commands in escape PCFs 160
Authorizations for PCF commands 160

Chapter 11. Working with the WebSphere MQ Secure Sockets Layer (SSL) support 163
Introduction to OpenSSL 163
Where the files containing the WebSphere MQ SSL support code are installed 164
The entropy daemon 165
Configuring and running the entropy daemon 165
Stopping the entropy daemon 166
Preparing to use the WebSphere MQ SSL support 166
Verifying that the WebSphere MQ SSL support is installed 166
Verifying that the entropy daemon is running 167
Deciding how to specify the configuration file for the openssl req command 167
Working with keys and digital certificates 167
Generating public and private keys, and a request for a personal certificate 168
Importing digital certificates 170
Preparing the queue manager’s SSL files 170
A sample configuration for testing 172
The sample shell scripts and MQSC command files 173
Building and verifying the sample configuration 176
Running one of the queue managers on another system 176

Chapter 12. Transactional support 179
Introducing units of work 179
Using TMF for local and global units of work 180
Using global units of work 181
Using local units of work 182
Avoiding long running transactions 182
Syncpoint limits 182
Performing operations on persistent messages outside of syncpoint control 183
Performing operations on nonpersistent messages within a unit of work 183
The number of concurrent active transactions for an application 183
Configuring TMF for WebSphere MQ 184
Monitoring 185
Audit trail size 185

Contents V
Resource manager configuration 185
Troubleshooting 185

Chapter 13. The WebSphere MQ
dead-letter queue handler 187
Invoking the DLQ handler 187
The sample DLQ handler, amqsd1q 188
The DLQ handler rules table 188
Control data 188
Rules (patterns and actions) 189
Rules table conventions 192
How the rules table is processed 194
Ensuring that all DLQ messages are processed 195
An example DLQ handler rules table 195

Chapter 14. Process management 197
Attributes and rules 198
Default process attributes 198
Agent attributes 199
Application rules 201
Channel rules 202
Repository manager 203
Keyword definitions 203
Process management examples 205
Example 1: Configuring attributes for all agents 205
Example 2: Configuring attributes for types of agent 206
Example 3: Using threaded agent attributes 206
Example 4: Using the Repository stanza 207
Example 5: Using channel rules 207
Example 6: Using application rules 208
Example 7: Using environment variables 209

Part 5. Recovery and problem
determination 211

Chapter 15. Recovery and restart 213
Fault tolerance and recovery 213
Backing up and restoring WebSphere MQ 214
Backing up WebSphere MQ 214
Restoring WebSphere MQ 214
Recovery and restart of channel server, execution controller, and queue servers 215
Disaster recovery 216
Configuring WebSphere MQ, NonStop RDF, and AutoSYNC to support disaster recovery 216
Restarting operations on the backup system after a disaster 218

Chapter 16. Problem determination 221
Preliminary checks 221
Has WebSphere MQ run successfully before? 221
Are there any error messages? 222
Are there any return codes explaining the problem? 222
Can you reproduce the problem? 222
Have any changes been made since the last successful run? 222
Has the application run successfully before? 222
Problems with commands 224
Does the problem affect specific parts of the network? 224
Does the problem occur at specific times of the day? 224
Is the problem intermittent? 224
Have you applied any service updates? 225
Looking at problems in more detail 225
Have you obtained incorrect output? 225
Have you failed to receive a response from a PCF command? 228
Are some of your queues failing? 229
Does the problem affect only remote queues? 229
Is your application or system running slowly? 230
Application design considerations 230
Effect of message length 230
Effect of message persistence 230
Searching for a particular message 231
Queues that contain messages of different lengths 231
Frequency of syncpoints 231
Use of the MQPUT1 call 231
Number of threads in use 231
Error log files 231
Dead letter queues 232
Configuration files and problem determination 233
Tracing 233
Selecting components to trace 233
A sample trace 233
Trace files 235
First failure support technology (FFST) 235

Part 6. WebSphere MQ control
commands 237

Chapter 17. How to use WebSphere
MQ control commands 239
Names of WebSphere MQ objects 239
How to read syntax diagrams 240
Example syntax diagram 241
Syntax help 242
Examples 242

Chapter 18. The control commands 243
amqfls (alter WebSphere MQ object attributes) 244
dmqcwl (alter WebSphere MQ user information) 249
amqfuav (create code for data conversion exit) 251
amqm (create queue manager) 253
dmqm (delete queue manager) 256
dmqm (dump authority) 257
dmqwq (display queue managers) 259
dmqwq (display authority) 260
dmqc (display command server) 263
dmqfls (display WebSphere MQ object attributes) 264
dmqtec (display formatted trace) 267
dmqqrs (display WebSphere MQ user information) 268
dmqc (end command server) 270
dmq (end listener) 271
Chapter 19. Installable services and components

Why installable services? 303
Functions and components
  Entry points 304
  Return codes 305
  Component data 305
Initialization
  Primary initialization 306
  Secondary initialization 306
  Primary termination 306
  Secondary termination 306
Configuring services and components
  Service stanza format 307
  Service component stanza format 307
Creating your own service component 308
Using multiple service components
  Omitting entry points when using multiple components 308
  Example of entry points used with multiple components 308

Chapter 20. Authorization service

Object authority manager (OAM) 311
  Defining the service to the queue manager 311
  Refreshing the OAM after changing a user’s authority 311
  Migrating from MQSeries 312
Authorization service 312
  Configuring authorization service stanzas 312
Authorization service interface 313

Chapter 21. Name service 315

How the name service works 315
  Name service interface 316

Chapter 22. Installable services interface reference information

How the functions are shown 320
  Parameters and data types 320
MQZEP – Add component entry point
  Syntax 321
  Parameters 321
MQZ_REFRESH_CACHE – Refresh all authorizations
  Syntax 346
  Parameters 346
  C invocation 347
MQZ_CHECK_AUTHORITY – Check authority
  Syntax 323
  Parameters 323
  C invocation 327
MQZ_COPY_ALL_AUTHORITY – Copy all authority
  Syntax 328
  Parameters 328
  C invocation 330
MQZ_DELETE_AUTHORITy – Delete authority
  Syntax 331
  Parameters 331
  C invocation 332
MQZ_ENUMERATE_AUTHORITY_DATA – Enumerate authority data
  Syntax 334
  Parameters 334
  C invocation 336
MQZ_GET_AUTHORITY – Get authority
  Syntax 337
  Parameters 337
  C invocation 339
MQZ_GET_EXPLICIT_AUTHORITY – Get explicit authority
  Syntax 340
  Parameters 340
  C invocation 342
MQZ_INIT_AUTHORITY – Initialize authorization service
  Syntax 343
  Parameters 343
  C invocation 344
MQZ_REFRESH_CACHE – Refresh all authorizations
  Syntax 346
  Parameters 346
  C invocation 347
MQZ_SET_AUTHORITY – Set authority
  Syntax 348
  Parameters 348
  C invocation 350
MQZ_TERM_AUTHORITY – Terminate authorization service
  Syntax 351
  Parameters 351
  C invocation 352
MQZAD – Authority data
  Fields 353
  C declaration 355
MQZED – Entity descriptor
  Fields 356
  C declaration 357
MQZ_DELETE_NAME – Delete name
  Syntax 358
  Parameters 358
  C invocation 359
TACL macro files for building C sample programs. .......................... 465
TACL macro files for building C++ sample programs. ....................... 466
TACL macro files for building COBOL sample programs. ..................... 467
TACL macro files for building TAL sample programs. ......................... 469

Appendix J. User exits for Shared Resource Library (SRL) applications . 471
User exits in the PIC environment ........................................... 471
User exits in the non-PIC environment ...................................... 472
MQ_LOAD_ENTRY_POINT_EXIT - Loading user exits ........................ 472
  Parameters .......................................................... 472
  MQLXP - MQ_LOAD_ENTRY_POINT_EXIT parameter structure ............ 472
  Fields ............................................................ 472
  MQ_LOAD_ENTRY_POINT_EXIT example .................................. 473
Installing a non-PIC exit in the WebSphere MQ native non-PIC libraries .............................. 475
Installing an exit in the WebSphere MQ non-native TNS library .......... 476

Appendix K. Setting up communications .................................... 479
Supported communications protocols ....................................... 479
Configuring TCP/IP channels ............................................... 479
  Configuring the calling end of TCP/IP channels ....................... 479
  Configuring the responding end of TCP/IP channels .................. 480
  Specifying the TCP/IP process for TCP/IP channels ................... 481
The TCP/IP keep alive function ........................................... 482
Configuring SNA LU 6.2 channels ......................................... 482
  Configuring the calling end of SNA LU 6.2 channels ................. 483
  Configuring the responding end of SNA LU 6.2 channels .............. 483
Channel initiators ....................................................... 484

Appendix L. Notices ....................................................... 485
Trademarks ................................................................ 486
Index ........................................................................ 489
Sending your comments to IBM ........................................... 505
## Figures

1. Queues, messages, and applications .............................................. 31
2. Extract from an MQSC command file .................................................. 36
3. Extract from an MQSC command report file ......................................... 37
4. Typical output from a DISPLAY QMGR command ........................................ 39
5. Typical results from queue browser ...................................................... 44
6. The Monitoring Panels Main Menu ....................................................... 50
7. Queue Manager Menu, panel 1 ............................................................ 51
8. Queue Manager Menu, panel 2 ............................................................ 51
9. Search Criteria panel for queues ........................................................... 52
10. Queue Menu ......................................................................................... 52
11. Display Local Queue, panel 1 ............................................................... 53
12. Display Local Queue, panel 2 ............................................................... 53
13. Monitor Local Queues ............................................................................ 54
14. Search Criteria panel for channels .......................................................... 55
15. Channel Menu ....................................................................................... 55
16. Display Sender Channel, panel 1 .......................................................... 56
17. Display Sender Channel, panel 2 .......................................................... 56
18. Monitor Channels ................................................................................... 57
19. Remote administration using MQSC commands ....................................... 63
20. Setting up channels and queues for remote administration ......................... 64
21. WebSphere MQ for HP NonStop Server processes ..................................... 78
22. Example of a WebSphere MQ configuration file ....................................... 132
23. Example of a default process management rules configuration file .............. 133
24. Example queue manager configuration file ................................................. 134
25. Sample WebSphere MQ trace .................................................................. 234
26. FFST report for WebSphere MQ ............................................................... 236
27. Understanding services, components, and entry points .......................... 305
28. Authorization service stanzas in qm.ini .................................................. 312
29. Name service stanzas in qm.ini ............................................................. 317
30. The directory structure for an installation with one queue manager ........... 430
31. Sample MQLOADEXIT .......................................................................... 474
<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Categories of control commands</td>
<td>23</td>
</tr>
<tr>
<td>2.</td>
<td>Using the queue overflow file compared to using message overflow files</td>
<td>101</td>
</tr>
<tr>
<td>3.</td>
<td>Critical database files</td>
<td>105</td>
</tr>
<tr>
<td>4.</td>
<td>Protection methods used for critical processes</td>
<td>106</td>
</tr>
<tr>
<td>5.</td>
<td>Server classes that a queue manager is created with</td>
<td>116</td>
</tr>
<tr>
<td>6.</td>
<td>Attributes whose values must not be modified</td>
<td>117</td>
</tr>
<tr>
<td>7.</td>
<td>Attributes whose values can be modified</td>
<td>118</td>
</tr>
<tr>
<td>8.</td>
<td>List of possible ISO CCSIDs.</td>
<td>136</td>
</tr>
<tr>
<td>9.</td>
<td>Security authorization needed for MQCONN calls.</td>
<td>158</td>
</tr>
<tr>
<td>10.</td>
<td>Security authorization needed for MQOPEN calls.</td>
<td>158</td>
</tr>
<tr>
<td>11.</td>
<td>Security authorization needed for MQPUT1 calls.</td>
<td>158</td>
</tr>
<tr>
<td>12.</td>
<td>Security authorization needed for MQCLOSE calls.</td>
<td>159</td>
</tr>
<tr>
<td>13.</td>
<td>MQSC commands and security authorization needed.</td>
<td>160</td>
</tr>
<tr>
<td>14.</td>
<td>PCF commands and security authorization needed.</td>
<td>161</td>
</tr>
<tr>
<td>15.</td>
<td>Default process attributes</td>
<td>198</td>
</tr>
<tr>
<td>16.</td>
<td>Process management: agent attributes</td>
<td>199</td>
</tr>
<tr>
<td>17.</td>
<td>Process management: application rules</td>
<td>201</td>
</tr>
<tr>
<td>18.</td>
<td>Process management: channel rules</td>
<td>202</td>
</tr>
<tr>
<td>19.</td>
<td>Process management: repository manager</td>
<td>203</td>
</tr>
<tr>
<td>20.</td>
<td>Process management: keyword definition summary</td>
<td>203</td>
</tr>
<tr>
<td>21.</td>
<td>How to read syntax diagrams</td>
<td>240</td>
</tr>
<tr>
<td>22.</td>
<td>The authorities that are applicable to each type of object</td>
<td>261</td>
</tr>
<tr>
<td>23.</td>
<td>Specifying authorities for different object types</td>
<td>288</td>
</tr>
<tr>
<td>24.</td>
<td>Installable service components summary</td>
<td>303</td>
</tr>
<tr>
<td>25.</td>
<td>Example of entry-points for an installable service</td>
<td>308</td>
</tr>
<tr>
<td>26.</td>
<td>Installable services functions</td>
<td>319</td>
</tr>
<tr>
<td>27.</td>
<td>Fields in MQZAD</td>
<td>353</td>
</tr>
<tr>
<td>28.</td>
<td>Fields in MQZED</td>
<td>356</td>
</tr>
<tr>
<td>29.</td>
<td>Fields in MQACH</td>
<td>379</td>
</tr>
<tr>
<td>30.</td>
<td>Fields in MQAXC</td>
<td>382</td>
</tr>
<tr>
<td>31.</td>
<td>Fields in MQAXP</td>
<td>386</td>
</tr>
<tr>
<td>32.</td>
<td>System and default objects: queues</td>
<td>427</td>
</tr>
<tr>
<td>33.</td>
<td>System and default objects: channels</td>
<td>428</td>
</tr>
<tr>
<td>34.</td>
<td>System and default objects: namelists</td>
<td>428</td>
</tr>
<tr>
<td>35.</td>
<td>System and default objects: processes</td>
<td>428</td>
</tr>
<tr>
<td>36.</td>
<td>The files and directories of a queue manager</td>
<td>430</td>
</tr>
<tr>
<td>37.</td>
<td>Commands for queue manager administration</td>
<td>441</td>
</tr>
<tr>
<td>38.</td>
<td>Commands for command server administration</td>
<td>441</td>
</tr>
<tr>
<td>39.</td>
<td>Commands for queue administration</td>
<td>441</td>
</tr>
<tr>
<td>40.</td>
<td>Commands for process administration</td>
<td>442</td>
</tr>
<tr>
<td>41.</td>
<td>Commands for channel administration</td>
<td>442</td>
</tr>
<tr>
<td>42.</td>
<td>Other control commands</td>
<td>443</td>
</tr>
<tr>
<td>43.</td>
<td>Supported languages and environments for building and running applications</td>
<td>457</td>
</tr>
<tr>
<td>44.</td>
<td>Native PIC: libraries and include files for linking and compiling WebSphere MQ applications</td>
<td>459</td>
</tr>
<tr>
<td>45.</td>
<td>Native non-PIC applications and SRLs: include files for linking and compiling WebSphere MQ applications</td>
<td>459</td>
</tr>
<tr>
<td>46.</td>
<td>Non-PIC libraries for linking and compiling unthreaded and multithreaded WebSphere MQ applications</td>
<td>459</td>
</tr>
<tr>
<td>47.</td>
<td>Native PIC: library names for applications and DLLs in OSS and NonStop OS</td>
<td>460</td>
</tr>
<tr>
<td>48.</td>
<td>Non-native PIC: library names for applications and DLLs in OSS and NonStop OS</td>
<td>461</td>
</tr>
<tr>
<td>49.</td>
<td>C and COBOL sample programs for running on OSS</td>
<td>463</td>
</tr>
<tr>
<td>50.</td>
<td>C and COBOL sample programs for running on NonStop OS</td>
<td>464</td>
</tr>
<tr>
<td>51.</td>
<td>C++ sample programs for running on NonStop OS</td>
<td>465</td>
</tr>
<tr>
<td>52.</td>
<td>TAL sample programs for running on NonStop OS</td>
<td>465</td>
</tr>
<tr>
<td>53.</td>
<td>Environment required for user exits</td>
<td>471</td>
</tr>
</tbody>
</table>
About this book

IBM® WebSphere® MQ for HP NonStop Server, Version 5 Release 3 is a member of the WebSphere MQ family of products. These products provide application programming services that enable applications, running on the same system or on different systems, to communicate with each other using messages and queues. WebSphere MQ applications can run on a variety of different hardware and software platforms. They use a common application programming interface, called the Message Queue Interface or MQI, so that applications developed for one platform can readily be ported to another.

This book describes how to configure and manage a WebSphere MQ for HP NonStop Server installation, which includes the queues that applications use to send and receive messages.

For information about how to install WebSphere MQ for HP NonStop Server, see the [WebSphere MQ for HP NonStop Server, V5.3 Quick Beginnings]. For information about how to perform administration tasks for distributed queuing, see [WebSphere MQ Intercommunication].

Who this book is for

This book is primarily for system administrators and system programmers who are responsible for configuring and managing a WebSphere MQ for HP NonStop Server installation. The book also provides some useful information for application programmers, particularly regarding those aspects of the application programming interface that are specific to the NonStop OS platform. The book might also be useful to those application programmers who need 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 NonStop OS and its associated facilities. You do not need to have worked with WebSphere MQ before, but you must at least understand the basic concepts of WebSphere MQ.

Terms used in this book

The variable `var_installation_path` refers to the path to the var directory for an installation of WebSphere MQ for HP NonStop Server. You select this path in the OSS file system when you install the product.

The variable `opt_installation_path` refers to the path to the opt directory for an installation of WebSphere MQ for HP NonStop Server. You select this path in the OSS file system when you install the product.
How to use this book

Certain sections of this book refer you to other books for more information. You can download all the WebSphere MQ family books from www.ibm.com/software/integration/wmq/library/. You can access the latest version of the information center for WebSphere MQ at publib.boulder.ibm.com/infocenter/wmqv6/v6r0/index.jsp.
What’s new in WebSphere MQ for HP NonStop Server, V5.3

WebSphere MQ for HP NonStop Server, V5.3 provides standard WebSphere MQ Version 5.3 function with enhancements for NonStop OS. At the same time, it retains support for the interfaces and languages that were supported by previous releases.

WebSphere MQ for HP NonStop Server, V5.3 introduces an architectural change. Many of the product’s components have been moved to the HP Open System Services (OSS) operating environment. Internal processes that are not fault-tolerant, administration utilities, and most configuration and error log files now reside in the OSS operating environment. Some components have not been moved however. Processes that are fault-tolerant, and databases that are audited, are examples of components that remain in the NonStop OS operating environment.

OSS is a UNIX® operating system. As a result, the way that you configure and administer a queue manager on WebSphere MQ for HP NonStop Server is similar to the way that you configure and administer a queue manager on other UNIX systems. And, by using OSS, the internal processes of a queue manager can be multithreaded. However, by retaining certain critical components in NonStop OS, the fault-tolerance, data integrity, and availability characteristics of these components can be maintained.

One of the new components in this release is a fault-tolerant execution controller (EC) that is implemented as a process-pair. The new execution controller replaces the EC-Boss and EC components of the previous release, which were not fault-tolerant.

The execution controller uses a process management rules configuration file, qmproc.ini, to manage the server processes of the queue manager that are not configured as server classes within Pathway. These server processes are local queue manager agents, message channel agents (MCAs), Object Authority Manager (OAM) servers, and repository managers. In the previous release, repository managers were configured within Pathway.

You can edit the process management rules configuration file to establish a hierarchy of rules for controlling the names of server processes, for specifying the CPUs in which server processes must run, and for setting other process attributes. Because every queue manager has its own process management rules configuration file, each queue manager can have its own set of rules, but you can specify a default set of rules for an installation. You can update the rules for a queue manager while the queue manager is still running. Using a hierarchy of rules, you can establish rules that apply to all processes, rules that apply only to certain classes of processes, and rules that apply only to specific processes. The rules specified at one level in the hierarchy override the rules specified at higher levels.

To enhance channel security, WebSphere MQ for HP NonStop Server, V5.3 provides support for the Secure Sockets Layer (SSL). The implementation is based on the OpenSSL toolkit from the OpenSSL Project. The required OpenSSL runtime components are supplied with the product. For more information about the OpenSSL Project, see www.openssl.org.
What's new

Using SSL, the queue managers at each end of a message channel can authenticate each other and, on an MQI channel, the WebSphere MQ client and the queue manager can authenticate each other. You can also use the SSL support to encrypt and decrypt data that is transported over a channel, and to detect any tampering of that data. The SSL support also provides tools for managing keys and digital certificates. The SSL support provided by WebSphere MQ for HP NonStop Server is fully compatible with, and interoperable with, the SSL support provided on other WebSphere MQ platforms.

WebSphere MQ for HP NonStop Server, V5.3 also has the following new features:

- Support for API exits, which an application can use to monitor MQI calls, and inspect and change the parameters on MQI calls.
- Performance enhancements to the Object Authority Manager (OAM), and support for OAM generic profiles.
- Support for the DISPLAY QSTATUS command, which you can use to determine the name, process ID, and thread ID of each application that is accessing a queue.
- Improvements to the trace facilities to aid problem determination.
- Support for the CONNAME and QMNAME parameters on the STOP CHANNEL command. A channel is stopped only if the partner connection name and partner queue manager name matches the values specified by these parameters.
- An MQI library with full support for threaded OSS applications.
- Support for sharing connection and object handles among threads belonging to the same process.
- An application can call MQCMIT to commit a local unit of work and MQBACK to back out a local unit of work. An application still controls a global unit of work by interfacing directly with TMF, using the BEGINTRANSACTION, ENDTRANSACTION, and ABORTTRANSACTION calls.
- Support for the local queue attribute NonPersistentMessageClass. If the attribute is set to the appropriate value, nonpersistent messages on the queue are not discarded when the queue manager restarts following a quiesced or immediate shutdown.
- WebSphere MQ classes for Java™ and WebSphere MQ classes for Java Message Service (JMS) are supplied as an integral component of the product. You no longer require SupportPac™ MA88 to provide this function.
- WebSphere MQ Publish/Subscribe is supplied as an integral component of the product. SupportPac MA0C, which used to provide this function, has now been withdrawn.
- Control commands can be issued from an OSS shell command prompt as well as from a TACL command prompt.
- Support for system configurations that use the Remote Database Facility (RDF) and AutoSYNC for disaster recovery. Operations on large persistent messages stored in message overflow files can be replicated to a backup system using RDF.
- Support for the DLL linkage model for applications and exits.
- Support for the IEEE floating point format in addition to the native hardware floating point format.
- Support for creating a Version 5.3 queue manager from an existing Version 5.1 queue manager and its data. Support is still provided for multiple installations of WebSphere MQ on a single system.
WebSphere MQ for HP NonStop Server, V5.3 is installed from a CD using the HP independent product setup tool, IPSetup, running on Microsoft® Windows®.

WebSphere MQ for HP NonStop Server, V5.3 runs on the HP NonStop servers and the new HP Integrity NonStop servers that are based on the Intel® Itanium® 2 processor.
# Part 1. Introduction

**Chapter 1. Introduction to WebSphere MQ**

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 ..................................... 4
- Retrieving messages from queues ................................................. 5
WebSphere MQ objects ......................................................................... 5
- Object names ................................................................................ 5
- 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
- Channels .................................................................................... 10
- Queue manager clusters ............................................................. 10
- Namelists .................................................................................... 10
- Authentication information objects ............................................. 11
- System and default objects ......................................................... 11
Clients and servers ............................................................................. 11
  - WebSphere MQ applications in a client/server environment ......... 11
Extending queue manager facilities ................................................. 12
- User exits .................................................................................... 12
- Installable services ....................................................................... 12
Security ............................................................................................ 13
  - Object Authority Manager (OAM) facility ................................. 13
  - Channel security using SSL ..................................................... 13
Transactional support ....................................................................... 13

**Chapter 2. An introduction to WebSphere MQ administration**

Local and remote administration ...................................................... 15
Performing administration tasks using commands ............................ 15
- Control commands ....................................................................... 16
  - WebSphere MQ Script (MQSC) commands ............................... 16
  - PCF commands ......................................................................... 16
Administration on WebSphere MQ for HP NonStop Server ................. 16
  - The Pathway control program (PATHCOM) .............................. 17
  - The process management rules configuration file ....................... 17
  - The TMF configuration utility (TMFCOM) ............................... 18
  - The Subsystem Control Facility (SCF) .................................... 18
Understanding WebSphere MQ file names .................................... 18
  - Transforming a queue manager name ...................................... 18
  - Transforming the name of a WebSphere MQ object .................. 19
Chapter 1. Introduction to WebSphere MQ

This chapter introduces the WebSphere MQ, Version 5.3 products 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”
- “WebSphere MQ objects” on page 5
- “Clients and servers” on page 11
- “Extending queue manager facilities” on page 12
- “Security” on page 13
- “Transactional support” on page 13

WebSphere MQ and message queuing

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

WebSphere MQ products implement a common application programming interface known as the Message Queue Interface (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 is has been started.

Message driven processing

When messages arrive on a queue, they can automatically start an application using triggering. If necessary, the application can be stopped when the 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.

**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 MQOPEN call
2. Issues an MQPUT call to put the message onto the queue

Another application can retrieve the message from the same queue by issuing an 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.

For information about planning the amount of storage you need for queues, see the WebSphere MQ for HP NonStop Server, V5.3 Quick Beginnings.

**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 call 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 call from the application determines the method used.

---

**WebSphere MQ objects**

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

The manipulation or administration of objects includes:

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

**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 239.
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

On WebSphere MQ for HP NonStop Server, you can display and look at the status of objects using the Monitoring Panels facility in Pathway. You cannot, however, create, alter, or delete objects using the Monitoring Panels.

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

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

In WebSphere MQ, there are two ways of referring to an attribute:
• Using its PCF name, for example, MaxMsgLength.
• Using its MQSC command name, for example, MAXMSGL.

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

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

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

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

Before applications can send messages to a queue on another queue manager, you must have defined a transmission queue and channels between the queue managers, unless you have grouped one or more queue managers together into a cluster. For more information about clusters, see “Remote administration using queue manager clusters” on page 62.

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 call specifying a queue name that is the name of a model queue. The dynamic queue that is created in this way is a local queue whose attributes are taken from the model queue definition. The dynamic queue name can be specified by the application, or the queue manager can generate the name and return it to the application.

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

**Defining queues**

Queues are defined to WebSphere MQ using:

- The MQSC command DEFINE
- The PCF Create Queue command

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

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

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

In addition to the attributes that all WebSphere MQ queues have, WebSphere MQ for HP NonStop Server queues have several attributes that are specific to this WebSphere MQ implementation and control features that are implemented only on the NonStop OS platform. You can set these attributes using the almqlfs control command. Here are some examples of these attributes:

- The symbolic name of the queue server that is managing the queue.
- A threshold message size that determines how persistent messages are stored in the queue. Persistent messages greater than or equal to the specified size are stored in a different way compared to persistent messages less than the specified size.
- Whether nonpersistent messages are checkpointed to the backup process of the queue server.
WebSphere MQ objects

WebSphere MQ objects other than queues also have attributes that are specific to WebSphere MQ for HP NonStop Server, and you can use **altmqfls** to set these attributes as well. For the definition of the **altmqfls** command, see “altmqfls (alter WebSphere MQ object attributes)” on page 244.

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 47 and “runmqtrm (start trigger monitor)” on page 284. 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 63. 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 “WebSphere MQ instrumentation events” on page 433.

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
WebSphere MQ objects

A 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 MQINQ call.

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.

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.

Queue manager 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 queue manager 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 61, 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 Secure Sockets Layer (SSL) support provided by WebSphere MQ for HP NonStop Server does not use authentication information objects. The location of certificate revocation lists (CRLs) is specified in a different way. For information about the SSL support provided by WebSphere MQ for HP NonStop Server, see Chapter 11, “Working with the WebSphere MQ Secure Sockets Layer (SSL) support,” on page 163.

System and default objects

The system and 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.DEFAULT; for example, the default local queue is SYSTEM.DEFAULT.LOCAL.QUEUE, and the default receiver channel is SYSTEM.DEFAULT.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 427 for more information about system and default objects.

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, and about client support in general, see WebSphere MQ Clients.

No WebSphere MQ client is supplied for the NonStop OS platform. This means that an application running on NonStop OS cannot connect to a queue manager that is running on another system. However, a queue manager running on NonStop OS can support WebSphere MQ client applications that are running on other systems.

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
- 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 exit

A data conversion exit converts a message in a user defined format from one coded character set to another. Data conversion exits are described in the WebSphere MQ Application Programming Guide.

Cluster workload exit

You can use a cluster workload exit to determine which queue manager in a cluster to route a message to. Call definition information is given in WebSphere MQ Queue Manager Clusters.

API exits

API exits let you write code that changes the behavior of MQI 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 371 and the WebSphere MQ Application Programming Guide.

Installable services

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

An implementation of an installable service is called a service component. As a general rule, you can use the default service components supplied with WebSphere MQ, or you can write your own components 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 143.

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

WebSphere MQ for HP NonStop Server does not provide a default name service component.

For more information about installable services and service components, see Chapter 19, “Installable services and components,” on page 303.

Security
In WebSphere MQ, there are two ways of providing security:

• The Object Authority Manager (OAM) facility
• 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 143.

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 156.

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.
In a local unit of work, only the resources of the queue manager are updated. From the point of view of the application, syncpoint coordination for a local unit of work is performed by the queue manager. Internally, however, the queue manager exploits the facilities of the NonStop Transaction Management Facility (TMF) to perform this function.

In a global unit of work, resources belonging to other resource managers, such as database managers, are also updated. On NonStop OS, a global unit of work can be coordinated only by TMF, or a transaction manager that is based on TMF, such as NonStop Tuxedo. A WebSphere MQ queue manager can participate in a global unit of work only as a resource manager, not as a transaction manager. An application can use TMF, for example, to coordinate a global unit of work that involves updates to WebSphere MQ resources and updates to Enscribe files, NonStop SQL/MP databases, or NonStop SQL/MX databases.

For more information about transactional support, see Chapter 12, “Transactional support,” on page 179.
Chapter 2. An introduction to WebSphere MQ administration

This chapter introduces WebSphere MQ administration.

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

The chapter contains the following sections:

- “Local and remote administration”
- “Performing administration tasks using commands”
- “Administration on WebSphere MQ for HP NonStop Server” on page 16
- “Understanding WebSphere MQ file names” on page 18

Local and remote administration

You administer WebSphere MQ objects locally or remotely.

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

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

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

Chapter 6, “Administering remote WebSphere MQ objects,” on page 61 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. These sets of commands are described in the following sections:

- “Control commands” on page 16
- “WebSphere MQ Script (MQSC) commands” on page 16
- “PCF commands” on page 16
Using control commands

Control commands
Control commands allow you to perform administrative tasks on queue managers. On NonStop OS, you can enter a control command at an OSS shell command prompt or a TACL command prompt.

For information about how to use control commands, see "Chapter 3. Administering queue managers," on page 23.

WebSphere MQ Script (MQSC) commands
Use MQSC commands to manage queue manager objects, including the queue manager itself, channels, queues, and process definitions.

You issue MQSC commands to a queue manager using the runmqsc control 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.

The WebSphere MQ Script (MQSC) Command Reference contains a description of each MQSC command and its syntax, but the commands are summarized in Appendix E, "Comparing command sets," on page 441.

See "Performing local administration tasks using MQSC commands" on page 32 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 59 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 60.

Administration on WebSphere MQ for HP NonStop Server
On WebSphere MQ for HP NonStop Server, you can perform administration tasks using the following interfaces. These are standard WebSphere MQ administration interfaces that are also available on other WebSphere MQ server platforms.
WebSphere MQ for HP NonStop Server

- Control commands, MQSC commands, and PCF commands, as described in “Performing administration tasks using commands” on page 15
- The WebSphere MQ configuration file and the queue manager configuration file, as described in Chapter 9, “Configuring WebSphere MQ,” on page 113

You can also perform administration tasks using the interfaces described in the following sections. These interfaces are specific to WebSphere MQ for HP NonStop Server and not available on other WebSphere MQ server platforms.

- “The Pathway control program (PATHCOM)”
- “The process management rules configuration file”
- “The TMF configuration utility (TMFCOM)” on page 18
- “The Subsystem Control Facility (SCF)” on page 18

The Pathway control program (PATHCOM)

Certain server processes of a queue manager are configured as server classes within Pathway. These server processes include the execution controller, the queue servers, the channel server, and the channel initiators.

The control command crmqm automatically creates a default Pathway configuration for a queue manager. This configuration can then be started by the strmqm command, ended by the endmqm command, and deleted by the dltmqm command. However, certain changes to the attributes of the server processes can be made only by using the Pathway control program, PATHCOM. Occasionally, you might also need to create and maintain additional server classes, which can be done only by using PATHCOM.

For more information about using PATHCOM to administer WebSphere MQ, see “Changing configuration information in Pathway” on page 115.

The process management rules configuration file

Every queue manager has an associated process management rules configuration file whose name is qmproc.ini. The process management rules configuration file contains a set of rules that are used by the execution controller to manage those server processes of the queue manager that are not configured as server classes within Pathway.

When you create a queue manager, a process management rules configuration file with a default set of rules is created automatically. However, you might want to edit the file subsequently to modify the default rules or to add your own rules. Here are some of the reasons why you might want to edit the file:

- You can allocate meaningful names to the server processes so that they can be identified easily.
- To aid load balancing and performance tuning, you can specify that certain server processes must run in designated CPUs.
- You can specify whether a local queue manager agent runs as a thread or a process.

For information about how to use a process management rules configuration file, see Chapter 14, “Process management,” on page 197.
The TMF configuration utility (TMFCOM)

WebSphere MQ for HP NonStop Server uses TMF to coordinate all units of work. By considering the transaction processing requirements of your WebSphere MQ applications, you can determine the number and size of the circular audit trail files that TMF requires in order to support the applications and operate effectively. You can then use the TMF configuration utility, TMFCOM, to create and maintain these log files. For guidance on how to perform these tasks, see “Configuring TMF for WebSphere MQ” on page 184.

The Subsystem Control Facility (SCF)

Several important subsystems on NonStop OS are administered using the Subsystem Control Facility (SCF). Here are some of the important ones:

- The OSS file sets used by WebSphere MQ (for more information, see the WebSphere MQ for HP NonStop Server, V5.3 Quick Beginnings).
- The logical and physical disk systems used by WebSphere MQ (for more information, see the WebSphere MQ for HP NonStop Server, V5.3 Quick Beginnings).
- The communications subsystems TCP/IP and SNAX (for more information, see Appendix K, “Setting up communications,” on page 479).

Understanding WebSphere MQ file names

WebSphere MQ objects and queue managers are represented by files and directories in the OSS file system, and by files and subvolumes in the NonStop OS file system. Because the names of objects and queue managers are not necessarily valid file, directory, or subvolume names, WebSphere MQ transforms the names of objects and queue managers into valid file system names where necessary. This process is referred to as name transformation.

Transforming a queue manager name

On NonStop OS, each queue manager has an associated directory in the OSS file system and an associated subvolume in the NonStop OS file system.

The fully qualified path name of a queue manager’s directory in the OSS file system has the following three components:

- A prefix, as specified by the Prefix entry in the QueueManager stanza for the queue manager in the WebSphere MQ configuration file, mqs.ini. The WebSphere MQ configuration file is in the directory var_installation_path/var/mqm.
- The directory name qmgrs.
- The queue manager name transformed into a valid directory name, as specified by the Directory entry in the QueueManager stanza for the queue manager in the WebSphere MQ configuration file.

The name of a queue manager can contain up to 48 characters. However, the OSS file system imposes limitations on the length of a directory name, and on the characters that can be used in the name. As a result, WebSphere MQ transforms a queue manager name 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, WebSphere MQ transforms the queue manager name
saturn.queue.manager to the directory name saturn!queue!manager.

The name of the subvolume associated with a queue manager is specified by the
HPNSSGuardianSubvol entry in the QueueManager stanza for the queue manager
in the WebSphere MQ configuration file. WebSphere MQ forms the subvolume
name by transforming the queue manager name. However, because a subvolume
name can contain a maximum of 8 characters, the transformation rules are not the
same as those that WebSphere MQ uses to form a directory name in the OSS file
system.

To form a subvolume name, WebSphere MQ uses the first eight characters of the
queue manager's name. Lowercase letters are folded to uppercase, and any
character that is not alphanumeric is replaced by the letter X. If the first character
is numeric, that character is also replaced by the letter X.

For example, WebSphere MQ transforms the queue manager name
saturn.queue.manager to the subvolume name SATURNXQ.

If a transformed name cannot be used because a subvolume with that name
already exists, WebSphere MQ uses a name with the format QMGRnnnn, where
nnnn is a numeric suffix starting at 0000. As an alternative to using a transformed
name, you can specify the name of the queue manager's subvolume by using the
-ns parameter on the crtmqm command.

**Transforming the name of a WebSphere MQ object**

WebSphere MQ might also need to transform the names of WebSphere MQ objects.
The rules for transforming the name of an object are different to those for
transforming a queue manager name because, although there are typically only a
few queue managers in an installation, there can be a large number of WebSphere
MQ objects for each queue manager. Only queues and namelists are represented in
the file system; process definitions and channels are not affected by these
considerations.

There is no simple relationship between the name of a WebSphere MQ object and
the names of its associated files. You can use the dspmqfls command to determine
the names of the files associated with an object.
Part 2. Administering WebSphere MQ for HP NonStop Server

Chapter 3. Administering queue managers . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
When a queue manager cannot convert messages
in built in formats . . . . . . . . . . .
The file ccsid.tbl . . . . . . . . . . . .
Default data conversion . . . . . . . .
Converting messages in user defined formats . .
Changing the queue manager's CCSID . . . .

22

72
72
73
73
73

WebSphere MQ for HP NonStop Server: System Administration Guide


Chapter 3. Administering queue managers

This chapter tells you how to administer queue managers using control commands and describes the Monitoring Panels facility of WebSphere MQ for HP NonStop Server.

It contains the following topics:
- "Using control commands"
- "Creating a queue manager" on page 24
- "Starting a queue manager" on page 29
- "Stopping a queue manager" on page 29
- "Deleting a queue manager" on page 30

Using control commands

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

Table 1. Categories of control commands

<table>
<thead>
<tr>
<th>Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Queue manager commands</td>
<td>Queue manager control commands include commands for creating, starting, stopping, and deleting queue managers and command servers.</td>
</tr>
<tr>
<td>Channel commands</td>
<td>Channel commands include commands for starting and ending channels and channel initiators.</td>
</tr>
<tr>
<td>Utility commands</td>
<td>Utility commands include commands associated with:</td>
</tr>
<tr>
<td></td>
<td>• Running MQSC commands</td>
</tr>
<tr>
<td></td>
<td>• Conversion exits</td>
</tr>
<tr>
<td></td>
<td>• Authority management</td>
</tr>
<tr>
<td></td>
<td>• Trigger monitors</td>
</tr>
<tr>
<td></td>
<td>• Displaying the names of files associated with WebSphere MQ objects</td>
</tr>
<tr>
<td></td>
<td>• Changing the attributes of WebSphere MQ objects that control features implemented only on the NonStop OS platform</td>
</tr>
</tbody>
</table>

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

Using control commands on NonStop OS

On NonStop OS, you can enter a control command at an OSS shell command prompt or a TACL command prompt.

At an OSS shell command prompt, the name of the command is case sensitive but, at a TACL command prompt, the name of the command is not case sensitive. At either command prompt, the parameters of the command are case sensitive.

For example, consider the following command:

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

Note the following comments:
Using control commands

- At an OSS command prompt, you must enter the name of the command as `crtmqm`, not `CRTMQM`. However, at a TACL command prompt, you can enter `crtmqm`, `CRTMQM`, or even `CRTmqm`.
- The flag `-u` must be lowercase, not uppercase as in `-U`.
- You must enter the name of the dead letter queue as `SYSTEM.DEAD.LETTER.QUEUE`.
- The command creates a queue manager called `jupiter.queue.manager`, which is not the same name as `JUPITER.queue.manager`, for example.

For more information about the `crtmqm` command, see “crtmqm (create queue manager)” on page 253.

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 Queue 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 253). The `crtmqm` command automatically creates the required default objects and system objects (described in “System and default objects” on page 11). 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.

Guidelines for creating queue managers

Before you 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.
Creating a queue manager

**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 nodes are run with a single queue manager, but larger server machines might run with multiple queue managers.

**Specify a default queue manager**

You can designate one queue manager in an installation to be the default queue manager, although an installation does not have to have one. The default queue manager is the queue manager to which applications connect 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 specified for the installation.

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

Creating a default queue manager is described in "Creating a default queue manager" on page 27.

**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` parameter to specify the name of the dead letter queue. Alternatively, you can use the MQSC command ALTER QMGR to specify the name of the dead letter queue for
Creating a queue manager

A queue manager that you have already created. See "Altering queue manager attributes" on page 40 for an example of the ALTER QMGR command.

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. Every queue manager can have a default transmission queue.

When you create a queue manager, use the -d parameter 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 40 for more information.

The queue manager's directory
You can create the queue manager's directory, var_installation_path/var/mqm/qmgrs/qmname, before you use the crtmqm command. The directory can be in a separate OSS fileset. If the queue manager's directory already exists when you run the crtmqm command, the command uses the directory for the queue manager data provided the directory is empty and is owned by the user who ran the installation script, instmqm. If the directory is not empty, the command creates a new directory. If the directory is not owned by user who ran the installation script, the command fails with a First Failure Support Technology™ (FFST™) message.

The queue manager's subvolume
The queue manager's subvolume is the subvolume where the NonStop OS files of the queue manager are stored. By default, when you create a queue manager, the queue manager's subvolume is in the volume specified by the HPNSSQMDefaultGuardianVol entry in the AllQueueManagers stanza in the WebSphere MQ configuration file, mqs.ini. The name of the subvolume is derived from the name of the queue manager, and the fully qualified local name of the subvolume is recorded in the HPNSSGuardianSubvol entry in the QueueManager stanza for the queue manager in the WebSphere MQ configuration file. You can specify a different volume and subvolume by using the -ns parameter on the crtmqm command. However, you cannot change the volume or subvolume after you have created the queue manager.

The home terminal of a queue manager
When you create a queue manager, you can use the -nh parameter on the crtmqm command to specify the home terminal to be used by the server processes of the queue manager. Subsequently, you can change the home terminal for an individual server process. If you don’t specify a home terminal when you create a queue manager, the default home terminal is $ZHOME.

For more information about using home terminals, see the description of the HOMETERM attribute of a server class in "The attributes of server classes" on page 116 and the WebSphere MQ for HP NonStop Server, V5.3 Quick Beginnings.

Naming the server processes of a queue manager
When you create a queue manager, NonStop OS allocates names to the server processes of the queue manager. If you require more control over the names of these processes, you can use parameters on the crtmqm command to specify the names of certain server processes. Being able to
choose the names of these processes might make it easier for you to identify which queue manager a process belongs to.

Creating a default queue manager

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

- Creates a default queue manager called SATURN.QUEUE.MANAGER
- Specifies the names of both a default transmission queue and a dead letter queue
- Creates the system and default objects
- Allows NonStop OS to allocate names to the server processes of the queue manager

```
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 create it yourself.

- **-u SYSTEM.DEAD.LETTER.QUEUE**
  Is the name of the dead letter queue that is created automatically as one of the system and default objects.

- **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 253. The system and default objects are listed in Appendix A, “System and default objects,” on page 427.

Making an existing queue manager the default

You can make an existing queue manager the default queue manager.

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 must do this manually, using a text editor.
- If you do not have a default queue manager in the installation, 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 the one you want.
Changing the default queue manager

See Chapter 9, “Configuring WebSphere MQ,” on page 115 for information about configuration files.

When you have changed the configuration information, stop the queue manager and restart it. See “Stopping a queue manager” on page 29 for information about how to do this.

Backing up configuration files after creating a queue manager

Certain WebSphere MQ configuration information is stored in configuration files. There are four types of configuration file:

- When you install WebSphere MQ for HP NonStop Server, the following configuration files are created automatically:
  - A WebSphere MQ configuration file, mqs.ini, which contains a list of queue managers that is updated each time you create or delete a queue manager.
  - A default process management rules configuration file, proc.ini, whose contents are used to form the process management rules configuration file, qmproc.ini, of a queue manager when you create the queue manager.

Each installation has its own WebSphere MQ configuration file and default process management rules configuration file.

- When you create a queue manager, the following configuration files are created automatically:
  - A queue manager configuration file, qm.ini, which contains configuration information specific to the queue manager.
  - A process management rules configuration file, qmproc.ini, which contains a set of rules that are used by the execution controller to manage those server processes of the queue manager that are not configured as server classes within Pathway.

As a general rule, after you create a new queue manager, back up the following configuration files:

- The WebSphere MQ configuration file
- The new queue manager configuration file
- The new process management rules configuration file

If, later on, you create a queue manager that causes a problem, you can reinstate the backups after you have removed the source of the problem.

For more information about configuration files, see “Changing configuration information in configuration files” on page 130.

For information about how to configure WebSphere MQ for HP NonStop Server for disaster recovery, see Chapter 15, “Recovery and restart,” on page 213.

Creating entries in the principal database of a queue manager

On NonStop OS, every queue manager has a principal database. Each entry in the principal database maps a WebSphere MQ principal to a NonStop OS user ID.

The crtmqm command automatically creates an entry in the principal database for the user who ran the installation script, instmqm. The principal created is always mqm, for compatibility with other WebSphere MQ implementations.
After you have created a queue manager, you can create entries in the principal database for the other users of the queue manager. For more information about the principal database and how to create entries in it, see "Identifying the user" on page 146.

Starting a queue manager

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

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

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

Stopping a queue manager

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

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

For a detailed description of the `endmqm` command and its options, see "endmqm (end queue manager)" on page 272.

Quiesced shutdown

By default, the `endmqm` command performs a quiesced, or controlled, shutdown of the specified queue manager. A quiesced shutdown notifies applications that the queue manager is quiescing and waits until all the applications have disconnected. This might take a while to complete.

For a quiesced shutdown, enter:

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

Control returns to you only after all the applications have disconnected and the queue manager has ended. The queue manager also performs a quiesced shutdown if you use the `-c` or `-w` parameter on the `endmqm` command.

Problems in shutting down a queue manager are often caused by applications that:

- 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 you use the `endmqm` command and the queue manager does not stop, or the shutdown is too slow, you can break out of the command by pressing Ctrl+C from within an OSS shell, or by pressing Break and then entering the STOP command from within TACL. You can then try an immediate shutdown of the queue manager.

Immediate shutdown

In 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, enter:
Stopping a queue manager

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 parameter. For example:
endmqm -p saturn.queue.manager

If this method still does not work, see “Stopping a queue manager manually” on page 439 for an alternative solution.

Deleting a queue manager

To delete a queue manager, first stop it and then enter the following command:
dltmqm saturn.queue.manager

Note: Deleting a queue manager is a drastic step, because you also delete all the resources associated with the queue manager, including all its queues and their messages, and all object definitions. When you enter the dltmqm command, no prompt is displayed to give you an opportunity to change your mind. When you press Enter, all the associated resources are lost.

If this method of deleting a queue manager does not work, see “Removing a queue manager manually” on page 439 for an alternative method.

For a description of the dltmqm command and its options, see “dltmqm (delete queue manager)” on page 256.

Make sure that only trusted administrators have the authority to use this command. For information about security, see Chapter 10, “WebSphere MQ security,” on page 143.
Chapter 4. Administering local WebSphere MQ objects

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

This chapter contains the following sections:

- “Supporting application programs that use the MQI” on page 31
- “Performing local administration tasks using MQSC commands” on page 32
- “Working with queue managers” on page 39
- “Working with local queues” on page 40
- “Working with alias queues” on page 44
- “Working with model queues” on page 46
- “Managing objects for triggering” on page 47
- “Using the Monitoring Panels” on page 49

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.

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

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

- The queue manager must exist and be running.
- The first application queue, from which the messages are to be removed, must be defined.
- The second queue, on which the application puts the messages, must also be defined.
Performing local administration tasks using MQSC commands

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

You can use MQSC commands to manage queue manager objects, including the queue manager itself, clusters, channels, queues, namelists, and process definitions. This section deals with queue managers, queues, and process definitions. For information about administering channel objects, see WebSphere MQ Intercommunication.

You issue MQSC commands to a queue manager using the runmqsc command. (For details of this command, see “runmqsc (run MQSC commands)” on page 281.) 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 a 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 35.)

You can run the runmqsc command in three ways, depending on which parameters you use:

• 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.

The names of parameters of MQSC commands are shown in this book in uppercase (for example, RQMNAME), although they are not case sensitive. The name of a parameter is limited to eight characters.

MQSC commands are summarized in Appendix E, “Comparing command sets,” on page 441. 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.
Using MQSC commands for local administration

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 parameters, can be written in uppercase or lowercase. Object names in MQSC commands are folded to uppercase (that is, ORANGE.LOCAL.QUEUE and orange.local.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.

When you enter a runmqsc command at an OSS shell command prompt, the name of the command is case sensitive but, at a TACL command prompt, the name of the command is not case sensitive. At either command prompt, the parameters of the command are case sensitive. This rule applies to all control commands. See “Using control commands” on page 23 for more information.

Standard input and output

The standard input device, also referred to as stdin, is the device from where an application or command takes its input. Typically this is the keyboard, but you can redirect the input to come from another device or a file. The standard output device, also referred to as stdout, is the device to where an application or command sends its output. Typically this is a display, but you can redirect the output to another device or a file.

When you enter a control command at an OSS shell command prompt, you can use the < operator to redirect the input. If this operator is followed by a file name, the command takes its input from the file. Similarly, the > operator redirects the output. If this operator is followed by a file name, the command sends its output to the file.

When you enter a control command at a TACL command prompt, you can use the standard /IN / directive to specify that the command takes its input from a file. Similarly, you can use the standard /OUT / directive to specify that the command sends its output to a file.

Running MQSC commands interactively

To run MQSC commands interactively, enter the following command at an OSS shell command prompt or a TACL command prompt:

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, enter the command:

runmqsc jupiter.queue.manager

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

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

DEFINE QLOCAL(ORANGE.LOCAL.QUEUE)
Using MQSC commands for local administration

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 enter MQSC commands, the queue manager returns operator messages that confirm your actions or tell you about the errors you have made. For example, this message confirms that a queue has been created:

AMQ8006: WebSphere MQ queue created.

And this message indicates that you have made a syntax error:

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

AMQ8426: Valid MQSC commands are:

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

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.

Using fix command features
On NonStop OS, the runmqsc command has fix command features, which you can use to recall and edit previously entered MQSC commands, and run the commands again. For example, using these features, you can perform the following tasks:

- Enter history or h to display the 10 most recently entered MQSC commands. Alternatively, you can enter history n or h n, where n is an integer, to display the n most recently entered MQSC commands.
- Enter !m, where m is the number of a previously entered MQSC command, to run the command again.
- Enter fc to present the last MQSC command entered for editing. Alternatively, you can enter fc m, where m is the number of a previously entered MQSC command.
Using MQSC commands for local administration

command, to present the command for editing. Or you can enter `fc string` to present for editing the most recently entered MQSC command that begins with the characters in `string`.

To edit an MQSC command, use `d` to delete a character, `i` to insert characters, and `r` to replace characters.

Ending interactive input of MQSC commands
To stop working with MQSC commands, enter the END command or press Ctrl+Y.

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, you can run MQSC commands from a text file. To do this, first prepare a text file containing the MQSC commands using your usual text editor. A file containing MQSC commands is called an MQSC command file. For more information about the contents of an MQSC command file and how to prepare one, see “MQSC command files.”

Then, when you enter the `runmqsc` command, redirect the input to come from the MQSC command file. For information about redirecting input, see “Standard input and output” on page 33.

For example, enter the following command at an OSS shell command prompt to run a sequence of MQSC commands contained in the file `mqscin`:

```
runmqsc < mqscin
```

Alternatively, enter the following command at a TACL command prompt:

```
runmqsc /IN mqscin/
```

In a similar way, you can redirect the output to a file. For example, enter the following command at an OSS shell command prompt to redirect the output to the file `mqscout`:

```
runmqsc < mqscin > mqscout
```

Alternatively, enter the following command at a TACL command prompt:

```
runmqsc /IN mqscin, OUT mqscout/
```

For more information about what the output contains, see “MQSC command reports” on page 36.

The previous examples of `runmqsc` commands do not contain a queue manager name as a parameter. As a result, the MQSC commands are run against the default queue manager. To run the MQSC commands against a queue manager that is not the default queue manager, enter this form of `runmqsc` command at an OSS command prompt:

```
runmqsc saturn.queue.manager < mqscin > mqscout
```

Alternatively, enter this form of `runmqsc` command at a TACL command prompt:

```
runmqsc /IN mqscin, OUT mqscout/ saturn.queue.manager
```

MQSC command files
An MQSC command file contains MQSC commands, which are written as text in a human readable format. Therefore, in an OSS environment, an MQSC command file is an ASCII text file and, in the NonStop OS environment, it is an EDIT file.

Chapter 4. Administering local WebSphere MQ objects 35
Figure 2 is an extract from an MQSC command file showing an MQSC command (DEFINE QLOCAL) with its parameters. 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;
```

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

**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 WebSphere MQ Commands.

- An optional numbered listing of the MQSC commands issued. By default, the text of the input is echoed to the output. Within this output, each command is prefixed by a sequence number, as shown in Figure 3 on page 37. However, you can use the `-e` parameter 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.
Running the supplied MQSC command file

The MQSC command file amqscos0.tst is supplied with WebSphere MQ and contains the MQSC commands to create the objects used by the sample programs. This file is in the directory opt_installation_path/opt/mqm/samp. To run the MQSC commands against the default queue manager, enter the following command at an OSS shell command prompt:

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

Using runmqsc to verify commands

You can use the -v parameter on the runmqsc command to verify MQSC commands on a local queue manager without actually running them. For example, to verify the MQSC commands in the file mqscin using the default queue manager, enter the following command at an OSS shell command prompt:

```
runmqsc -v < mqscin > mqscout
```

Alternatively, enter the following command at a TACL command prompt:

```
runmqsc /IN mqscin, OUT mqscout/ -v
```

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.

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:
Problems with MQSC commands

- If you enter the command at an OSS shell command prompt and redirect the input to come from an MQSC command file, don’t forget to include the < operator. 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 enter the command at an OSS shell command prompt and redirect the output to a file, specifying an unqualified file name, the file is put in the current working directory. If you enter the command at a TACL command prompt, the file is put in the default subvolume instead. Specify a fully qualified file name to send the output to a different directory or subvolume.
- Check that you have created the queue manager that is going to run the commands.
  To do this, look in the WebSphere MQ configuration file, mqs.ini. This file contains the names of all the queue managers in the installation and identifies the default queue manager, if you have one.
- The queue manager must be running. If it is not, start it (see “Starting a queue manager” on page 29). You get an error message if you try to start a queue manager that is already running.
- Specify the name of a queue manager on the runmqsc command if you have not defined a default queue manager, or you receive this error message:
  AMQ8146: WebSphere MQ queue manager not available.
- You cannot specify an MQSC command as a parameter on the runmqsc command. For example, the following command is not valid:
  runmqsc DEFINE QLOCAL(FRED)
- You cannot enter an MQSC command directly at an OSS shell or TACL command prompt. You must use the runmqsc control command.
- You cannot use runmqsc to run other control commands. For example, you cannot issue the strmqm command to start a queue manager while you are running MQSC commands interactively. If you try to do this, you receive error messages similar to the following:
  runmqsc
  .
  Starting WebSphere MQ Commands.

  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 4.

```
MQSC >DISPLAY QMGR
10 : DISPLAY QMGR
AMQ8408: Display Queue Manager details.
  DESC( )    DEADQ( )
  DEFINE( )  CHADEXIT( )
  CLWLEXIT( ) CLWLDATA( )
  REPOS( )   REPOSNL( )
  SSLKEYR(/var/mqm/qmgrs/HHQM1/ssl)
  SSLCRNL( ) SSLCRYP( )
  COMMANDQ(SYSTEM.ADMIN.COMMAND.QUEUE)
  QMNAME(ACCOUNTS)
  CRDATE(2005-10-06)
  CRTIME(10.18.32)
  ALTDATE(2005-10-06)
  ALTTIME(10.18.32)
  QMID(HHQM1_2005-10-06_10.18.32)
  MAXHANDS(256)
  MAXUMSGS(10000)
  AUTHOREV(DISABLED)
  INHIBTEV(DISABLED)
  LOCALEV(DISABLED)
  REMOTEENV(DISABLED)
  PERFMEV(DISABLED)
  STRSTPEV(ENABLED)
  CHAD(DISABLED)
  CHADEV(DISABLED)
  CLWLEN(100)
  MAXMSG(4194304)
  CCSID(819)
  MAXPRTY(9)
  CMDLEVEL(530)
  PLATFORM(13)
  SYNCPT
```

Figure 4. Typical output from a DISPLAY QMGR command

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

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

DISPLAY QUEUE(SYSTEM.*)

This displays a list of queues that match the stem SYSTEM.*, The parentheses are required.
Problems with MQSC commands

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, disabled for puts, and operates on a first-in-first-out (FIFO) basis.
- It is an ordinary queue; it is not an initiation queue or transmission queue, and it does not generate trigger messages.
- The maximum queue depth is 1000 messages; the maximum message length is 2000 bytes.

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

Notes:

1. Most of these attributes are the defaults as supplied with the product. 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 the attributes of a queue” on page 41.

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 the attributes of a local queue" on page 42.

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 using the -u parameter on the crtmqm command, or by using the DEADQ parameter on the ALTER QMGR command to specify one later. You must create the dead letter queue before using it.

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

A dead letter queue has no special requirements except that:
• It must be a local queue
• Its MAXMSGL (maximum message length) attribute must enable the queue to accommodate the largest messages that the queue manager has to handle plus the size of the dead letter header (MQDLH)

WebSphere MQ provides a dead letter queue handler that allows you to specify how messages found on a dead letter queue are to be processed or removed. For more information, see Chapter 13, "The WebSphere MQ dead-letter queue handler," on page 187

Displaying the attributes of a queue
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)     MAXDEPTH(5000)
  MAXMSGL(4194304)               CURDEPTH(0)
  5 : end

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

Copying a local queue

You can copy a queue 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 the 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, but changing one or more attributes of the original. For example:

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

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

Notes:

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

Changing the attributes of a local queue

You can change the attributes of a local queue in two ways, using either the ALTER QLOCAL command or the DEFINE QLOCAL command with the REPLACE parameter. In “Defining a local queue” on page 40, we defined a queue called ORANGE.LOCAL.QUEUE. Suppose, for example, you want to increase the maximum message length on this queue to 10000 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 parameter:
  ```
  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
Enter the messages are lost.

You cannot clear a queue if:

- There are uncommitted messages that have been put on the queue under
  syncpoint
- An application currently has the queue open

Deleting a local queue

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

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

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

Browsing queues

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.

The default file names and paths are:

**Source**

```
opt_installation_path/opt/mqm/samp/amqsbcg0.c
```

**Executable**

```
opt_installation_path/opt/mqm/samp/bin/amqsbcg
```

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

```
amqsbcg TEST_Q saturn.queue.manager
```

Typical results from this command are shown in Figure 5 on page 44.
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 an initiation queue" on page 48)
- A local definition of a remote queue (see "Creating a local definition of a remote queue" on page 68)

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.

Figure 5. Typical results from queue browser
When an application issues 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 calls MQOPEN 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)
```
Working with alias queues

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 the attributes of an alias queue, or to delete the alias queue object. See the following examples.

Use the following command to display the attributes of an alias queue:

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

Use the following command to alter the base queue name to which an alias resolves. The FORCE parameter 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 an alias queue:

```
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 MQOPEN 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 dynamic queues are maintained across queue manager restarts, temporary dynamic queues are not. For example:

```
DEFINE QMODEL(GREEN.MODEL.QUEUE) +
  DESCR('Queue for messages from application X') +
  PUT(DISABLED) +
  GET(ENABLED) +
  NOTRIGGER +
  MSGDLVSQ(FIFO) +
```
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 SYSTEM.DEFAULT.MODEL.QUEUE default queue.

You can use the LIKE and REPLACE parameters 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. See the following examples.

Use the following command to display the attributes of a model queue:

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

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

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

Use the following command to delete a model queue:

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

### Managing objects for triggering

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

This section tells you how to set up the required objects to support triggering in 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 for the application queue. Triggering itself is enabled when the `TriggerControl` attribute is set to MQTC.ON. The attribute is set to this value by including the TRIGGER parameter in an MQSC command such as `DEFINE QLOCAL`.

As an example, the following command creates an application queue called MOTOR.INSURANCE.QUEUE which causes a trigger event when there are 100 messages of priority 5 or greater in the queue:

```
DEFINE QLOCAL(MOTOR.INSURANCE.QUEUE) +
    PROCESS(MOTOR.INSURANCE.QUOTE.PROCESS) +
    MAXMSGL(2000) +
    DEFPERSIST(Y) +
    INITQ(MOTOR.INS.INIT.QUEUE) +
```
Managing objects for triggering

TRIGGER +
TRIGTYPE(DEPTH) +
TRIGDPTH(100)+
TRIGMPRI(5)

The parameters have the following meanings:

MOTOR.INSURANCE.QUEUE
The name of the application queue being created

PROCESS(MOTOR.INSURANCE.QUOTE.PROCESS)
Identifies the application to be started by a trigger monitor

MAXMSGL(2000)
The maximum permitted length of a message on the queue

DEFPSIST(YES)
Whether the messages on the queue are persistent by default

INITQ(MOTOR.INS.INIT.QUEUE)
The name of the initiation queue in which the queue manager puts trigger messages

TRIGGER
Enables triggering

TRIGTYPE(DEPTH)
Specifies that a trigger event is generated when the number of messages with the required priority, as specified by the TRIGPRI parameter, reaches the value specified by the TRIGDPTH parameter.

TRIGDPTH(100)
The number of messages required to generate a trigger event.

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

Defining an initiation queue

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

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

Creating a process definition

Use the DEFINE PROCESS command to create a process definition. A process definition identifies the application to be used to process messages in the application queue. The application can be either an OSS application or a NonStop OS application. The definition of the application queue names the process to be used, and therefore associates the application queue with the application to be used to process its messages.

The following MQSC command creates a process definition called MOTOR.INSURANCE.QUOTE.PROCESS:
DEFINE PROCESS(MOTOR.INSURANCE.QUOTE.PROCESS) +
  DESCR('Insurance request message processing') +
  APPLTYPE(UNIX) +
  APPLICID('/u/admin/test/IRMP01') +
  USERDATA('open, close, 235')

The parameters have the following meanings:

**MOTOR.INSURANCE.QUOTE.PROCESS**
The name of the process definition.

**DESCR('Insurance request message processing')**
A description of the application 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)**
The type of application to be started. For an OSS application, specify APPLTYPE(UNIX). For a NonStop OS application, specify APPLTYPE(NSK).

**APPLICID('/u/admin/test/IRMP01')**
The fully qualified name of the application executable file. This is the format of the parameter for an OSS application. For a NonStop OS application, the parameter must specify a fully qualified local name with the format `volume_name.subvolume_name.file_name`; for example, APPLICID('DATA01.APPL.COMP').

**USERDATA('open, close, 235')**
User defined data that can be used by the application.

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.

**Displaying the attributes of a process definition**
Use the DISPLAY PROCESS command to display the attributes of a process definition. See the following example:

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

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

**Using the Monitoring Panels**
The Monitoring Panels facility of WebSphere MQ for HP NonStop Server runs as a Pathway SCOBOL requester under the Terminal Control Process (TCP). It uses a Monitoring Panels server process. Because every queue manager has its own Pathway configuration, each queue manager has its own instance of the Monitoring Panels. Consequently, you can use an instance of the Monitoring Panels to monitor only the queue manager to which it belongs.
Monitoring Panels

By default, no more than 10 users can use an instance of the Monitoring Panels concurrently. To change this limit to 20, for example, enter the following command at the PATHCOM prompt of the queue manager:

```
ALTER TCP MQS-TCP-01, MAXTERMS 20
```

To start the Monitoring Panels, enter `run mqmc` at the PATHCOM prompt of the queue manager. The Main Menu is displayed:

```
IBM WebSphere MQ for HP NonStop Server Version 5.3

** Main Menu **

Enter Choice: _
1. Queue Manager
2. Queues
3. Channels

F1 - Enter               F16 - Return

5724-A39 (C) Copyright IBM Corp. 1993, 2005 All Rights Reserved.
```

*Figure 6. The Monitoring Panels Main Menu*

You can select the following options from the Main Menu:
1. Queue Manager
2. Queues
3. Channels

The following sections describe each of these options.

At any time while you are using the Monitoring Panels, you can return to the Main Menu by pressing Alt+F6, or return to the previous panel by pressing the Return function key (F16). To leave the Monitoring Panels, press F16 from the Main Menu.

**Using the Queue Manager Menu**

To select the Queue Manager option from the Main Menu, type 1 in the **Enter Choice** field and then press the Enter function key (F1). The first panel of the Queue Manager Menu is displayed:
You can use the Queue Manager Menu to inspect the attributes of the queue manager. To display the second panel, press the Page Down key:

** Queue Manager Menu **

```
Name : JOBS
Description :
Command Level : 530  Trigger Interval : 999999999
Coded Char Set : 819  Platform : NSK
Max Handles : 256  Max Uncommitted Msg: 10000
Max Message : 4194304  Max Priority : 9
Dead Letter Queue Name : DEADQ
Command Input Queue Name : SYSTEM.ADMIN.COMMAND.QUEUE
Default Xmit Queue Name :
Authority Event Enabled : N  Inhibit Event Enabled : N
Local Event Enabled : N  Remote Event Enabled : N
Start/Stop Event Enabled : Y  Performance Event Enabled : N
```

Figure 7. Queue Manager Menu, panel 1

Using the Queue Menu

To select the Queues option from the Main Menu, type 2 in the Enter Choice field and then press the Enter function key (F1). The Search Criteria panel for queues is displayed:

** Queue Manager Menu **

```
Queue Manager Id : JOBS_2005-11-08_15.43.12
Channel Auto Definition: N  Channel Auto Definition Events Enabled : N
Auto Definition Exit :
Cluster Workload Data :
Cluster Workload Exit :
Cluster Workload Length: 100  Distribution List Support: Y
Repository Name :
Repository Name List :
```

Figure 8. Queue Manager Menu, panel 2
To select which queues to display on the Queue Menu, type a partial or complete queue name in the Queue Name field of the Search Criteria panel. You can also select a queue type to limit your search to queues of just one type. Then press the Enter function key (F1). The Queue Menu is displayed:

The Queue Menu lists the names of the queues that satisfy the search criteria. If the list is too large to fit on one panel, you can use the Page Down and Page Up keys to scroll through the list.

You can use the Queue Menu to perform the following tasks:

- Display the attributes of a queue that is listed on the menu
- Monitor the local queues
Displaying the attributes of a queue

To display the attributes of a queue that is listed on the Queue Menu, first select the queue by typing any character in the input field opposite the name of the queue, and then press the Display function key (F1). The panel that is displayed depends on the type of queue selected. If you select a local queue, the first Display Local Queue panel is displayed:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Queue Name</td>
<td>QL.FROM.LAPTOP</td>
</tr>
<tr>
<td>Description</td>
<td>WebSphere MQ Default Local Queue</td>
</tr>
<tr>
<td>Default Msg Priority</td>
<td>0</td>
</tr>
<tr>
<td>Put Enabled</td>
<td>Y</td>
</tr>
<tr>
<td>Default Persistence</td>
<td>N</td>
</tr>
<tr>
<td>Get Enabled</td>
<td>Y</td>
</tr>
<tr>
<td>Retention Interval</td>
<td>999999999</td>
</tr>
<tr>
<td>Queue Definition Type</td>
<td>PREDEFINED</td>
</tr>
<tr>
<td>Max Queue Depth</td>
<td>5000</td>
</tr>
<tr>
<td>Priority/FIFO [P/F]</td>
<td>P</td>
</tr>
<tr>
<td>Max Message Length</td>
<td>4194304</td>
</tr>
<tr>
<td>Share</td>
<td>Y</td>
</tr>
<tr>
<td>Backout Threshold</td>
<td>0</td>
</tr>
<tr>
<td>Usage [N/X]</td>
<td>N</td>
</tr>
<tr>
<td>Backout Requeue Name</td>
<td></td>
</tr>
<tr>
<td>Init. Queue</td>
<td></td>
</tr>
<tr>
<td>Process Name</td>
<td></td>
</tr>
<tr>
<td>Trigger Type [N/E/F/D]</td>
<td>F</td>
</tr>
<tr>
<td>Trigger/NoTrigger</td>
<td>N</td>
</tr>
<tr>
<td>Trigger Depth</td>
<td>1</td>
</tr>
<tr>
<td>Trigger Priority</td>
<td>0</td>
</tr>
<tr>
<td>Q Depth Max Event</td>
<td>Y</td>
</tr>
<tr>
<td>Q Serv. Int. Event[H/O/N]</td>
<td>N</td>
</tr>
<tr>
<td>Q Depth High Limit</td>
<td>80</td>
</tr>
<tr>
<td>Q Depth High Event</td>
<td>N</td>
</tr>
<tr>
<td>Q Depth Low Limit</td>
<td>20</td>
</tr>
<tr>
<td>Q Depth Low Event</td>
<td>N</td>
</tr>
<tr>
<td>Q Service Interval</td>
<td>999999999</td>
</tr>
<tr>
<td>Scope</td>
<td>QMGR</td>
</tr>
<tr>
<td>PGDN - Next Page</td>
<td>F16 - Return</td>
</tr>
</tbody>
</table>

Figure 11. Display Local Queue, panel 1

To display the second panel, press the Page Down key:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cluster Name</td>
<td></td>
</tr>
<tr>
<td>Cluster Name List</td>
<td></td>
</tr>
<tr>
<td>Distribution List</td>
<td>N</td>
</tr>
<tr>
<td>Default Binding [O/N]</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 12. Display Local Queue, panel 2

There are similar panels for displaying the attributes of a model queue, a local definition of a remote queue, and an alias queue.
**Monitoring local queues**

To monitor the local queues, press the Monitor function key (F5) from the Queue Menu. The Monitor Local Queues panel is displayed:

```
<table>
<thead>
<tr>
<th>Queue</th>
<th>OPEN</th>
<th>INPUT</th>
<th>OPEN</th>
<th>OUTPUT</th>
<th>DEPTH</th>
</tr>
</thead>
<tbody>
<tr>
<td>QL.FROM.LAPTOP</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
<td>37</td>
</tr>
<tr>
<td>SYSTEM.ADMIN.CHANNEL.EVENT</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SYSTEM.ADMIN.COMMAND.QUEUE</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SYSTEM.ADMIN.PRFM.EVENT</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SYSTEM.ADMIN.QMGR.EVENT</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SYSTEM.AUTH.DATA.QUEUE</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SYSTEM.CHANNEL.INITQ</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SYSTEM.CHANNEL.SYNCQ</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SYSTEM.CLUSTER.COMMAND.QUEUE</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SYSTEM.CLUSTER.REPOSITORY.QUEUE</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SYSTEM.CLUSTER.TRANSMIT.QUEUE</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SYSTEM.DEAD.LETTER.QUEUE</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SYSTEM.DEFAULT.INITIATION.QUEUE</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

F12 - Refresh  PGDN  PGUP  F16 - Return

**Figure 13. Monitor Local Queues**

The Monitor Local Queues panel lists the names of all the local queues. If the list is too large to fit on one panel, you can use the Page Down and Page Up keys to scroll through the list.

For each local queue, the panel displays the following information:

- The number of applications that have the queue open for input
- The number of applications that have the queue open for output
- The number of messages in the queue

The MQMQMREFRESHINT attribute of the Pathway server class MQS-MQMSVR00 determines the frequency with which the Monitor Local Queues panel is refreshed. By default, the panel is refreshed every 30 seconds. To change the frequency to every 10 seconds, for example, enter the following command at the PATHCOM prompt of the queue manager:

```
ALTER SERVER MQS-MQMSVR00, PARAM MQMQMREFRESHINT 10
```

**Using the Channel Menu**

To select the Channels option from the Main Menu, type 3 in the Enter Choice field and then press the Enter function key (F1). The Search Criteria panel for channels is displayed:
To select which channels to display on the Channel Menu, type a partial or complete channel name in the **Channel Name** field of the Search Criteria panel. You can also select a channel type to limit your search to channels of just one type. Then press the Enter function key (F1). The Channel Menu is displayed:

<table>
<thead>
<tr>
<th>Channel Name</th>
<th>TYPE</th>
<th>STATUS</th>
</tr>
</thead>
<tbody>
<tr>
<td>FROM.LAPTOP</td>
<td>RECEIVER</td>
<td></td>
</tr>
<tr>
<td>SYSTEM.AUTO.RECEIVER</td>
<td>RECEIVER</td>
<td></td>
</tr>
<tr>
<td>SYSTEM.AUTO.SVRCONN</td>
<td>SVRCONN</td>
<td></td>
</tr>
<tr>
<td>SYSTEM.DEF.CLUSRCVR</td>
<td>CLUSRCVR</td>
<td></td>
</tr>
<tr>
<td>SYSTEM.DEF.CLUSDDR</td>
<td>CLUSDDR</td>
<td></td>
</tr>
<tr>
<td>SYSTEM.DEF.RECEIVER</td>
<td>RECEIVER</td>
<td></td>
</tr>
<tr>
<td>SYSTEM.DEF.REQUESTER</td>
<td>REQUESTER</td>
<td></td>
</tr>
<tr>
<td>SYSTEM.DEF.SENDER</td>
<td>SENDER</td>
<td></td>
</tr>
<tr>
<td>SYSTEM.DEF.SERVER</td>
<td>SERVER</td>
<td></td>
</tr>
<tr>
<td>SYSTEM.DEF.SVRCONN</td>
<td>SVRCONN</td>
<td></td>
</tr>
<tr>
<td>TO.LAPTOP</td>
<td>SENDER</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 15. Channel Menu**

The Channel Menu lists the names of the channels that satisfy the search criteria. If the list is too large to fit on one panel, you can use the Page Down and Page Up keys to scroll through the list.

You can use the Channel Menu to perform the following tasks:
- Display the attributes of a channel that is listed on the menu
- Monitor the current channels
**Monitoring Panels**

### Displaying the attributes of a channel
To display the attributes of a channel that is listed on the Channel Menu, first select the channel by typing any character in the input field opposite the name of the channel, and then press the Display function key (F1). The panel that is displayed depends on the type of channel selected. If you select a sender channel, the first Display Sender Channel panel is displayed:

![Display Sender Channel, panel 1](image)

**Figure 16. Display Sender Channel, panel 1**

To display the second panel, press the Page Down key:

![Display Sender Channel, panel 2](image)

**Figure 17. Display Sender Channel, panel 2**

There are similar panels for displaying the attributes of a server channel, a receiver channel, a requester channel, a server connection channel, a cluster sender channel, and a cluster receiver channel.
Monitoring current channels

To monitor the current channels, press the Monitor function key (F5) from the Channel Menu. The Monitor Channels panel is displayed:

<table>
<thead>
<tr>
<th>Channel Name</th>
<th>Status</th>
<th>Curr MSN</th>
<th>Last MSN</th>
<th>MCA Status</th>
<th>Stop</th>
</tr>
</thead>
<tbody>
<tr>
<td>REG2_REG1_SORC_3000</td>
<td>RUNNING</td>
<td>1000</td>
<td>1000</td>
<td>RUNNING</td>
<td>NO</td>
</tr>
<tr>
<td>REG1_REG2 SVRQ_3000</td>
<td>RUNNING</td>
<td>500</td>
<td>500</td>
<td>RUNNING</td>
<td>NO</td>
</tr>
<tr>
<td>REG1_REG2 SORC_3000</td>
<td>RUNNING</td>
<td>1000</td>
<td>1000</td>
<td>RUNNING</td>
<td>NO</td>
</tr>
</tbody>
</table>

F12 - Refresh        PGDN   PGUP   F16 - Return

Figure 18. Monitor Channels

The Monitor Channels panel lists the names of all current channels. If the list is too large to fit on one panel, you can use the Page Down and Page Up keys to scroll through the list.

For each channel, the panel displays the following information:

- The state of the channel
- The message sequence number of the last message sent or received
- The message sequence number of the last message in the last committed batch of messages
- Whether an MCA is running
- Whether an administrator has issued a request to stop the channel

The frequency with which the Monitor Channels panel is refreshed is determined by the same mechanism that determines the frequency with which the Monitor Local Queues panel is refreshed. For more details, see "Monitoring local queues" on page 54.
Chapter 5. Automating administration tasks

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 introduces the PCF commands. It assumes that you have experience of administering WebSphere MQ objects.

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 you can create queues, process definitions, channels, and namelists, and change queue managers, from a program.

PCF commands cover the same range of functions provided by 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 call MQPUT in the same way as any other message. The command server on the queue manager receiving the message interprets it as a command message and runs the command. To get the replies, the application issues an MQGET call and the reply data is returned in another data structure. The application can then process the reply and act accordingly. For information about how to use the command server, see "Managing the command server for remote administration" on page 66.

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

Briefly, these are some of the things needed to create a PCF command message:

Message descriptor
This is a standard WebSphere MQ message descriptor, in which:
- Message type (MsgType) is MQMT_REQUEST.
- Message format (Format) is MQFMT_ADMIN.

Application data
Contains the PCF message including the PCF header, in which:
- The PCF message type (Type) specifies MQCFT_COMMAND.
- The command identifier specifies the command, for example, Change Queue (MQCMD_CHANGE_Q).

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

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. 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 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.
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 63
- “Creating a local definition of a remote queue” on page 68
- “Using remote queue definitions as aliases” on page 71
- “Data conversion” on page 72

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 system
- On another system 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 or the MQAI
- 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 listener
- A channel initiator

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

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

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 "Preparing channels and transmission queues for remote administration" on page 139 for details of how to configure a channel to use pipelining.

"Preparing channels and transmission queues for remote administration" on page 64 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 queue manager 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 queue manager cluster is a group of queue managers set up in such a way that the queue managers can communicate directly with one another over a single network without complex transmission queue, channel, and queue definitions. Clusters can be set up easily, and typically contain queue managers that are logically related in some way and need to share data or applications. Even the smallest cluster reduces system administration overheads.

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

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

For more information about clusters, their attributes, and how to set them up, refer to 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 19 shows the configuration of queue managers and channels that you need for remote administration using the runmqsc command. source.queue.manager is the name of the source queue manager from which you can issue MQSC commands and to which the results of these commands (operator messages) are returned. 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 parameter, source.queue.manager must be the default queue manager. For further information about creating a queue manager, see “crtmqm (create queue manager)” on page 253.

Figure 19. Remote administration using MQSC commands

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

- Create the queue manager and the system and 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.
- Start the command server using the `strmqcsv` command.

You must 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 name of the transmission queue must match the name of the remote queue manager in order for remote administration to work, unless you are using a queue manager alias. Figure 20 summarizes this configuration.

```
runmqsc
commands
replies
XMITQ=target.queue.manager
SYSTEM.MQSC.REPLY.QUEUE

source.queue.manager

source.to.target

target.to.source

SYSTEM.ADMIN.COMMAND.QUEUE
XMITQ=source.queue.manager

target.queue.manager

Local system

Remote system
```

Figure 20. Setting up channels and queues for remote administration

See [WebSphere MQ Intercommunication](#) for more information about setting up channels.

**Defining channels and transmission queues**

On the source queue manager, issue the following MQSC commands to create the channels and the transmission queue:

1. Create the sender channel at the source queue manager:
DEFINE CHANNEL('source.to.target') + 
   CHLTYPE(SDR) + 
   CONNAME(RHX5498) + 
   XMITQ('target.queue.manager') + 
   TRPTYPE(TCP)

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

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

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

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

On the target queue manager, issue the following MQSC commands to create the channels and the transmission queue:

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

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

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

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

3. Create 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 channels

First start a listener at the receiving end of each channel:

- For the source queue manager, enter:
  runmqslr -t TCP -m source.queue.manager

- For the target queue manager, enter:
  runmqslr -t TCP -m target.queue.manager

Then, at the sending end of each channel, start the channel as a background process:

- For the source queue manager, enter the following command at an OSS shell command prompt:
  runmqchl -c source.to.target -m source.queue.manager &

  Alternatively, enter the following command at a TACL command prompt:
  run /NOWAIT/ runmqchl -c source.to.target -m source.queue.manager

- For the target queue manager, enter the following command at an OSS shell command prompt:
  runmqchl -c target.to.source -m target.queue.manager &

  Alternatively, enter the following command at a TACL command prompt:
  run /NOWAIT/ runmqchl -c target.to.source -m target.queue.manager
Remote administration

A TCP/IP listener can also be started as a server process that is configured as a server class within Pathway. You can use the MQSC command START LISTENER to start the default TCP/IP listener.

You can use the MQSC command START CHANNEL to start a channel instead of using the runmqchl control command.

**Automatic definition of channels:** If WebSphere MQ receives an inbound attach request and cannot find an appropriate receiver or server-connection definition in the channel definition file (CDF), it creates a definition automatically and adds it to the CDF. 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.

**Starting the command server**

To start the command server, use 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 called saturn.queue.manager, the command is:
You must issue this command on the target machine. If the command server is running, the following message is returned:

AMQ8027  WebSphere MQ Command Server Status ..: Running

Stopping the 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 parameter, which is the default.
- For an immediate stop, use the endmqcsv command with the -i parameter.

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

Issuing MQSC commands to 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.)

- On the target queue manager, enter:
  
  strmqcsv target.queue.manager

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

  runmqsc -w 30 target.queue.manager

This form of the runmqsc command, with the -w parameter, runs the MQSC commands in indirect mode, where the commands are put (in a modified form) on the command queue of the target queue manager and run 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, you can enter the following command at an OSS shell command prompt:

runmqsc -w 60 target.queue.manager < mqscin > mqscout

Alternatively, you can enter the following command at a TACL command prompt:

runmqsc /IN mqscin, OUT mqscout/ -w 60 target.queue.manager

In these commands, mqscin is the file containing the MQSC commands and mqscout is the report file.
**Command server remote administration**

**Working with queue managers on z/OS**
You can issue MQSC commands to a z/OS queue manager from a queue manager running on NonStop OS. 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 on 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 (CONNAME) in the channel definition.
- Started the listeners before you started the message channels.
- Checked that the disconnect interval has not expired, for example, if a channel started but then shut down after some time. This is especially important if you start the channels manually.
- 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 37](#).

---

**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 name 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:

<table>
<thead>
<tr>
<th>Field value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ObjectName</strong></td>
<td>CYAN.REMOTE.QUEUE</td>
</tr>
<tr>
<td><strong>ObjectType</strong></td>
<td>(Queue)</td>
</tr>
<tr>
<td><strong>ObjectQmgrName</strong></td>
<td>Blank or saturn.queue.manager</td>
</tr>
</tbody>
</table>

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 command:

```sql
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)
```

The parameters have the following meanings:

**CYAN.REMOTE.QUEUE**

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

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

Additional text that describes the use of the queue.

**RNAME(AUTOMOBILE.INSURANCE.QUOTE.QUEUE)**

The name of the target queue on the remote queue manager. This is the
Local definition of remote queue

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

XMITQ(INQUOTE.XMIT.QUEUE)
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.
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 specified reply-to queue name is an alias.
**Aliases**

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

For more information about request messages, reply messages, and reply-to queues, see the [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 135](#).

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” on page 73](#).

**The file ccsid.tbl**

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

- It records all the supported code sets.
- 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.

The file ccsid.tbl is in the directory `var_installation_path/var/mqm/conv/table`. 
Data conversion

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.

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 239) 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's CCSID
When you have used the CCSID parameter 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. Managing WebSphere MQ for HP NonStop Server

Chapter 7. WebSphere MQ for HP NonStop

Server architecture ............................................ 77
An overview of the queue manager processes ........... 77
The execution controller ................................. 79
Queue servers ............................................. 81
The channel server ........................................ 82
The remaining server processes that are configured within Pathway .......... 83
Control commands and the Pathway configuration of a queue manager ........ 83
The product code and files ................................ 84

Chapter 8. Managing scalability, performance, availability, and data integrity ........................................ 85

Introduction to scalability and performance ............. 85
Designing new applications for performance and scalability ........................................ 85
Designing to minimize or eliminate the use of shared resources ...................... 86
Performance tuning is inherently iterative ......... 86
Message persistence .................................... 86
Persistent messages .................................... 87
Nonpersistent messages .................................. 87
Nonpersistent messages and channels .............. 87
Queue servers and queue files ........................................ 88
How persistent messages are stored .............. 88
How nonpersistent messages are stored .......... 89
Queue server CPU distribution ...................... 89
Reassigning a WebSphere MQ object to another queue server .............. 89
Cluster transmission queue: SYSTEM.CLUSTER.TRANSMIT.QUEUE ... 90
Moving the files associated with a local queue ...... 90
Increasing the capacity of a local queue .......... 91
Partitioning a queue file or queue overflow file .... 92
Message overflow files .................................. 92
Browsing persistent messages ..................... 93
The Measure counter .................................... 93
The queue server options .................................... 94
Checkpoint nonpersistent messages .......... 94
Lock in memory ....................................... 94
Audited message overflow files ...................... 94
Load at startup ....................................... 94
Putting a local queue into maintenance mode ..... 95
CPU assignment ........................................ 95
Fastpath binding application programs .............. 96
Background ........................................... 97
Reducing MQI overhead .................................. 97
Enabling fastpath binding ......................... 97
Restrictions when using fastpath binding .......... 97
Data integrity ........................................ 98
Availability ........................................ 99
Persistent and nonpersistent data ................. 100
Persistent messages .................................. 100
Nonpersistent messages .................................. 102
Database consistency .................................. 102

Internal database consistency ...................... 102
External database consistency ................... 103
OpenTMF .......................................... 104
NonStop Tuxedo .................................. 104
Interleaved application transactions ............. 104
WebSphere MQ critical database files .......... 105
Critical processes .................................... 105
Queue manager clusters ......................... 110
Configuration considerations for availability .... 111
Configuration considerations for data integrity ..... 111
Chapter 7. WebSphere MQ for HP NonStop Server architecture

This chapter describes the architecture of WebSphere MQ for HP NonStop Server. It contains the following sections:

- “An overview of the queue manager processes”
- “The execution controller” on page 79
- “Queue servers” on page 81
- “The channel server” on page 82
- “The remaining server processes that are configured within Pathway” on page 83
- “Control commands and the Pathway configuration of a queue manager” on page 83
- “The product code and files” on page 84

This information can help you to prepare the optimal configuration for WebSphere MQ for HP NonStop Server in your operating environment.

Chapter 8, “Managing scalability, performance, availability, and data integrity,” on page 85 provides specific configuration guidance in the context of this architectural information.

An overview of the queue manager processes

Figure 21 on page 78 shows the server processes of a queue manager and the interactions between the processes. In the diagram, the Pathway manager process, PATHMON, is shown for completeness because some server processes of a queue manager run within a Pathway environment. PATHMON starts and manages one server process or process-pair for each server class in the Pathway configuration of the queue manager. However, not all the server processes are started when you start the queue manager. PATHMON starts a server process only when it is required.
The following server processes are configured as server classes within Pathway. When you create a queue manager, a default Pathway configuration for the queue manager is created automatically. The default configuration contains only one server class for each type of server process, which means that only one server process of each type is started. However, for certain types of server process, as indicated in the following list, you can cause additional server processes to be started by configuring additional server classes.
Queue manager processes

Execution controller
An execution controller is a fault-tolerant process-pair. A queue manager can have only one execution controller.

Queue server
A queue server is a fault-tolerant process-pair. By default, a queue manager has one queue server, but you can configure more.

Channel server
An channel server is a fault-tolerant process-pair. A queue manager can have only one channel server.

Channel initiator
A channel initiator is an OSS process. By default, a queue manager has one channel initiator, but you can configure more.

Command server
A command server is an OSS process. A queue manager can have only one command server.

Monitoring Panels server
A Monitoring Panels server is an OSS process. By default, a queue manager has one Monitoring Panels server, but you can configure more.

Queue manager server
A queue manager server is an OSS process. A queue manager can have only one queue manager server.

TCP/IP listener
A TCP/IP listener is an OSS process. By default, a queue manager has one TCP/IP listener, but you can configure more.

Trigger monitor
A trigger monitor is an OSS process. By default, a queue manager has one trigger monitor, but you can configure more.

Deferred message server
A deferred message server is an OSS process. A queue manager can have only one deferred message server.

The execution controller, queue servers, and channel server are the most critical components of the queue manager because all the other components depend on them. As a result, these server processes are implemented as fault-tolerant process-pairs and are referred to as the key server processes. The remaining server processes provide less critical, though important, function and are not therefore implemented as fault-tolerant process-pairs.

The execution controller
The execution controller coordinates all the work performed by the queue manager. The execution controller receives and responds to requests from applications to connect to the queue manager, and manages the resources required to maintain the connections. The execution controller also receives and responds to requests to start channels, and manages the resources consumed by active channels.

The execution controller manages the local queue manager agents, which are the server processes that process MQI calls on behalf of applications, and message channel agents (MCAs), which are the server processes that implement channels. The execution controller also manages the repository managers, which are the
server processes that manage the queue manager cluster repository, and the Object Authority Manager (OAM) servers, which are the server processes that are responsible for access control to WebSphere MQ objects.

The server processes managed by the execution controller are not configured as server classes in Pathway. The execution controller uses the rules in the process management rules configuration file, qmproc.ini, to manage them. The execution controller uses the rules to name a process, to determine the CPU in which a process runs, and to set other attributes of a process. The configuration file contains rules that apply to all processes, to classes of processes, and to specific processes.

When you create a queue manager, the contents of its process management rules configuration file are derived from the default process management rules configuration file for the installation. This configuration file is called proc.ini and is in the directory var_installation_path/var/mqm. The default process management rules configuration file is created when you install WebSphere MQ for HP NonStop Server, and contains a default set of rules, which you can customize to meet the requirements of all the queue managers created in the installation.

You can change the contents of the process management rules configuration file of a queue manager at any time using a text editor. The execution controller reads and processes the rules only when you restart the queue manager or issue the MQSC command RESET QMGR TYPE(NSPRC), or the equivalent PCF command. If the execution controller detects a problem with the rules, the entire contents of the configuration file are ignored and an operator message is issued to indicate the problem. The execution controller continues by using a set of safe default rules.

If an incorrect entry is encountered in a process management rules configuration file,

For more information about the process management rules configuration file, see Chapter 14, “Process management,” on page 197.

A local queue manager agent processes MQI calls on behalf of an application. The primary purpose of a local queue manager agent is to protect the critical resources of the queue manager from an errant or malicious application. The number of local queue manager agents depends on the number of connected applications and the rules specified in the process management rules configuration file. A local queue manager agent can run as a process, or as a thread within a process. The rules in the process management rules configuration file specify which connections must be handled by a local queue manager agent running as a process, and which must be handled by a local queue manager agent running as a thread. The execution controller maintains a pool of local queue manager agents that are ready to receive connection requests from applications.

A message channel agent (MCA) implements one end of a message channel, or the server connection end of an MQI channel. An MCA can run as a process, or as a thread within a process, depending on the rules in the process management rules configuration file, the definition of the channel, or the command used to start the listener. The execution controller maintains a pool of MCAs that are ready to implement channels.
Queue servers

A queue server performs all the messaging operations on the local queues for which it is responsible. Each local queue is assigned to one queue server only. When you create a queue manager, it has only one queue server called the default queue server. When you create a local queue, it is assigned to the default queue server. To store very large messages, or a very large number of messages, or to handle a very high throughput of messages, you might need to configure additional queue servers and reassign local queues to the new queue servers. In extreme cases, you might need to dedicate a queue server to a single queue.

A queue server also manages the internal status of WebSphere MQ objects, except channels. As is the case for a local queue, when an object is created, it is assigned to the default queue server, but the object can be reassigned subsequently to another queue server. The queue server needs to manage the status of objects so that any changes in the attributes of an object that affect messaging operations can be detected and acted upon.

A queue server is responsible for the way messages are stored in the local queues that it manages. Persistent messages are stored in files in the NonStop OS file system. Nonpersistent messages are normally stored in the memory of the queue server. However, if the NonPersistentMessageClass attribute of a local queue is set to MQNPM_CLASS_HIGH, nonpersistent messages are also stored on disk while the queue manager is not running.

The storage for a local queue comprises the following files:

The queue file
This file is an audited, key-sequenced Enscribe file. The queue file contains one record of message data for each persistent message stored in the queue.

The queue overflow file
This file is an audited, key-sequenced Enscribe file. The queue overflow file contains zero or more records of message data for each persistent message stored in the queue.

Zero or more message overflow files
These files are unstructured Enscribe files. Message overflow files contain the message data for very large persistent messages. The message data is stored in a format that is efficient for transferring data in bulk. Message overflow files are unaudited by default, but you can configure them to be audited if you require operations on the messages to be replicated to a backup system using RDF.

Zero or one nonpersistent data storage file
This file is an unaudited, unstructured Enscribe file. A nonpersistent data storage file contains the nonpersistent messages in the queue. The file exists only while the queue manager is not running, and only if the NonPersistentMessageClass attribute of the queue is set to MQNPM_CLASS_HIGH.

Zero or one queue metadata file
This file is an unaudited, unstructured Enscribe file. The queue metadata file holds information about the relative positions of persistent and nonpersistent messages in a queue whose NonPersistentMessageClass attribute is set to MQNPM_CLASS_HIGH,
Queue servers

When a local queue is first opened, by either an application or the queue manager, the queue server reads the queue file and, if present, the queue metadata file. The queue server uses the information in these files to build in memory a representation of the messages in the queue. The queue server also reads the nonpersistent data storage file, if present, to load the nonpersistent messages into memory. The queue server then deletes the nonpersistent data storage file and the queue metadata file. Using the representation of the messages in the queue that is stored in its own memory, the queue server can access persistent messages without having to search the indexes of files on disk. The queue server also uses this representation to maintain information about the relative positions of persistent and nonpersistent messages in the queue.

A queue server controls the availability of messages in the queues that it manages. Whether a message is available is determined primarily by whether it is currently involved in a unit of work. The queue server uses TMF to monitor all units of work that involve messages in its queues. If a persistent message is currently part of a unit of work, the queue server does not attempt to access the records on disk containing that message until TMF notifies the queue server that the unit of work has completed. When TMF notifies the queue server that a unit of work has been committed, the queue server can then make available any messages put on its queues during that unit of work. Similarly, when TMF notifies the queue server that a unit of work has been backed out, the queue server can then make available any messages got from its queues during that unit of work. When messages become available, any MQGET calls waiting for the messages can complete. If more than one MQGET call is waiting for the same message, the queue server decides which call receives the message. The queue server also ensures that expired messages are not returned to applications.

A queue server provides fault-tolerant access to its queues and the messages stored in the queues. The representation of queues held in memory, the status information about WebSphere MQ objects, and access to those objects is checkpointed to the backup queue server process. Nonpersistent messages are also checkpointed to the backup queue server process and are stored in the memory of the backup process as well as in the memory of the primary process. If the CPU in which the primary queue server process is running fails, or if the primary queue server process itself fails, the backup process takes over as the primary process and creates a new backup process. The new process-pair then continues to function on behalf of connected applications without interruption. If you need to reduce resource consumption and improve throughput, and provided your applications do not require the additional reliability of fault-tolerance for nonpersistent messages, you can configure a queue to disable the checkpointing of nonpersistent messages. In this case, failure of the primary queue server process causes any nonpersistent messages on that queue to be lost.

The channel server

The channel server manages the channel status information and coordinates the starting and stopping of channels. Channel status information is maintained in memory for quick access and is checkpointed to the backup channel server process to provide fault-tolerance.

The channel server also maintains synchronization data for its end of a message channel. The channel server uses the synchronization data to ensure that, when a batch of messages is sent across the channel, the removal of the messages from the transmission queue at the sending end is not committed until all the messages have been committed on their respective destination queues at the receiving end.
At the sending end of a message channel, the removal of a batch of messages from the transmission queue and updates to the synchronization data are committed or backed out as a unit of work coordinated by TMF. Similarly, at the receiving end of a message channel, putting the messages on their respective destination queues and updates to the synchronization data are committed or backed out as a unit of work coordinated by TMF.

The remaining server processes that are configured within Pathway

Channel initiators, the command server, and trigger monitors perform their standard WebSphere MQ function. These server processes are configured in Pathway to be restarted automatically in the event of a failure. They are not fault-tolerant themselves but, because the key server processes are fault-tolerant, they can recover quickly and reestablish their function. Provided their server classes are configured appropriately, Pathway ensures that, in the event of a failure of the CPU in which one or more of these server processes are running, they are restarted automatically in a different CPU.

The default TCP/IP listener is initially configured in Pathway to listen on the default port, 1414, and to start a responder MCA as a process, not as a thread. You can modify this initial configuration and add additional TCP/IP listeners to the Pathway configuration. TCP/IP listeners are also configured to restart automatically in the event of a failure.

The queue manager server performs some internal processing on behalf of the key server processes. This processing includes generating expiration report messages and coordinating the starting of cluster sender channels. The queue manager server is also configured to restart automatically in the event of a failure.

The Monitoring Panels server supports the monitoring and display function of the Monitoring Panels. PATHMON starts this server when the Monitoring Panels are started.

The deferred message server performs some internal processing to support the publish/subscribe function in WebSphere MQ classes for Java Message Service.

Control commands and the Pathway configuration of a queue manager

In addition to providing the standard function required to administer a queue manager, the control commands crtmqm, strmqm, endmqm, and dltmqm have the following relationships with Pathway:

- **crtmqm** creates a default Pathway configuration for the queue manager.
- **strmqm** starts the Pathway runtime environment of the queue manager.
- **endmqm** ends the Pathway runtime environment of the queue manager.
- **dltmqm** deletes the Pathway configuration of the queue manager.

If you need to customize the default Pathway configuration and its server classes, you can still use these control commands provided you follow the guidance in this book. For example, **strmqm** automatically starts any additional queue servers that you configure provided you name them according to the conventions described in this book. Do not add application server classes to the Pathway configuration of the queue manager because it might disrupt starting and shutting down the queue manager when using the control commands.
The product code and files

All the server processes of the queue manager are implemented using the Position Independent Code (PIC) model. This means that a queue manager can support applications, exits, and installable service components that are compiled and linked as dynamic load libraries (DLLs). For compatibility with earlier versions of the product, libraries that are linked with applications are also provided in the Shared Resource Library (SRL) and nonnative (or TNS) code models.

The MQI library provides full support for threaded OSS applications that are written in C, C++, or Java and use the HP T1248 pthreads package. Operations like waiting for a message on an MQGET call can be performed by a multithreaded application without blocking the entire process.

Most of the product files that are not required for technical reasons to be in the NonStop OS file system are stored in the OSS file system. The OSS directory and file structure is similar to that used by WebSphere MQ on other UNIX platforms. Support for multiple installations of WebSphere MQ on a system is implemented by using environment variables to locate the var and opt directories for an installation. For each installation, the files in the NonStop OS file system are stored in three installation-wide subvolumes, and one subvolume for each queue manager that runs in the installation. The three installation-wide subvolumes contain the following files:

- The WebSphere MQ libraries and executable files for the NonStop OS environment
- The NonStop OS sample applications, the files to support EMS events generated by WebSphere MQ, and the file for setting the environment variables that are required by WebSphere MQ applications running in NonStop OS
- C and C++ header files, COBOL copy files, and TAL definition files for NonStop OS applications
Chapter 8. Managing scalability, performance, availability, and data integrity

This chapter discusses techniques for maximizing the performance and scalability of WebSphere MQ applications. It also describes the levels of data integrity and availability that you can expect from WebSphere MQ for HP NonStop Server and the configuration choices that can influence these levels.

The chapter contains the following sections:
- “Introduction to scalability and performance”
- “Message persistence” on page 86
- “Queue servers and queue files” on page 88
- “CPU assignment” on page 95
- “Fastpath binding application programs” on page 96
- “Data integrity” on page 98
- “Availability” on page 99
- “Persistent and nonpersistent data” on page 100
- “Database consistency” on page 102
- “Critical processes” on page 105
- “Queue manager clusters” on page 110
- “Configuration considerations for availability” on page 111
- “Configuration considerations for data integrity” on page 111

You need to have read Chapter 7, “WebSphere MQ for HP NonStop Server architecture,” on page 77 to understand and use the information in this chapter.

Introduction to scalability and performance

Tuning for performance and scalability is done to minimize the use of three key resources: CPU, memory, and the disk subsystem. Applications that use less CPU, less memory, and less disk I/O will perform better and scale better. For example, they can be configured to process ever growing workloads by using the using hardware and system software to its fullest.

The first part of this chapter addresses techniques to improve the performance of both applications and WebSphere MQ. The following sections summarize the broad principles for improving application performance.

Designing new applications for performance and scalability

Early in the design phase for new applications, you should consider how WebSphere MQ and other subsystems are used. The business need should determine which WebSphere MQ features are needed or are relevant for each application. Some WebSphere MQ features, such as persistent messages, carry strong integrity and delivery assurances, which require larger amounts of CPU, memory, and disk I/O to provide. If these assurances are not required for a particular application, then configuring WebSphere MQ accordingly can yield significant performance gains. This judgment is best made early in the design phase when the driving business need is being examined.
Introduction to scalability and performance

Designing to minimize or eliminate the use of shared resources
Absorbing growth in message traffic demands that the underlying hardware and system software be used to its fullest. Usually, any resource that is shared becomes a bottleneck, as the load increases. This bottleneck develops either because the degree of sharing has increased (for example, more users sharing the same CPU) or there is simply more of the resource is being consumed (for example, each user is performing more work).

From a performance and scalability perspective, the CPU and the Disk subsystem represent the most often-shared resources, and therefore require most attention.

WebSphere MQ processes can be spread across as many CPUs as you want. On a system with several CPUs, distributing the WebSphere MQ processes across the available CPUs provides better performance than using the default CPU assignment. Similarly, to maximize utilization of the disk subsystem, it is wise to position separate queue files on separate disks volumes, serviced by separate NonStop OS disk processes, if possible.

Memory is a critical shared resource within a CPU, especially when there are many processes running. Each process requires a certain amount of physical memory and, when the available physical memory reaches a critical threshold, the operating system might use paging to relieve the pressure, which can consume large amounts of CPU and disk I/O. Alternatively, the operating system might prevent any more processes from running. In either case, the result might be an undesirable level of performance. You can reduce the number of processes that are needed to service the application workload by using local queue manager agents that run as threads. This in turn reduces the usage of memory within the CPU, leading to an overall improvement in performance.

Performance tuning is inherently iterative
Achieving objectively better performance requires a measure-tune-remeasure cycle. Each tuning cycle should involve the change of only one major variable so that the effect of that variable can be compared against the effect of other variables. It is usually counterproductive to alter more than one setting at the same time, since some changes may improve overall performance more than others, while still others may reduce it. The performance of an application system is usually determined by the single, limiting, bottleneck. Making a tuning change to a system usually causes some other resource to become the bottleneck. The interplay between changes of this nature emphasizes the need to carefully follow the measure-tune-remeasure method when tuning for performance.

For example, a given application may write large numbers of small persistent messages to a queue. It may be useful to test the effects of making all those messages nonpersistent, or writing the same amount of data inside a smaller number of large messages. Both changes improve performance (usually) but without a separate measure-tune-remeasure cycle for each change, it may not be clear which brings the greatest improvement.

Message persistence
Message persistence is a property of a message, not of the queue in which the message is stored. A queue can store both persistent and nonpersistent messages. When an application puts a message on a queue, the application can specify whether the message is persistent or nonpersistent. Alternatively, the application
can specify that the persistence of the message is determined by the value of the DefPersistence attribute of the queue. For more information about the DefPersistence attribute of a queue, see the WebSphere MQ Application Programming Reference.

**Persistent messages**

Persistent messages carry the strongest assurances offered by WebSphere MQ regarding delivery and recoverability. Persistent messages are always stored on hardened media, and therefore survive a queue manager restart. TMF audits queue files so that reading or writing persistent messages results in disk activity to both the queue file itself, and the TMF audit files. TMF audit logging is required to maintain transactional integrity for persistent messages, even in the case of a system or hardware failure. The TMF audit logging associated with persistent messages must be considered when assessing the performance of a WebSphere MQ application design.

**Nonpersistent messages**

Unlike persistent messages, nonpersistent messages are not hardened to disk while the queue manager is running. Instead, they are stored in memory managed by the queue servers. If the NonPersistentMessageClass attribute of a local queue is set to MQNPM_CLASS_HIGH, nonpersistent messages in the queue are saved to disk when the queue manager shuts down and are recovered when the queue manager restarts. If the NonPersistentMessageClass attribute is not set in this way, nonpersistent messages in the queue do not survive a restart of the queue manager.

The primary reason for using nonpersistent messages is to improve performance. Persistent messages carry strong assurances for delivery and recoverability, so reading or writing them require disk activity to the queue files and the TMF audit files. This disk activity reduces the performance of applications that read or write persistent messages and WebSphere MQ channels that move persistent messages to other queue managers.

Depending on a queue server option for the queue, nonpersistent messages in a queue can be checkpointed to the backup queue server. The checkpointing of nonpersistent messages provides a high level of fault-tolerance, though not recoverability in the event of multiple failures or a complete site disaster.

**Nonpersistent messages and channels**

Message channels use synchronization logging at both the sending and receiving end to assure once and once-only delivery of messages sent over the network. This synchronization logging is additional to any audit logging performed by TMF (on behalf of the queue manager) when channels read and write messages to queues. Message channels can be configured to not perform synchronization logging when sending and receiving nonpersistent messages, by setting the NPMMSPEED attribute to FAST. The NPMMSPEED channel attribute controls the behavior of both sending and receiving channels that are processing nonpersistent messages. When NPMMSPEED is set to NORMAL for a channel, nonpersistent messages are part of the channel's message batch (as defined by the BATCHINT and BATCHSZ attributes) and require synchronization logging in the same way as persistent messages. Further, when NPMMSPEED is set to NORMAL, the channel reads and writes nonpersistent messages under syncpoint control, which causes a small amount of TMF audit file activity at the beginning and end of a transaction.
Message persistence

When NPMSPEED is set to FAST, nonpersistent messages are not part of the channel’s current batch and are read and written to queues outside of syncpoint control. Using NPMSPEED(FAST) therefore removes two sources of channel disk activity; the logging done by the channel batch synchronization mechanism, and the TMF audit logging that would otherwise be done for reading and writing messages under syncpoint.

NPMSPEED(FAST) is a performance option that trades recoverability of nonpersistent messages after a failure for considerably higher performance; using NPMSPEED(FAST) can cause nonpersistent messages to be lost if the channel or network fails.

The default value for NPMSPEED is FAST.

Queue servers and queue files

Queue servers are WebSphere MQ processes that mediate the reading and writing of messages and the storage of those messages. As such, queue servers represent a key WebSphere MQ component worthy of close attention. Queue server configuration can have a major impact on performance of a busy WebSphere MQ system. Within a queue manager, you can configure as many queue servers as you need to handle the messaging load.

Queue servers have responsibility for the physical storage of messages held in queue. It is useful to examine the storage of persistent and nonpersistent messages separately, because queue servers manage them in fundamentally different ways.

A queue server can manage one or more queues. When created, a queue is managed by the default queue server. Therefore, by default, all queues are managed by the default queue server unless they are assigned to other queue servers.

For each queue managed by a queue server, the following files exist:
• A queue file
• A queue overflow file
• A touch file

Additionally, there might be a message overflow file for each large persistent message in the queue, as defined by the message overflow threshold of the queue. Message overflow files are discussed in "Message overflow files" on page 92.

If the NonPersistentMessageClass attribute of a local queue is set to MQNPM_CLASS_HIGH, the queue might have up to two additional files. These files are used to store the nonpersistent messages in the queue, and the relative positions of the persistent and nonpersistent messages in the queue, while the queue manager is not running. When the queue manager restarts and the queue is opened, the contents of these files, including the nonpersistent messages, are reloaded into memory and the files are then deleted.

How persistent messages are stored

Persistent messages are always hardened to disk. The way that persistent messages are stored depends mostly on the size of the message:
<table>
<thead>
<tr>
<th>Message Size</th>
<th>How stored</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt; 3000 bytes (approximately)</td>
<td>A message is stored as a single record in the queue file.</td>
</tr>
<tr>
<td>&gt; 3000 bytes (approximately) and</td>
<td>The first 3000 bytes of a message (approximately) are stored as a single record in the queue file. The remainder of the message is stored as one or more records in the queue overflow file.</td>
</tr>
<tr>
<td>&lt;= message overflow threshold</td>
<td>The first 3000 bytes of a message (approximately) are stored as a single record in the queue file. The remainder of the message is stored in a message overflow file.</td>
</tr>
</tbody>
</table>

**How nonpersistent messages are stored**

Nonpersistent messages in a queue are stored in the memory buffers of the queue server while the queue manager is running. For this reason, the queue server performs no disk I/O when reading and writing nonpersistent messages.

Note, however, that the size of a queue server’s memory is limited to approximately 1 GB. Within this memory, at any point in time, the queue server must be able to store all the nonpersistent messages in all the queues that the queue server is managing. As part of your configuration planning, therefore, you need to consider whether one queue server is sufficient to manage all your queues. If some of your queues might contain very large nonpersistent messages, or a very large number of nonpersistent messages, you might need to configure additional queue servers to manage these queues.

**Queue server CPU distribution**

Queue servers are Pathway server classes, and are therefore defined in the queue manager’s Pathway. When the queue manager is created, a default queue server is defined in the Pathway with the queue server’s primary process running in CPU 0. Since CPU 0 normally contains many high priority system processes, it is usually better not to run a queue server in CPU 0. Instead, identify a number of relatively quiet CPUs and create queue servers in these CPUs.

Another factor to consider when deciding where to run a queue server is the location of the applications that use the queues managed by the queue server. Application processes transfer data directly to and from the queue server. If an application requires a high throughput of messages, running the primary queue server process in the same CPU as the application can improve throughput under certain conditions. This is because data transfer between two processes running in the same CPU is faster than data transfer between two processes running in different CPUs. You might expect to achieve the greatest improvement for an application that performs queuing operations using primarily nonpersistent messages.

**Reassigning a WebSphere MQ object to another queue server**

A queue server manages the internal status of all the WebSphere MQ objects for which it is responsible. A queue server also performs all the messaging operations on its local queues.

When you create a WebSphere MQ object other than a channel, the default queue server is initially responsible for the object. However, you can use the --server
Queue servers and queue files

parameter on the `altmqfls` command to allocate the object to a different queue server. For example, the following command reassigns the local queue TEST.QUEUE to the queue server with the symbolic name app.queue.server:

```
altmqfls --qmqr QMGR --type QLOCAL --server app.queue.server TEST.QUEUE
```

You must ensure that the new queue server is configured as a server class within Pathway, and is started, before applications can use the object again. When you use the `altmqfls` command to change responsibility for an object, no check is made at that time to determine whether the new queue server is configured or active.

Note that, in the previous release, a queue server was identified by its process name. This meant that the WebSphere MQ objects managed by the queue server were directly associated with the process name. In this release, a queue server is identified by a symbolic name instead. This means that the WebSphere MQ objects managed by the queue server are directly associated with the symbolic name, not the process name. This arrangement provides greater flexibility. For example, you can now change the process name of a queue server without then having to assign back to the queue server the WebSphere MQ objects for which it was originally responsible.

For more information about the `altmqfls` command and its syntax, see “`altmqfls (alter WebSphere MQ object attributes)`” on page 244.

You can use the `dspmqfls` command to determine the queue server that is currently responsible for a WebSphere MQ object. For more information about the `dspmqfls` command and its syntax, see “`dspmqfls (display WebSphere MQ object attributes)`” on page 264.

In a busy WebSphere MQ system it is neither efficient nor scalable for a single queue server to manage all the local queues. The main reason for creating new queue servers and assigning local queues to them is to spread the processing load of the queue servers more evenly across the available CPUs.

Cluster transmission queue:
SYSTEM.CLUSTER.TRANSMIT.QUEUE

WebSphere MQ uses a single cluster transmission queue for all clustering operations. If your queue manager is part of a busy cluster, this queue should be assigned to a dedicated queue server, both to maximize performance of clustering operations and to minimize the impact on other applications.

Moving the files associated with a local queue

By default, the queue file, queue overflow file, and touch file of a local queue are stored in the queue manager’s subvolume. There are two reasons why you might consider moving these files to another disk volume:

- To spread the disk I/O load more evenly across disk volumes
- To avoid reaching the limit on the number of files on a disk volume that can be open.

You can use the `--volume` parameter on the `altmqfls` command to move these files to another volume. For example, the following command, entered at an OSS shell command prompt, moves the files associated with the local queue TEST.QUEUE to the volume $DATA01:

```
altmqfls --qmqr QMGR --type QLOCAL --volume $DATA01 TEST.QUEUE
```
The following command, entered at a TACL command prompt, performs the same function:

```
altmqfls --qmgr QMGR --type QLOCAL --volume $DATA01 TEST.QUEUE
```

The queue manager must be running when you move the files, but no application must have the queue open. The files are stored in a subvolume with the same name as that of the queue manager’s subvolume.

By default, message overflow files are also stored in the queue manager’s subvolume. You cannot move existing message overflow files, but you can cause new message overflow files for a local queue to be created in a different subvolume by using the --msgofsubvol parameter on the `altmqfls` command. For example, the following command, entered at an OSS shell command prompt, causes new message overflow files for the local queue TEST.QUEUE to be created in the subvolume $DATA01.TESTMOF:

```
altmqfls --qmgr QMGR --type QLOCAL --msgofsubvol $DATA01.TESTMOF TEST.QUEUE
```

The following command, entered at a TACL command prompt, performs the same function:

```
altmqfls --qmgr QMGR --type QLOCAL --msgofsubvol $DATA01.TESTMOF TEST.QUEUE
```

For more information about the `altmqfls` command and its syntax, see "altmqfls (alter WebSphere MQ object attributes)" on page 244.

### Increasing the capacity of a local queue

By default, a local queue can store up to 100 MB of data belonging to persistent messages. If an application attempts to put a persistent message on a queue, and there is not enough space in the queue for the message, the MQPUT or MQPUT1 call fails with reason code MQRC_Q_SPACE_NOT_AVAILABLE.

To increase the capacity of a local queue, first use the `dspmqfls` command to identify the queue file and queue overflow file for the local queue. Then use NonStop OS facilities to determine whether you need to increase the size of the queue file, the queue overflow file, or both. For each file whose size you need to increase, determine the following properties:

- The size of the primary extent
- The size of a secondary extent
- The maximum number of extents

Finally, use the `--qsize` parameter on the `altmqfls` command to increase any or all of these values.

For example, consider the following `altmqfls` command, as entered at an OSS shell command prompt:

```
altmqfls --qmgr QMGR --type QLOCAL --qsize \(100,200,500,200,200,100\) TEST.QUEUE
```

This command sets the following properties of the queue file and queue overflow file for the local queue called TEST.QUEUE:

- The size of the primary extent of the queue file is 100 pages.
- The size of the secondary extent of the queue file is 200 pages.
- The maximum number of extents for the queue file is 500.
- The size of the primary extent of the queue overflow file is 200 pages.
- The size of the secondary extent of the queue overflow file is 200 pages.
Queue servers and queue files

- The maximum number of extents for the queue overflow file is 100.

The following command, entered at a TACL command prompt, performs the same function:

```
altmqfls --qmgr QMGR --type QLOCAL --qsize (100,200,500,200,200,100) TEST.QUEUE
```

If you need to increase the size of the queue file or queue overflow file beyond the maximum allowed by NonStop OS, you must partition the file. For more information about partitioning a queue file or queue overflow file, see "Partitioning a queue file or queue overflow file." If the queue file or queue overflow file are already partitioned, the `altmqfls` command applies any changes in their properties to all the partitions.

For more information about the `altmqfls` command and its syntax, see "`altmqfls` (alter WebSphere MQ object attributes)" on page 244.

### Partitioning a queue file or queue overflow file

File partitioning is a technique that splits a file across more than one disk volume. The file then consists of a primary partition and one or more secondary partitions. Each partition resides on a different volume but has the same file name and subvolume name as each of the other partitions.

Partitioning a queue file or queue overflow file has the following advantages:

- Partitioning spreads the disk I/O load for a single queue across more than one disk volume.
- Partitioning allows the size of a queue file or queue overflow file to be larger than the maximum allowed on a single disk volume.

You can partition an existing queue file or queue overflow file by using the FUP utility.

After you have partitioned a queue file or queue overflow file, the queue server tries to spread new messages evenly across all available partitions.

**Note:** You cannot use the `altmqfls` command to move a partitioned queue file or queue overflow file.

### Message overflow files

A message overflow file is created for each persistent message that is larger than the defined message overflow threshold for the queue. For a very large message, it is more efficient to store most of the message in a message overflow file. For a smaller message, it is more efficient to store the entire message in the queue file and queue overflow file. The crossover point has been empirically determined to be about 200 KB.

The message overflow threshold is set to its default of 200 KB when a queue is created. You can change the threshold by using the `--oflowsize` parameter on the `altmqfls` command, as in the following example:

```
altmqfls --qmgr QMGR --type QLOCAL --oflowsize 400000 TEST.QUEUE
```

For more information about the `altmqfls` command and its syntax, see "`altmqfls` (alter WebSphere MQ object attributes)" on page 244.
Message overflow files are unstructured files that are not normally audited by TMF. However, if you need message overflow files that are audited, see “Audited message overflow files” on page 94.

**Browsing persistent messages**

A queue server can maintain in memory the first $n$ bytes of the application data of all persistent messages in a local queue. This feature can have a dramatic effect on the performance of applications that are browsing persistent messages in a queue. If the application data in each message that the application browses is smaller than this browse threshold, no disk I/O is needed to browse the messages.

You can use the --browse parameter on the `altmqfls` command to set the browse threshold for a local queue. For example, the following command sets the browse threshold for the local queue TEST.QUEUE to 100 bytes:

```
altmqfls --qmgr QMGR --type QLOCAL --browse 100 TEST.QUEUE
```

By default, the browse threshold is 0 bytes, which means that no application data is kept in memory. The maximum browse threshold is 25000 bytes.

You can also exploit this feature if the first 100 bytes, say, of the application data in each persistent message is sufficient for an application to determine whether it is interested in the message. When browsing these messages in a local queue with a browse threshold of 100 bytes, the application can supply a buffer with a length of 100 bytes and accept truncated messages. In this way, no disk I/O is needed to browse the messages. When the application finds a message of interest, it can remove the message from the queue.

For more information about the `altmqfls` command and its syntax, see “`altmqfls` (alter WebSphere MQ object attributes)” on page 244.

**The Measure counter**

If you use the --meascount parameter on the `altmqfls` command, the queue server maintains a user defined Measure counter whose value is the current depth of the queue. For example, the following command causes the queue server responsible for the local queue TEST.QUEUE to maintain a Measure counter called TESTCOUNT1:

```
altmqfls --qmgr QMGR --type QLOCAL --meascount TESTCOUNT1 TEST.QUEUE
```

The counter is useful when you need to gather data to assess the overall performance of a system. Using the counter, you can correlate put and get activity on a queue with other system activity such as CPU utilization and disk I/O.

The queue server can maintain the Measure counter only if it is included in an active measurement. If it is not included in an active measurement, and messages are put in the queue and removed from the queue, the value of the counter will no longer represent the current depth of the queue. If the counter is subsequently included in an active measurement, you can cause the queue server to reset the Measure counter to the current depth of the queue by using the --resetmeascount parameter on the `altmqfls` command, as in the following example:

```
altmqfls --qmgr QMGR --type QLOCAL --resetmeascount TEST.QUEUE
```

For more information about the `altmqfls` command and its syntax, see “`altmqfls` (alter WebSphere MQ object attributes)” on page 244.
Queue servers and queue file

The queue server options

You can use the --qsoptions parameter on the altmqfls command to set certain queue server options. For example, one option causes a representation of the messages in a queue to be loaded from disk into cache when the queue manager starts, instead of when the queue is first opened. Another option causes nonpersistent messages in a queue to be checkpointed to the backup queue server. Each of these options (C, L, R, and S) can be used alone, or they can be combined, depending on your requirements for the queue.

Every time you use the altmqfls command with the --qsoptions parameter, each of the options C, L, R, and S is either set or left not set. For example, using the parameter --qsoptions S sets the S option, but leaves the C, L, and R options not set. The --qsoptions parameter can be specified once and once only in a command.

For more information about the altmqfls command and its syntax, see “altmqfls (alter WebSphere MQ object attributes)” on page 244.

Checkpoint nonpersistent messages

You set this option by using the --qsoptions C parameter on the altmqfls command. The option causes the queue server to checkpoint nonpersistent messages to its backup process. This is the default behavior when a queue has just been created. See the following command, for example:

altmqfls --qmgr QMGR --type QLOCAL --qsoptions C TEST.QUEUE

Using this option results in a higher degree of reliability for nonpersistent messages, at the cost of greater IPC traffic and greater CPU and memory utilization for both the primary and backup queue server processes.

Lock in memory

You set this option by using the --qsoptions L parameter on the altmqfls command. The option causes the queue server to lock in memory the data structures and chains associated with a queue. The queue’s memory data structures are not unloaded to disk to make room for other queues. The default behavior, when a queue has just been created, is to unload a queue’s data structures to disk when required. See the following command, for example:

altmqfls --qmgr QMGR --type QLOCAL --qsoptions L TEST.QUEUE

Using this option results in faster access to a queue’s memory data structures at the possible expense of other queues.

Audited message overflow files

Message overflow files are unstructured files that are not normally audited by TMF. However, if you use the --qsoptions R parameter on the altmqfls command, message overflow files for the specified queue are created as audited files so that operations on their contents can be replicated to a backup system using RDF. See the following command, for example:

altmqfls --qmgr QMGR --type QLOCAL --qsoptions R TEST.QUEUE

Audited message overflow files provide a higher degree of recoverability and better support for disaster recovery at the expense of higher levels of TMF audit logging.

Load at startup

You set this option by using the --qsoptions S parameter on the altmqfls command. The option causes the queue server to read the queue files, and build its
internal message data structures, at queue server startup (typically, at queue manager startup) rather than when the queue is first opened. See the following command, for example:

```
altmqfls --qmgr QMGR --type QLOCAL --qsoptions S TEST.QUEUE
```

Using this option results in less CPU and disk I/O activity when a queue is first opened, but causes more queue server activity (CPU and disk I/O) during queue manager start.

**Putting a local queue into maintenance mode**

You can use the `--maintain` parameter on the `altmqfls` command to put a local queue into maintenance mode. In this mode, no application can open the queue. You can therefore perform certain maintenance operations on the queue, such as changing the queue server responsible for the queue or partitioning the queue file and queue overflow file associated with the queue.

When you put a queue into maintenance mode, all nonpersistent messages in the queue are discarded. You cannot put a queue into maintenance mode while an application still has the queue open.

For example, the following command puts the local queue TEST.QUEUE into maintenance mode:

```
altmqfls --qmgr QMGR --type QLOCAL --maintain ON TEST.QUEUE
```

The following command takes the same queue out of maintenance mode so that applications can open the queue again:

```
altmqfls --qmgr QMGR --type QLOCAL --maintain OFF TEST.QUEUE
```

For more information about the `altmqfls` command and its syntax, see "altmqfls (alter WebSphere MQ object attributes)" on page 244.

**CPU assignment**

When you start a queue manager, WebSphere MQ for HP NonStop Server creates a number of processes. Some of these processes provide the core messaging operations while others perform functions that indirectly support these operations. Processes that provide the core messaging features of WebSphere MQ are busy when applications are busy making MQI messaging calls (MQGETs and MQPUTs).

The following WebSphere MQ and NonStop OS processes are involved in core messaging operations:
- Local queue manager agents (LQMAs)
- Queue servers
- NonStop OS disk processes

The following WebSphere MQ and NonStop OS processes are involved in distributed queuing operations:
- MCAs
- Queue servers
- Channel server
- NonStop OS disk processes
- NonStop OS TCPIP or SNA processes

The following WebSphere MQ processes are involved in support or administrative operations:
**CPU assignment**

- Queue manager server
- Repository managers
- Execution controller
- Channel initiators
- Trigger monitors
- TCP/IP listeners

A heavily loaded WebSphere MQ system usually shows high CPU utilizations for the processes in the first two categories shown above (core messaging processes and distributed queuing processes).

Distributing the CPU load of WebSphere MQ therefore usually involves spreading the queue manager’s LQMA, queue servers and MCAs across as many CPUs as possible.

The default configuration for a new queue manager is to run processes in CPU 0 (for NonStop process pairs, the primary processes run in CPU 0 and the backup in CPU 1). This is not adequate for a production environment, and you should be prepared to re-configure the queue manager’s Pathway to sensibly spread busy processes across as many CPUs as possible. CPU 0 usually contains many high-priority operating system processes and is therefore a poor choice for running busy WebSphere MQ processes.

You can use the rules in the process management rules configuration file, qmproc.ini, to specify the CPUs in which local queue manager agents and MCAs must run. The rules allow you to establish a predictable configuration in order to achieve a stable, tuned environment.

For example, you can specify a rule that assigns the MCA for an particularly busy channel to the CPU that is running the queue server for the queues that the channel uses. For a very active application that requires rapid response times, you can use a rule to ensure that the application is serviced by a local queue manager agent that runs as a process in a lightly loaded CPU. You can also use rules to give local queue manager agent and MCA processes meaningful names, which makes it easier to identify the processes when measuring and tuning your system.

Looking beyond WebSphere MQ processes, the NonStop OS disk processes are an important component of messaging operations (particularly when queue servers are handling persistent messages). Heavy use of distributed queuing over a network necessarily causes the corresponding TCP/IP or SNA processes to consume CPU. You should consider the number and CPU placement of these processes when assessing the overall performance profile on a WebSphere MQ installation.

---

**Fastpath binding application programs**

Fastpath binding is a feature of the MQI that is designed to make WebSphere MQ applications run more efficiently. Fastpath binding can be used to reduce the overhead inherent in all MQI verbs issued by WebSphere MQ application programs. Applications that use fastpath binding are referred to as trusted applications because of the proximity of the queue manager software and memory to the customer’s application software. Errors in trusted applications can damage WebSphere MQ data structures and can compromise queue manager integrity.
**Fastpath binding application programs**

**Background**

When an application program executes an MQCONN verb, WebSphere MQ creates (or reuses) a special process called a Local Queue Manager Agent (LQMA). The LQMA services all subsequent MQI calls made by the application using that connection handle. On WebSphere MQ for HP NonStop Server, the LQMA may be running in the same or a different CPU as the connecting application.

Since the LQMA is a separate process, an application program does not have direct access to the memory or files used by WebSphere MQ. An errant application program cannot therefore damage the LQMA. In this way, WebSphere MQ software and data structures that are critical to its operation are isolated from the customer's application software and data. This isolation comes at a price. The MQCONN verb cannot complete until the new LQMA process is created (or an existing one is reused), but the greater cost results from the fact that information must be passed to the LQMA each time the application issues an MQI verb.

This MQI information is passed to the LQMA using an Interprocess Communications (IPC) mechanism. IPC requests may be intra-CPU (if the LQMA happens to be running in the same CPU as the connecting application) or the more expensive inter-CPU.

**Reducing MQI overhead**

Application designers can use fastpath binding as a way of removing the application-LQMA IPC overhead associated with each MQI verb (the possible LQMA process creation is also avoided). When fastpath binding is enabled for an application, no separate LQMA process is used. Instead, the components of WebSphere MQ normally contained in the LQMA, are loaded into the user’s process (for example, the connecting application’s process).

Subsequent MQI calls issued by the application require no IPC activity with the LQMA, since the WebSphere MQ software and data structures (normally stored in the LQMA process) are held locally within the application's process. Note that other IPC activity may still occur when the queue manager needs to communicate with other WebSphere MQ processes such as queue servers. Fastpath binding does not remove all IPC activity, but it does remove an important source of IPC activity.

**Enabling fastpath binding**

To use fastpath binding, an application connects to the queue manager by calling MQCONNX with the MQCNO_FASTPATH_BINDING option. However, fastpath binding is used only if the environment variable MQCONNECTTYPE has a value that is not STANDARD, or has an undefined value. If the value of MQCONNECTTYPE is STANDARD, standard binding is used even though the application has requested fastpath binding.

After an application has connected using fastpath binding, it can issue MQI calls as if it had connected using standard binding. The behavior of MQI calls for fastpath binding is the same as for standard binding. However, for certain restrictions when using fastpath binding, see "Restrictions when using fastpath binding."

**Restrictions when using fastpath binding**

When using standard binding connections, application software is isolated from internal WebSphere MQ data. This isolation is removed when a trusted application establishes a fastpath binding connection. Errors in trusted applications can therefore damage WebSphere MQ data structures and can compromise queue
Fastpath binding application programs

manager integrity. This must be taken in consideration when assessing whether to use fastpath bindings for a given application.

The following additional considerations apply to trusted applications:

• Trusted applications must explicitly disconnect from the queue manager by calling MQDISC.
• You must stop trusted applications before stopping the queue manager by issuing the **endmqm** command.
• Trusted applications must run under the user ID whose corresponding principal is mqm.

Data integrity

The concept of data integrity can be understood best by considering the following desirable aspects of the storage and management of data, particularly for online transaction processing applications:

• When a record of data is written or read from a record in a database, the data must not be corrupted, duplicated or lost without an error indication during the transfer.
• When data is required to be accessed concurrently by multiple processes, these processes must be presented with the same view of the data and the data must be protected from corruption, duplication or loss.
• When a set of consistent changes are required to data in multiple databases, the changes must either be all made or none made.

For WebSphere MQ, the data integrity requirements for data storage listed above are just as applicable to messaging operations (for example, MQPUT and MQGET) on queues. Note that consistency of multiple database changes must be preserved across and between application databases and WebSphere MQ queues.

WebSphere MQ for HP NonStop Server is designed to maintain data integrity for persistent data operations through any single point of failure (hardware or software). In fact, data integrity can be maintained in several cases through multiple points of failure. This does not imply that nonpersistent messages are unreliable; queue server architecture provides features for making nonpersistent messages as reliable as persistent ones, except in the case of a catastrophic system failure. (For more on queue server architecture, see Chapter 7, “WebSphere MQ for HP NonStop Server architecture,” on page 77.)

With WebSphere MQ for HP NonStop Server, data integrity is provided by a combination of fundamental features of NonStop OS, the NonStop Server hardware, and WebSphere MQ.

There are several ways in which the level of data integrity can be influenced by choices in the configuration of WebSphere MQ:

• Choice of message persistence by the application
• Choice of storage technique for persistent messages
• Choice of nonpersistent message tuning options
• Choice of queue server configuration options
• Configuration of hardware supporting queue files
• Use and configuration of TMF
• Use and configuration of Remote Database Facility (NonStop RDF)
Availability

Availability in general terms is a measure of the time that an application, or service is operational and usable compared to elapsed time. Thus continuous availability expresses the ultimate aim of all such systems. Of course, such measurements mean nothing without a corresponding time period associated with the measurement, since it is easy to claim 100% availability over a short period of time.

In a real-world situation over a reasonable operational time span, a system will suffer a number of different types of challenge to its availability:

- Hardware and system software failures
- Failures within the application software itself
- The need to make changes to any aspect of the system for preventative maintenance
- Traffic or transaction load that exceeds design constraints or resource limitations

As for data integrity, with WebSphere MQ for HP NonStop Server, availability is provided by a combination of fundamental features of NonStop OS, the NonStop Server hardware, and WebSphere MQ.

It is important to recognize that (at least in its current form) WebSphere MQ for HP NonStop Server does not aim to provide a level of continuous availability equivalent to that provided by the NonStop OS system software such as the file system. There are in fact some components of the queue manager that do provide this level of availability, but the queue manager as a whole does not.

WebSphere MQ for HP NonStop Server is intended to provide a level of availability such that on any single point of failure (hardware or software):

- The queue manager connections that suffer interruption or discontinuation of service are limited to those with components that suffer the failure directly (for example, on a CPU failure, connections that fail should only be those that are provided by LQMA processes that were running in that CPU)
- The queue manager remains available for new connection attempts without manual intervention being required from system administrators
- Access to a queue manager object (for example a local queue) must not be prevented from any connection other than those directly affected by the failure.

In addition to these NonStop OS specific features of WebSphere MQ, there are several features that are common to all WebSphere MQ platforms that you can make use of to enhance the availability of WebSphere MQ for HP NonStop Server. There are several ways in which in which the level of availability can be influenced by choices in the configuration of WebSphere MQ:

- Choice of application design
- Choice of Pathway configuration options
- Use and configuration of standard WebSphere MQ functions (including clusters)
- Choice of hardware supporting WebSphere MQ
- Choice of nonpersistent message configuration options
- Use and configuration of TMF
- Use and configuration of NonStop RDF
Persistent and nonpersistent data

When used in relation to WebSphere MQ, the term *persistence* implies several qualities to data:

- A change to persistent data survives queue manager restart
- Persistent data is stored in non-volatile media
- Persistent data satisfies the highest requirement for data integrity provided by the particular operating environment
- Persistent data operations trade this higher level of integrity for speed and resource utilization

Most administrative operations are made to persistent data, since the configuration databases of WebSphere MQ must have the highest level of data integrity to minimize the risk that the availability of WebSphere MQ is seriously degraded. The speed or resource utilization of most administrative operations is not of prime concern, since they are performed infrequently.

Examples of persistent administrative operations:

- Change to a queue’s attributes
- Change to a channel’s attributes
- Creating a new queue, process, namelist or channel

Examples of nonpersistent administrative operations:

- Starting or stopping a channel
- Inquiring about the attributes of a queue manager object

WebSphere MQ for HP NonStop Server provides several choices for the way messages are stored, based on the choice between persistent and nonpersistent made by the application when enqueuing a message, and on queue level configuration choices made by the system administrator.

Persistent messages

Persistent messages are always stored on disk. As system administrator, you can choose between two storage techniques for persistent messages on a queue by queue basis. The choice of which type of storage technique to use is based on message size since the primary purpose for implementing different storage techniques is to improve the performance of messaging operations on very large messages.

All persistent messages have a single record in the audited queue file. This record contains the headers and important control information about the message, plus as much message data as can be accommodated within the maximum record size of 4096 bytes.

The fastest and most efficient mechanism for small and medium size persistent messages under TMF control is to store overflow message data in the queue overflow file in multiple records. All data in these messages is logged in TMF and is therefore fully recoverable from audit trails if necessary.

For large messages (over about 200 KB of data), the most efficient mechanism is to use dedicated message overflow files, which are normally unaudited. The data that is written to a message overflow file does not therefore get written to the TMF audit trail, saving CPU and disk I/O, and can also be transferred in large blocks.
However, you do have the option of using audited message overflow files for a queue so that operations on their contents can be replicated to a backup system using RDF.

Table 2 compares the two storage techniques from the point of view of data integrity.

<table>
<thead>
<tr>
<th>Aspect of data integrity</th>
<th>Using the queue overflow file</th>
<th>Using message overflow files</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amount of data that is audited by TMF</td>
<td>All message data is audited.</td>
<td>Only the data that will fit in the record in the queue file is audited.</td>
</tr>
<tr>
<td>Recoverability of data from audit trail in case of multiple failures resulting in total volume loss</td>
<td>Entire message is recoverable from audit trail.</td>
<td>Only the message header and first part of data is recoverable from audit trail.</td>
</tr>
<tr>
<td>Maximum size of message possible</td>
<td>Limited by number of record locks per volume per transaction imposed by Enscribe. For a non-partitioned file, this is approximately 20 MB for a default Enscribe configuration. The practical limit may be smaller than this due to physical memory limitations.</td>
<td>Limited only by available disk space or 100 MB which is the maximum permissible message size for WebSphere MQ.</td>
</tr>
<tr>
<td>Compatibility with RDF</td>
<td>Fully compatible.</td>
<td>Compatible only if the --qsoptions R parameter on the altmqfls command is specified for the queue.</td>
</tr>
<tr>
<td>Fault-tolerance to disk hardware problems</td>
<td>Fully fault-tolerant to any single point of failure if mirrored disks are employed and since all data is audited, file recovery can be performed in the event of failure of both disks in a mirrored pair, or the only disk if not mirrored.</td>
<td>Fully fault-tolerant to any single point of failure if mirrored disks are employed to hold message overflow files. Message data cannot be recovered in the event of total volume failure unless the --qsoptions R parameter on the altmqfls command is specified for the queue.</td>
</tr>
</tbody>
</table>

In summary, for persistent messages, both storage techniques attain a very high level of data integrity, but there are some limitations for message overflow files because these files are not normally audited by TMF. You can address the limitations in the following ways:

- Avoid the need for message overflow files by using segmentation to split very large messages into smaller physical messages.
- Avoid the need for message overflow files by changing the message overflow threshold for a queue so that it is greater than the size of the largest message that might be put on the queue.
- Use audited message overflow files.
Persistent and nonpersistent data

Nonpersistent messages

Nonpersistent messages are normally stored in memory. However, if the NonPersistentMessageClass attribute of a local queue is set to MQNPM_CLASS_HIGH, nonpersistent messages are also stored on disk while the queue manager is not running. Occasionally the queue server also copies nonpersistent messages to disk files for certain administrative operations while changes are made to configuration online. When applications access nonpersistent messages, the messages are always resident in memory.

The queue server manages the storage of all messages for the queues that it is responsible for and, for each queue, provides a configuration option to control the level of data integrity applied to nonpersistent messages. The configuration parameter controls whether the queue server checkpoints nonpersistent messages to the backup process, so that the nonpersistent messages are as tolerant as persistent messages to the failure of the CPU containing the primary queue server process.

The price paid for the use of nonpersistent message checkpointing is that the queue server consumes more CPU (primary and backup), transfers more data to the backup during checkpointing and consumes more memory in the backup process since it has to store the message again.

If nonpersistent message checkpointing is not enabled for a queue, and the primary queue server process terminates abnormally (due to process or CPU failure), then any nonpersistent messages that were present in the queue before the takeover are discarded since they were only stored in the primary process.

The option of fault-tolerant nonpersistent messages is not available on any other WebSphere MQ platform and, for consistency across the product line, nonpersistent message checkpointing is enabled by default (use atmqfls to disable this feature). You should consider carefully whether nonpersistent messages meet your needs for data integrity. If they do, then you should be able to take advantage of their significant resource utilization savings and performance gains.

Database consistency

Database consistency must be preserved both internally by WebSphere MQ, and externally when the syncpoint option is used for messaging operations. Both are vital for data integrity and availability.

Internal database consistency

All critical database files within the queue manager are audited by TMF. The queue manager processes must therefore use transactions to make changes to them. All changes are logged in the TMF audit trail and also (if used) duplicated using RDF to one or more disaster recovery systems. Thus the highest level of data integrity for internal databases can be assured.

The use of TMF to protect internal databases helps ensure that on system failures as well as software failures the integrity of the critical databases is not compromised. This means that restarting (automatically or manually) the processes or services that use these databases is much more likely to be successful, leading to higher availability.
External database consistency

The coordination of changes to and consistency of external databases with WebSphere MQ queues is enabled by the use of TMF within WebSphere MQ and by applications. WebSphere MQ messaging operations can be made under global syncpoint control, which requires the application to have an active current transaction (either inherited from another process or started using TMF BEGINTRANSACTION service). This transaction is inherited by the queue manager and any storage or critical database update that is required is performed using this transaction. TMF ensures that the appropriate audit trail entries are recorded for all disk I/O performed under the transaction in whatever process performs the update.

TMF also maintains a consistent view of the updates that have been made but not yet committed by holding record locks on the affected records in all database files. For example, a row in a SQL table that has been inserted under TMF control cannot be updated or deleted until the transaction commits.

When the queue manager replies to the application process (and returns from the MQI), the application may continue to do more work under this global unit of work by using WebSphere MQ to enqueue or dequeue more messages or performing database updates of its own using Enscribe or NonStop SQL. When the application is ready to make the changes to WebSphere MQ queues and, other databases permanent, the ENDTRANSACTION service of TMF is called, which commits the changes to all databases, system wide at the same time.

If the application determines that an error has occurred during the processing of the global unit of work, and some updates to databases have been performed, then the application should call ABORTTRANSACTION to cause TMF to back out the changes to all databases, system wide at the same time. This could cause, for example, a message to be replaced on a queue after it has been de-queued in a syncpoint MQGET operation as well as the removal of a prior insert into an SQL database table. With careful application design, these errors can be handled to maintain consistency and enhance data integrity and availability for applications.

An application can also perform messaging operations within local units of work. In a local unit of work, WebSphere MQ starts a TMF transaction for the first MQPUT, MQPUT1, or MQGET call that is performed within syncpoint control at a time when there is no current TMF transaction. A local unit of work can be used only for WebSphere MQ messaging operations, not for operations on external databases. A local unit of work is committed using the MQCMIT call, or backed out using the MQBACK call.

If an application calls MQGET with the wait option to start a local unit of work, the queue manager does not begin the TMF transaction for the unit of work until a suitable message is available. This feature ensures that the MQGET call does cause a long running TMF transaction that might require too many TMF audit trail files to be retained for an indefinite period, and therefore cause system wide problems with transaction processing.

When WebSphere MQ performs a syncpoint MQPUT or MQGET operation, it adjusts the queue depth at the time of the operation on the assumption that the transaction will eventually be committed. Thus the queue depth includes the number of uncommitted messages that are on the queue as well as the committed ones. If the transaction is eventually backed out, either deliberately or due to failure, WebSphere MQ adjusts the queue depth to maintain a fully accurate value.
Database consistency

"OpenTMF" describes the mechanism by which this is possible.

OpenTMF

OpenTMF is the informal name for an internal interface to the TMF product, which WebSphere MQ, with HP’s assistance, has been able to use to determine the outcome of a transaction that the queue manager uses to perform syncpoint messaging operations.

OpenTMF allows WebSphere MQ to register as a participant in any transaction it has inherited from applications. TMF then sends WebSphere MQ a notification at the completion of the transaction to tell it whether the transaction completed successfully or was backed out. This notification from TMF allows WebSphere MQ to:

- Keep accurate queue depth counts under all conditions
- Keep other internal status information relating to local queues accurate
- Control the availability of nonpersistent messages involved in syncpoint operations
- Improve the efficiency and response time of waited MQGET operations

From the system administrator’s point of view, the use of OpenTMF is visible in only one way: WebSphere MQ processes are visible in a list of resource managers that can be produced using the STATUS RESOURCEMANAGER command of the TMFCOM utility. All WebSphere MQ processes that use OpenTMF appear in this list as VOLATILE resource managers named automatically by TMF.

No special administrative actions are required for this use of TMF. WebSphere MQ uses and manages it automatically. You must ensure that the RMOPENPERCPU (maximum number of VOLATILE and RECOVERABLE resource managers per CPU) configuration parameter of TMF is set to a value that is larger than the maximum number of queue servers and status servers that can run in a single CPU across the system. Note that you need to allow for Backup processes since these servers are NonStop process pairs. The default value of 128 is usually adequate for most installations. The TMF Planning and Configuration Guide describes the subject of resource managers and heterogeneous transaction processing.

NonStop Tuxedo

WebSphere MQ can coordinate messaging operations for OSS applications using NonStop Tuxedo, since this product is based on TMF and uses the same facilities for heterogeneous transaction processing as does WebSphere MQ.

For information about the NonStop Tuxedo transaction environment and how it interacts with TMF, see the NonStop Tuxedo documentation.

Interleaved application transactions

With WebSphere MQ for HP NonStop Server, applications can take advantage of the unique transaction environment in ways that are not possible on other platforms. In general, on NonStop OS, a process can manage multiple transactions concurrently. An update to an audited database is always performed under the control of the current transaction and the application can switch to any one of the other active transactions before committing any of them. An application can therefore participate in multiple global units of work at the same time.
WebSphere MQ critical database files

The critical database files for a WebSphere MQ queue manager are described in Table 3.

Table 3. Critical database files

<table>
<thead>
<tr>
<th>Descriptive name</th>
<th>Location/name</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object catalog</td>
<td>Queue manager’s subvolume/AMQCAT and AMQCAT1</td>
<td>Holds the attributes of the queue manager object and the attributes of each queue, process, and namelist object.</td>
</tr>
<tr>
<td>Channel definitions</td>
<td>OSS queue manager’s directory/AMQRFCDADAT</td>
<td>Holds the definition of each channel of type SENDER, SERVER, RECEIVER, REQUESTER, SVRCONN, CLUSSDR, or CLUSRCVR.</td>
</tr>
<tr>
<td>Client channel definitions</td>
<td>OSS queue manager’s directory/AMQCLCHL.TAB</td>
<td>Holds the definition of each channel of type CLNTCONN.</td>
</tr>
<tr>
<td>OAM database</td>
<td>Queue manager’s subvolume/AMQOAM</td>
<td>Holds permissions (access rights) for each object and OAM principal authorized to access the queue manager.</td>
</tr>
<tr>
<td>Principal database</td>
<td>Queue manager’s subvolume/AMQPDB and AMQPDBA</td>
<td>Holds the name of each authorized OAM principal and the NonStop OS user identifier that the principal corresponds to.</td>
</tr>
<tr>
<td>Namelist files</td>
<td>Queue manager’s subvolume/Nxxxxxxx</td>
<td>A namelist file holds the list of names in a namelist.</td>
</tr>
<tr>
<td>Queue files</td>
<td>Queue manager’s subvolume/Qxxxxxxx</td>
<td>A queue file holds the header information, and some of the application data, for every persistent message in a local queue.</td>
</tr>
<tr>
<td>Queue overflow files</td>
<td>Queue manager’s subvolume/Oxxxxxxx</td>
<td>A queue overflow file holds the application data for all medium to large persistent messages in a local queue.</td>
</tr>
<tr>
<td>Message overflow files</td>
<td>Queue manager’s subvolume/Mxxxxxxx</td>
<td>A message overflow file holds the application data for a very large persistent message.</td>
</tr>
<tr>
<td>Object touch files</td>
<td>Queue manager’s subvolume/Txxxxxxx</td>
<td>An object touch file is used to record changes to the attributes of a WebSphere MQ object.</td>
</tr>
</tbody>
</table>

Critical processes

Table 4 on page 106 describes the critical processes of the queue manager, and shows how WebSphere MQ is protected from and can recover from their failure due to software or system failures. In the table below, disaster refers to cases of multiple system failures, or total system loss.
## Critical processes

### Table 4. Protection methods used for critical processes

<table>
<thead>
<tr>
<th>Process</th>
<th>Protection methods used</th>
<th>Recovery processing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Queue server</td>
<td>NonStop OS process pair. Maintains accurate status of local queues, other WebSphere MQ object, and messages at all times, except in cases of disaster.</td>
<td>Re-initializes from audited databases after catastrophic failure. No other recovery required. Nonpersistent messages are lost in cases of disaster.</td>
</tr>
<tr>
<td>Channel server</td>
<td>NonStop OS process pair. Maintains accurate status of channels at all times, except in cases of disaster.</td>
<td>Re-initializes from audited databases after catastrophic failure. No other recovery required.</td>
</tr>
<tr>
<td>Local queue manager agent</td>
<td>Connection is marked as broken for the application (2009). The repository manager garbage collection cleans up registration areas of the cluster cache that are left by failed processes. Execution controller, queue servers, and channel server notice a failure immediately by means of a NonStop OS IPC connection and correct or adjust the status data appropriately. TMF backs out any transaction that was active and used by the process at the time of failure.</td>
<td>None. A local queue manager agent process services one connection if running without threads, and potentially multiple connections if running with threads. The affected connections are dropped if a failure occurs, and new connections must be initiated by the applications.</td>
</tr>
</tbody>
</table>
Table 4. Protection methods used for critical processes (continued)

<table>
<thead>
<tr>
<th>Process</th>
<th>Protection methods used</th>
<th>Recovery processing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Message channel agent (MCA)</td>
<td>Channel server notices a failure immediately by means of a NonStop OS IPC connection and marks the channel status appropriately.</td>
<td>None. An MCA process services one channel if running without threads, and multiple channels if running with threads. Affected channels stop if a failure occurs and automated facilities of WebSphere MQ can be used to restart the channels. WebSphere MQ features, such as channel heartbeats and queue manager clusters, can also be used to increase the availability of channels.</td>
</tr>
<tr>
<td></td>
<td>Channel server ensures retry of outbound channels that fail.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Adopt MCA feature can be used to allow restart of the failed channel.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Channel synchronization data is audited by TMF and is used by WebSphere MQ to preserve the integrity of the channel. If one end of a channel is left in doubt, and the situation cannot be resolved automatically, standard administrative facilities exist for resolving it.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>All other protection methods are the same as those for the local queue manager agent process.</td>
<td></td>
</tr>
<tr>
<td>Channel initiator</td>
<td>Multiple channel initiators can be configured to provide higher availability, where each channel initiator monitors a different channel initiation queue.</td>
<td>None. The default channel server maintains accurate channel status information under all conditions and is responsible for channel retry. A channel initiator uses the triggering facilities of WebSphere MQ to detect when a channel needs to be started.</td>
</tr>
<tr>
<td></td>
<td>A channel initiator is normally run as a Pathway server class and configured to restart automatically on failure.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>You can use standard Pathway configuration options to specify alternative CPUs that can be used in the event of a CPU failure.</td>
<td></td>
</tr>
<tr>
<td>Command server</td>
<td>Command server performs administrative commands within syncpoint control, so that consistency is maintained.</td>
<td>None. Restart causes a new connection to the queue manager.</td>
</tr>
<tr>
<td></td>
<td>Standard Pathway protection features as described for the channel initiator.</td>
<td></td>
</tr>
</tbody>
</table>
### Table 4. Protection methods used for critical processes (continued)

<table>
<thead>
<tr>
<th>Process</th>
<th>Protection methods used</th>
<th>Recovery processing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execution controller</td>
<td>NonStop OS process pair. Maintains accurate status of queue manager processes at all times, except in cases of disaster.</td>
<td>Re-initializes from configuration files. No other recovery is required.</td>
</tr>
<tr>
<td>TCP/IP listener</td>
<td>Multiple TCP/IP listener processes can be configured to provide higher availability. Standard Pathway protection features as described for the channel initiator.</td>
<td>None. On restart, a TCP/IP listener attempts to listen on any of the ports configured for the queue manager that are available.</td>
</tr>
<tr>
<td>Repository manager</td>
<td>On failure of a repository manager, current or new users of the cluster cache in the same CPU experience no interruption of access to the cache or clustered operations. A repository manager maintains a consistent, hardened version of the cluster cache on the cluster repository queue at all times. Making a change to the cluster cache, and getting the message that caused the change from the cluster command queue, are done within the same unit of work. In this way, the consistency of the information in the cluster cache is maintained even if the repository manager fails during the hardening process. The execution controller restarts a repository manager on failure. A new repository manager attaches to the cluster cache if the cache is still in memory. Otherwise, the cluster cache is reloaded from disk provided it is present there. If the disk file does not exist, a new cluster cache is created in the CPU and initialized from the cluster repository queue.</td>
<td></td>
</tr>
<tr>
<td>Queue manager server</td>
<td>A queue manager server removes an expired message from a queue, and generates the expiration report, within the same unit of work so that a failure of the server process does not cause any inconsistency. Standard Pathway protection features as described for the channel initiator.</td>
<td>A restarted queue manager server re-synchronizes automatically with each queue server when the queue server reports the current set of expired messages in its queues during housekeeping activity.</td>
</tr>
<tr>
<td>Process</td>
<td>Protection methods used</td>
<td>Recovery processing</td>
</tr>
<tr>
<td>---------</td>
<td>-------------------------</td>
<td>---------------------</td>
</tr>
<tr>
<td>Application (using standard binding)</td>
<td>Failures within a queue manager are detected by a broken connection or unexpected errors from MQI calls. The application must disconnect from the queue manager and reconnect. The queue manager detects the failure of an application process immediately. The queue manager performs an implicit disconnect on behalf of the application, causing all open resources to be closed and released. Any units of work currently in progress are backed out by TMF, and the queue manager makes the appropriate adjustments to its local queues. Application code that contains errors is not able to corrupt queue manager critical databases or shared resources because the only shared memory that is in the address space of the application is read-only and no critical database files are accessed directly.</td>
<td>None. When an application starts again, it reconnects to the queue manager.</td>
</tr>
</tbody>
</table>
## Table 4. Protection methods used for critical processes (continued)

<table>
<thead>
<tr>
<th>Process</th>
<th>Protection methods used</th>
<th>Recovery processing</th>
</tr>
</thead>
</table>
| Application (using fastpath binding) | Failures within a queue manager are detected by unexpected errors from MQI calls. The application must disconnect from the queue manager and reconnect.  
  
The queue manager detects the failure of an application process immediately. The queue manager performs an implicit disconnect on behalf of the application, causing all open resources to be closed and released.  
  
Any units of work currently in progress are backed out by TMF, and the queue manager makes the appropriate adjustments to its local queues.  
  
Application code that contains errors is able to corrupt certain critical databases of the queue manager. However, queue structures and messages are safe because only queue servers can access them. | None. When an application starts again, it reconnects to the queue manager. |

## Queue manager clusters

Queue manager clusters are aimed at reducing the administration requirements of a WebSphere MQ network and also to enhance the overall availability and scalability of WebSphere MQ as a distributed service.

Queue Managers that belong to clusters can MQPUT to queues that are advertised to the cluster as if they are local queues. The WebSphere MQ clustering function deals with the administration and management of all the definitions and channels required to transfer the message to the destination queue.

Clustered queues may be defined on more than one queue manager within a cluster. This creates multiple instances of a queue within the cluster. An application puts to only one instance of a queue as chosen by the Cluster Workload Manager (CWLM), a component of WebSphere MQ. This choice may be made when the queue is opened, or dynamically for every put. The CWLM can determine the best instance of a cluster queue to use based on whether the channel to the instance is running or not, and on certain other factors like network priority and also application consideration via the Cluster Workload Management Exit.

Clusters can therefore provide an WebSphere MQ network-level availability enhancement. WebSphere MQ for HP NonStop Server is a good choice to act as a full repository for clusters due to the reliability and scalability of its operation.
Configuration considerations for availability

This section summarizes the configuration options enhancing the availability of WebSphere MQ for HP NonStop Server and its applications on NonStop OS:

- Configure Pathway with alternate CPUs for all server classes to protect against CPU failures.
- Consider the use of clustering for enhanced availability of WebSphere MQ network resources.
- Consider the use of nonpersistent messages with checkpointing enabled to obtain high performance with high availability for suitable message types.
- Ensure that your TMF configuration is sized to cope with the peak predicted demand of WebSphere MQ and its applications.
- Consider using message overflow files to reduce the audit trail requirement for very large messages.
- If a disaster recovery requirement exists, consider the use of RDF for creating and maintaining a backup site for WebSphere MQ.
- Ensure that CPUs run with enough available physical memory to cope with peak demands of WebSphere MQ and its applications. Where appropriate, use local queue manager agents and MCAs that run as threads in order to reduce physical memory requirements.
- Ensure that sufficient swap space is available for the CPUs that hold WebSphere MQ and its applications.

Configuration considerations for data integrity

This section summarizes the configuration options enhancing the data integrity of WebSphere MQ for HP NonStop Server and its applications on NonStop OS:

- Determine which message or transaction types carried by WebSphere MQ require which level of data integrity as provided by persistent messages and nonpersistent messages.
- Determine whether message overflow files are a suitable storage mechanism for storing any very large messages that you need to use.
- Determine whether nonpersistent messages require checkpointing or whether only some do. The different types of message should be put to different queues to enable different checkpointing options to be specified.
- Ensure that when applications require the highest data integrity that syncpoint operations using persistent messages are employed.
- If a disaster recovery requirement exists, configure and use RDF to create and maintain a duplicate backup of the WebSphere MQ environment. If your applications use very large persistent messages, make sure that the queues containing the messages use audited message overflow files so that operations on the contents of the files can be replicated to a backup system using RDF.
Configuration considerations for data integrity
Part 4. Configuring WebSphere MQ for HP NonStop Server

Chapter 9. Configuring WebSphere MQ . . . . 115
Changing configuration information in Pathway . . . . 115
The server classes of a queue manager . . . . 116
The attributes of server classes . . . . 116
  Attributes that must have specific values and
  must not be modified . . . . 117
  Attributes whose values can be modified . . . . 118
Modifying the attributes of server classes . . . . 121
  Changing the attributes of a server class
  while the queue manager is still running . . . . 121
Changing the attributes of a server class
  when the queue manager is not running . . . . 123
Adding new server classes . . . . 124
Removing server classes . . . . 126
Modifying the PATHWAY attributes of the
queue manager’s Pathway configuration . . . . 126
Changing configuration information in
configuration files . . . . 130
Editing configuration files . . . . 130
  When do you need to edit a configuration
  file? . . . . 131
  Configuration file priorities . . . . 131
  The WebSphere MQ configuration file, mq.ini . . . . 131
The default process management rules
configuration file, proc.ini . . . . 132
The queue manager configuration file, qm.ini . . . . 134
The process management rules configuration
file, qmproc.ini . . . . 134
The contents of a WebSphere MQ configuration file
  All queue managers . . . . 134
  Default queue manager . . . . 135
  Exit properties . . . . 136
API exits . . . . 137
Queue managers . . . . 137
The contents of a queue manager configuration file
  Installable services . . . . 137
  Service components . . . . 138
  Restricted mode . . . . 138
Channels . . . . 139
TCP . . . . 140
Exit path . . . . 141
API exits . . . . 141

Chapter 10. WebSphere MQ security . . . . 143
Authority to administer WebSphere MQ . . . . 143
  Managing the MQM group . . . . 144
Authority to work with WebSphere MQ objects . . . . 144
  When security checks are made . . . . 144
  How access control is implemented by
WebSphere MQ. . . . . 145
Identifying the user . . . . 146
  Principals and groups . . . . 146
Alternate user authority . . . . 147
Context authority . . . . 148
Creating and managing groups . . . . 148
  Creating a group . . . . 148
  Adding a user to a group . . . . 149
  Displaying who is in a group . . . . 149
  Removing a user from a group . . . . 149
Using the OAM to control access to objects . . . . 149
  Giving access to a WebSphere MQ object . . . . 149
  Examples of using the command . . . . 150
  Using the command with a different
  authorization service . . . . 150
Using OAM generic profiles . . . . 151
  Using wildcard characters . . . . 151
  Profile priorities . . . . 151
Dumping profile settings . . . . 152
Displaying access settings . . . . 153
Changing and preventing access to a WebSphere
MQ object . . . . 153
Preventing access control checks . . . . 153
Channel security . . . . 153
  Operating on channels, channel initiators, and
  listeners . . . . 155
  Transmission queues . . . . 155
  Channel exit programs . . . . 155
  Protecting channels with SSL . . . . 156
How authorizations work . . . . 157
  Authorizations for MQI calls . . . . 157
  Authorizations for MQSC commands in escape
  PCFs . . . . 160
  Authorizations for PCF commands . . . . 160

Chapter 11. Working with the WebSphere MQ
Secure Sockets Layer (SSL) support . . . . 163
Introduction to OpenSSL . . . . 163
Where the files containing the WebSphere MQ SSL
support code are installed . . . . 164
The entropy daemon . . . . 165
  Configuring and running the entropy daemon
  Stopping the entropy daemon . . . . 166
Preparing to use the WebSphere MQ SSL support
  Verifying that the WebSphere MQ SSL support
  is installed . . . . 166
  Verifying that the entropy daemon is running
  Deciding how to specify the configuration file
  for the openssl req command . . . . 167
Working with keys and digital certificates . . . . 167
  Generating public and private keys, and a
  request for a personal certificate . . . . 168
  Importing digital certificates . . . . 170
Preparing the queue manager’s SSL files . . . . 170
  Creating the queue manager’s SSL files . . . . 171
  Finding out where the queue manager’s SSL
  files are stored . . . . 171
  Changing the directory where the queue
  manager’s SSL files are stored . . . . 171
  When changes to the queue manager’s SSL
  files become effective . . . . 172
A sample configuration for testing . . . . 172

© Copyright IBM Corp. 2006
113
The sample shell scripts and MQSC command files
Building and verifying the sample configuration
Running one of the queue managers on another system

Chapter 12. Transactional support
Introducing units of work
Using TMF for local and global units of work
Using global units of work
Using local units of work
Avoiding long running transactions
Syncpoint limits
Performing operations on persistent messages outside of syncpoint control
Performing operations on nonpersistent messages within a unit of work
The number of concurrent active transactions for an application
Configuring TMF for WebSphere MQ
Monitoring
Audit trail size
Resource manager configuration
Troubleshooting

Chapter 13. The WebSphere MQ dead-letter queue handler
Invoking the DLQ handler
The sample DLQ handler, amqsd1q
The DLQ handler rules table
Control data
Rules (patterns and actions)
The pattern-matching keywords
The action keywords
Rules table conventions
How the rules table is processed
Ensuring that all DLQ messages are processed
An example DLQ handler rules table

Chapter 14. Process management
Attributes and rules
Default process attributes
Agent attributes
Application rules
Channel rules
Repository manager
Keyword definitions
Process management examples
Example 1: Configuring attributes for all agents
Example 2: Configuring attributes for types of agent
Example 3: Using threaded agent attributes
Example 4: Using the Repository stanza
Example 5: Using channel rules
Example 6: Using application rules
Example 7: Using environment variables
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.

On WebSphere MQ for HP NonStop Server, configuration information is stored in configuration files and in the Pathway database of a queue manager.

The chapter contains the following sections:

- “Changing configuration information in Pathway” on page 130
- “Changing configuration information in configuration files” on page 134
- “The contents of a WebSphere MQ configuration file” on page 137
- “The contents of a queue manager configuration file” on page 137

Although the default process management rules configuration file and the process management rules configuration file are discussed in this chapter, see Chapter 14, “Process management,” on page 197 for a more detailed description of their contents.

Changing configuration information in Pathway

Certain server processes in a queue manager run as instances of server classes under the control of a Pathway environment. Each server class in Pathway has a separate set of configuration information that specifies attributes of the server class, such as a list of the CPUs in which the server process can run, the home terminal to be used by the server process, and the name of the program that runs when the server process starts.

The server classes are created initially with attributes whose values are supplied by the crtmqm control command. These values are either default values or, in some cases, values specified by the user. For example, using the -nh parameter on the crtmqm command, you can specify the home terminal to be used by the queue manager.

You must not change the values of certain attributes because WebSphere MQ requires the specific values set by the crtmqm command in order to work properly. However, you can change the values of other attributes according to your requirements.

You might also need to add new server class definitions to Pathway to extend the capabilities of a queue manager, or remove server class definitions from Pathway that you no longer require.

This section contains the following topics:

- “The server classes of a queue manager” on page 116
- “The attributes of server classes” on page 116
- “Modifying the attributes of server classes” on page 121
Configuration information in Pathway

- “Adding new server classes” on page 124
- “Removing server classes” on page 126
- “Modifying the PATHWAY attributes of the queue manager's Pathway configuration” on page 126

The server classes of a queue manager

A server class, as used by WebSphere MQ, is a configuration entry in the Pathway database of a queue manager. The server class specifies attributes for a single server process or, in some cases, a process-pair. A queue manager is created with the server classes listed in Table 5. The names of all of the server classes begin with the characters MQS-. You cannot change the names of these server classes, and you must not delete any of them.

Table 5. Server classes that a queue manager is created with

<table>
<thead>
<tr>
<th>Server class name</th>
<th>Queue manager server process</th>
<th>OSS server process or NonStop process-pair?</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQS-CHANINIT00</td>
<td>Default channel initiator</td>
<td>OSS server process</td>
</tr>
<tr>
<td>MQS-CHANNELSVR</td>
<td>Channel server</td>
<td>NonStop process-pair</td>
</tr>
<tr>
<td>MQS-CMDSERV00</td>
<td>Command server</td>
<td>OSS server process</td>
</tr>
<tr>
<td>MQS-EC</td>
<td>Execution controller</td>
<td>NonStop process-pair</td>
</tr>
<tr>
<td>MQS-MQMSVR00</td>
<td>Default Monitoring panels server</td>
<td>OSS server process</td>
</tr>
<tr>
<td>MQS-QMGRSVR00</td>
<td>Queue manager server</td>
<td>OSS server process</td>
</tr>
<tr>
<td>MQS-QUEUE00</td>
<td>Default queue server</td>
<td>NonStop process-pair</td>
</tr>
<tr>
<td>MQS-TCPLIS00</td>
<td>Default TCP/IP listener</td>
<td>OSS server process</td>
</tr>
<tr>
<td>MQS-TRIGMON00</td>
<td>Default trigger monitor</td>
<td>OSS server process</td>
</tr>
<tr>
<td>MQS-DEFMSGSVR</td>
<td>Deferred message server</td>
<td>OSS server process</td>
</tr>
</tbody>
</table>

In general, a Pathway server class can be used to represent multiple server processes of the same type, but WebSphere MQ does not use this capability. If you require multiple server processes of the same type to run in parallel, you must create additional server classes, modelling the additional server classes on the default server class that is created when you create the queue manager. Note, however, that only queue servers, channel initiators, TCP/IP listeners, trigger monitors, and Monitoring Panels servers support parallel operation within a queue manager. For more information about queue manager processes, see “An overview of the queue manager processes” on page 77.

A queue server is particularly important because it manages messages, queues, and other WebSphere MQ objects for a queue manager. In order to improve message throughput, and increase the amount of memory available for storing messages, you usually need to configure additional queue servers.

For information about how to configure additional server classes, see “Adding new server classes” on page 124.

The attributes of server classes

For WebSphere MQ to work properly, some attributes of the server classes must have specific values, which must not be modified. You can, however, modify the values of other attributes to meet to your requirements. This section describes both sets of the attributes.
Attributes that are not described in this section are not relevant to the operation of WebSphere MQ. They are given default values when a server class is created initially, but the values are not used.

**Attributes that must have specific values and must not be modified**

Table 6 lists those attributes of the server classes that must have specific values and must not be modified. If you configure an additional server class that is modelled on one of the default server classes, the corresponding attributes of the new server class must also have the values specified in the table.

Table 6. Attributes whose values must not be modified

<table>
<thead>
<tr>
<th>Attribute name</th>
<th>Meaning</th>
<th>Value</th>
</tr>
</thead>
</table>
| PROCESSTYPE   | Whether the server process is an OSS server process or a NonStop process-pair | • OSS, for an OSS server process  
• GUARDIAN, for a NonStop process-pair |
| DEBUG         | Whether the server process starts under the control of the symbolic debugger | OFF |
| HIJHPIN       | Whether the server process runs in high pin mode | ON |
| MAXSERVERS    | The maximum number of server processes that can run | 1 |
| NUMSTATIC     | The minimum number of server processes that must run | 1 |
| OWNER         | The owner of the server class | The user ID of the user who ran the installation script, instmqm. |
| SECURITY      | The permitted level of access to the server class | N |
| PROGRAM       | The name of the program that runs when the server process starts | Depends on the server class |
| VOLUME        | The subvolume for the server process | The queue manager’s subvolume |

The following attributes are environment variables that are required for all OSS server processes:

**ENV MQQUEMGRNAME**

The name of the queue manager

**ENV MQNSKVARPATH**

The fully qualified OSS path name of the var/mqm directory for the installation

**ENV MQNSKOPTPATH**

The fully qualified OSS path name of the opt/mqm directory for the installation

**ENV _RLD_LIB_PATH**

The fully qualified OSS path name of the opt/mqm/lib directory for the installation

The following attributes are environment variables that are required for all NonStop process-pairs:
Attributes of the server classes

PARAM MQQUEMGRNAME
The name of the queue manager

PARAM MQNSKVARPATH
The fully qualified OSS path name of the var/mqm directory for the installation

PARAM MQNSKOPTHOPATH
The fully qualified OSS path name of the opt/mqm directory for the installation

Attributes whose values can be modified
Table 7 lists those attributes of the server classes whose values can be modified according to your requirements.

Table 7. Attributes whose values can be modified

<table>
<thead>
<tr>
<th>Attribute name</th>
<th>Meaning</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARGLIST</td>
<td>The command line arguments (for an OSS server process only)</td>
<td>Depends on the type of the server process</td>
</tr>
<tr>
<td>AUTORESTART</td>
<td>The maximum number of times that Pathway attempts to restart the server process after a failure</td>
<td>10</td>
</tr>
<tr>
<td>CPUS</td>
<td>A list of the CPUs in which the server process can run</td>
<td>(0:1)</td>
</tr>
<tr>
<td>HOMETERM</td>
<td>The home terminal of the server process</td>
<td>$ZHOME</td>
</tr>
<tr>
<td>OUT</td>
<td>The standard output device of the server process</td>
<td>$ZHOME</td>
</tr>
</tbody>
</table>
| PRI            | The priority of the server process
• 175, for an OSS server process
• 176, for a NonStop process-pair |
| PROCESS        | The name of the server process | None |

The following attribute is an environment variable that is required for all queue servers:

PARAM WMQ^QS^NAME
The symbolic name of the queue server. The value of this environment variable for the default queue server must be DEFAULTQS, but you can choose the value for any additional queue server.

For information about other environment variables used by WebSphere MQ, see Appendix F, “Environment variables,” on page 445.

The following sections describe in more detail the attributes whose values can be modified.

Attributes that can be modified for all server processes: The following attributes apply to all server processes of a queue manager that are configured in Pathway:

AUTORESTART
The maximum number of times that Pathway attempts to restart the server process within a defined period of time. If there is a persistent problem that is causing a server process to fail, this attribute ensures that Pathway does not attempt to restart the server process indefinitely. The default value of the attribute is 10, which is suitable for most configurations.
Attributes of the server classes

**CPUS** A list of CPUs in which the server process can run. If more than one CPU is specified, Pathway can restart the server process in another CPU if the CPU in which the server process was running fails. The default value of the attribute is (0:1), which means that the server process is initially started in CPU 0 but, if CPU 0 becomes unavailable, the server process can be restarted in CPU 1.

In order to balance the processing load, you might want to ensure that, under normal circumstances, certain server processes run in a specific CPUs.

**HOMETERM**

The home terminal of the server process. Every NonStop OS process has a home terminal. The home terminal must always be available, which means that it must exist and the process must be able to write to it.

The default home terminal is $ZHOME. $ZHOME is a write only device, which means that any data sent to it cannot be monitored or logged. Normally, a server process does not write a lot of information to its home terminal, and therefore $ZHOME is a good choice.

If you want to monitor the home terminal, use a Virtual Hometerm Subsystem (VHS) device. A VHS device is always available on NonStop OS and allows data written to it to be monitored and logged.

There is a limit on the number of processes that can have a home terminal open, but the limit depends on the type of the home terminal. You must therefore plan carefully to ensure that this limit is not exceeded. If necessary, use a VHS device, which has a relatively high limit.

When you create a queue manager, you can specify the home terminal for all the server classes of the queue manager by using the -nh parameter on the `crtmqm` command. Subsequently, you can change the home terminal of an individual server class.

In order to capture information about a problem, your IBM Support Center might ask you to modify the HOMETERM attribute so that it specifies an alternative device, such as a VHS device.

**OUT** The standard output device of the server process. Like the home terminal, the specified device must always be available.

For a newly created queue manager, the value of this attribute is the same as the value of the HOMETERM attribute for each server class of the queue manager. If you don’t use the -nh parameter on the `crtmqm` command to specify the home terminal for all the server classes of the queue manager, the default value of this attribute is $ZHOME for each server class. Subsequently, you can change the standard output device of an individual server class.

In order to capture information about a problem, your IBM Support Center might ask you to modify the OUT attribute so that it specifies an alternative device, such as a VHS device.

**PRI** The priority of the server process. The default value of the attribute is 175 for an OSS server process, or 176 for a NonStop process-pair. You can modify these values provided the priorities of NonStop process-pairs are higher than the priorities of OSS server processes.

You might need to adjust the priorities of some server processes in order to balance the CPU consumption of WebSphere MQ with that of applications.
Attributes of the server classes

and other subsystems in your environment. However, to avoid a possible performance degradation, always give queue servers and the channel server the highest priority.

PROCESS
The name of the server process. There is no default value for this attribute. If you do not specify a name for a server process, NonStop OS allocates one when the server process starts.

You can use the -ne parameter on crtmqm command to specify the name of the execution controller, the -nq parameter to specify the name of the default queue server, and the -nc parameter to specify the name of the channel server. However, you can change these names subsequently.

You are free to choose the name of any server process in order to help you identify and manage the processes in your system. However, the name you choose must be unique within the system. There is no dependency between the name of a server process and any other configuration data belonging to the queue manager, or the applications that connect to the queue manager.

The ARGLIST attribute of a channel initiator: Initially, the ARGLIST attribute of the default channel initiator has the following format:

ARGLIST -q,SYSTEM.CHANNEL.INITQ,-m,queue_manager_name

This causes the default channel initiator to monitor the default channel initiation queue, SYSTEM.CHANNEL.INITQ. To change the channel initiation queue monitored by the default channel initiator, replace SYSTEM.CHANNEL.INITQ by the name of a different channel initiation queue.

If you configure additional channel initiators for a queue manager, each one must monitor a different channel initiation queue. You can therefore use the ARGLIST attribute to specify the channel initiation queue for each channel initiator.

Alternatively, if you use the runmqchi command to start a channel initiator, you can specify the name of the channel initiation queue as a parameter on the command.

The ARGLIST attribute of a TCP/IP listener: Initially, the ARGLIST attribute of the default TCP/IP listener has the following format:

ARGLIST -t,tcp,-m,queue_manager_name

This causes the default TCP/IP listener to listen on port 1414, the default port for WebSphere MQ, and to use the default TCP/IP process, $ZTC0.

You can change the port that the default TCP/IP listener listens on by modifying the format of the ARGLIST attribute so that it specifies a different port number:

ARGLIST -t,tcp,-m,queue_manager_name,-p,port_number

If you configure additional TCP/IP listeners for a queue manager, each one must listen on a different port. You can therefore use the ARGLIST attribute to specify the port for each TCP/IP listener.

In a similar way, you can change the TCP/IP process that the default TCP/IP listener uses by modifying the format of the ARGLIST attribute so that it specifies a different TCP/IP process:

ARGLIST -t,tcp,-m,queue_manager_name,-g,TCP/IP_process_name
Alternatively, if you use the `runmqlsr` command to start a listener, you can specify the port number and TCP/IP process name as parameters on the command.

**The ARGLIST attribute of a trigger monitor:** Initially, the ARGLIST attribute of the default trigger monitor has the following format:

```
ARGLIST -m,queue_manager_name
```

This causes the default trigger monitor to monitor the default initiation queue, `SYSTEM.DEFAULT.INITIATION.QUEUE`.

You can change the initiation queue monitored by the default trigger monitor by modifying the format of the ARGLIST attribute so that it specifies the name of a different initiation queue:

```
ARGLIST -m,queue_manager_name,-q,initiation_queue_name
```

If you configure additional trigger monitors for a queue manager, each one must monitor a different initiation queue. You can therefore use the ARGLIST attribute to specify the initiation queue for each trigger monitor.

Alternatively, if you use the `runmqtrm` command to start a trigger monitor, you can specify the name of the initiation queue as a parameter on the command.

### Modifying the attributes of server classes

You can change the attributes of a server class only when the server process is not running. Normally, after you start a queue manager, only certain server processes of the queue manager are running. Other server processes run only when their function is required.

You can change the attributes of the server classes for the execution controller, channel server, queue servers, deferred message server, and queue manager server only when the queue manager is not running. These server processes must run continuously while the queue manager is running. Other server processes can be stopped while the queue manager is running, but their function becomes unavailable until they are restarted. For example, if a channel initiator is stopped while the queue manager is running, no channels can be started using that channel initiator until the channel initiator is restarted.

### Changing the attributes of a server class while the queue manager is still running

To change the attributes of a server class while the queue manager is still running, you must stop the server process. Before you can stop the server process, you must first freeze it by using the PATHCOM command `FREEZE SERVER server_class_name`. For example, the following command freezes the default channel initiator:

```
*freeze server mqs-chaninit00
$X0DM: SERVER MQS-CHANINIT00, FROZEN
```

After you have frozen a server process, you can use the PATHCOM command `STOP SERVER server_class_name` to stop the server process. For example, the following command stops the default channel initiator:

```
*stop server mqs-chaninit00
$X0DM: SERVER MQS-CHANINIT00, STOPPED
```
Note that it might take several seconds to stop a server process. If the STOP SERVER command fails to stop a server process, you can stop it by using the OSS kill command or the TACL STOP command. In this example, the name of the channel initiator server process is $X0DM.

You can verify that a server process has been stopped by using the PATHCOM command STATUS SERVER *server_class_name*. See the following command, for example:

```
=status server mqs-chaninit00
SERVER           #RUNNING  ERROR  INFO
MQS-CHANINIT00   0  1018  40
```

This command verifies that no server processes are running for the server class MQS-CHANINIT00, the default channel initiator. Note that the ERROR and INFO fields in the displayed status information indicate that the request to stop the server process timed out. This is normal behavior for WebSphere MQ server processes.

To review the current attributes of a server class, use the PATHCOM command INFO SERVER *server_class_name*. For example, the following command displays the attributes of the server class for the default channel initiator:

```
=info server mqs-chaninit00
SERVER MQS-CHANINIT00
  PROCESTYPE    OSS
  ARGLIST       -q,SYSTEM.CHANNEL.INITQ,-m,SAMPLE_QMGR
  AUTORESTART   10
  CPUS          (0:1)
  CREATEDELAY   1 MINS
  DEBUG         OFF
  DELETEDELAY   10 MINS
  ENV           MQQUEMGRNAME=SAMPLE_QMGR
  ENV           MQNSKVARPATH=/home/prod_top/var/mqm
  ENV           MQNSKOPTPATH=/home/prod_top/opt/mqm
  ENV           _RLD_LIB_PATH=/home/prod_top/opt/mqm/lib
  HIGHPIN       ON
  HOMETERM      \*.$VHS
  LINKDEPTH     255
  MAXSERVERS    1
  NUMSTATIC     1
  OWNER         \ZAPHOD.44,255
  PRI           175
  PROGRAM       /home/prod_top/opt/mqm/bin/runmqchi
  SECURITY      "N"
  TMF           ON
  VOLUME        \*.$DATA01.SAMPLEQM
```

To change an attribute of a server class, use the PATHCOM command ALTER SERVER *server_class_name*, as in the following example:

```
=alter server mqs-chaninit00, arglist -m,SAMPLE_QMGR,-q,TEST.INITQ
```

This command changes the ARGLIST attribute of the server class for the default channel initiator so that the default channel initiator now monitors the channel initiation queue TEST.INITQ instead of SYSTEM.CHANNEL.INITQ.
To restart a server process, use the PATHCOM commands
THAW SERVER server_class_name and START SERVER server_class_name, in that order. For example, the following sequence of commands restart the default channel initiator:

```
thaw server mqs-chaninit00
$XOM: SERVER MQS-CHANINIT00, THAWED
start server mqs-chaninit00
$XOM: SERVER MQS-CHANINIT00, STARTED
```

**Changing the attributes of a server class when the queue manager is not running**

To change the attributes of a server class when the queue manager is not running, use the following procedure.

At a TACL command prompt, set your default subvolume to the queue manager’s subvolume. The queue manager’s subvolume is specified by the HPNSSGuardianSubvol entry in the QueueManager stanza for the queue manager in the WebSphere MQ configuration file, mqs.ini. The queue manager’s subvolume contains the Pathway database, PATHCTL, of the queue manager. See the following sequence of commands, for example:

```
SYSTEM 7> volume $DATA01.SAMPLEQM
$DATA01.SAMPLEQM 8> fileinfo PATHCTL
$DATA01.SAMPLEQM 9>
```

These commands set the default subvolume to $DATA01.SAMPLEQM, and verify that the file PATHCTL is in the default subvolume.

From your default subvolume, start the Pathway manager process, PATHMON, in the background. Use the same process name for PATHMON as when you run the queue manager. This process name is specified by the ProcessName entry in the Pathway stanza in the process management rules configuration file, qmproc.ini. For example, the following command starts PATHMON in the background with the process name $SWPM:

```
DATA01.SAMPLEQM 10> PATHMON /name $SWPM, NOWAIT/
```

Using the PATHCOM command START PATHWAY, load the Pathway configuration data for the queue manager from the Pathway database. See the following sequence of commands, for example:

```
DATA01.SAMPLEQM 12> PATHCOM $SWPM
$XOMH: PATHCOM - T6344D44 - (27SEP04)
(C)1980 Tandem (C)2003 Hewlett Packard Development Company, L.P.
start pathway cool
PATHWAY CONTROL FILE DATED: 04 OCT 2005, 20:58:26
status server *
```

<table>
<thead>
<tr>
<th>SERVER</th>
<th>#RUNNING</th>
<th>ERROR</th>
<th>INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQS-CHANINIT00</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MQS-CHANNELSVR</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MQS-CMDSERV00</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MQS-EC</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MQS-MQMSVR00</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MQS-QMGRSVR00</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Attributes of the server classes

| MQS-QUEUE00 | 0 |
| MQS-TCPLIS00 | 0 |
| MQS-TRIGMON00 | 0 |

These commands start PATHCOM, connecting to the PATHMON process started previously, load the Pathway configuration data for the queue manager, and verify that no server processes of the queue manager are running.

You can now use the ALTER SERVER server_class_name command to change the attributes of any server class, and use the INFO SERVER server_class_name command to verify the changes. When you have completed all your changes, save the new Pathway configuration data to the Pathway database by using the PATHCOM command SHUTDOWN2. After the configuration data has been saved, PATHMON ends automatically, and you can then exit from PATHCOM. For example, the following sequence of commands saves the new configuration data, stops PATHMON, and exits from PATHCOM:

```
=shutdown2
=exit
$DATA01 SAMPLEQM 13>
```

Start the queue manager for your configuration changes to take effect.

Adding new server classes

You can add new server classes to the Pathway configuration of a queue manager while the queue manager is still running or when it is not running.

If the queue manager is not running, you must first load the Pathway configuration data for the queue manager as described in "Changing the attributes of a server class when the queue manager is not running" on page 123. You can then use the procedure described in this section to add the new server classes. After adding the server classes, you must save the new Pathway configuration data to the Pathway database by using the PATHCOM command SHUTDOWN2.

Use the following procedure to add a new server class to the Pathway configuration of a queue manager that is still running. You can use the procedure to add a new server class only for a server process that is capable of parallel operation within a queue manager; that is, to add a new queue server, channel initiator, TCP/IP listener, trigger monitor, or Monitoring Panels server.

1. Create a working set of attributes for the new server class.

   Use the PATHCOM command SET SERVER LIKE template_server_class_name to perform this step, using as a template the default server class upon which you want to model the new server class. The command copies the attributes of the default server class into the working set. At this stage, the attributes in the working set do not belong to any server class. See the following sequence of commands, for example:

   ```
   =set server like mqs-queue00
   =show server
   SERVER
   PROCPRIORITY GUARDIAN
   AUTORESTART 10
   CPUS (0:1)
   CREATEDELAY 1 MINS
   DEBUG OFF
   DELETEDELAY 10 MINS
   HIGHPIN ON
   HOME TERM \*.VHS1
   ```
These commands create a working set of attributes copied from the server class of the default queue server and, using the PATHCOM command SHOW SERVER, display the current working set of attributes.

2. Customize the working set of attributes.

Use the PATHCOM commands SET SERVER, RESET SERVER, and SHOW SERVER to perform this step.

Normally, you need to change at least one of the attributes in the working set for the new server class. In the case of a queue server, for example, you might need to change several attributes:

- Typically, you might configure a new queue server to run in CPUs that are different from the ones in which the default queue server can run.
- A new queue server requires a different and unique symbolic name.
- If the template server class specifies a process name, you must remove this process name in the working set, or change it to a different and unique process name.

See the following sequence of commands, for example:

```
= set server param WMQ^QS^NAME "NEWQS"
= reset server process $SWQ5
= set server process $SWQ2
= show server

SERVER
  PROCESTYPE GUARDIAN
  AUTORESTART 10
  CPUS (2:3)
  CREATEDELAY 1 MINS
  DEBUG OFF
  DELETEDELAY 10 MINS
  HIGHPIN ON
  HOMETERM $.*.VHS1
  LINKDEPTH 255
  MAXSERVERS 1
  NUMSTATIC 1
  OUT $.*.VHS1
  OWNER $ZAPHOOD.44,8
  PARAM MQQUEMGRNAME "SAMPLE_QMGR"
  PARAM MQNSKVARPATH "*/home/prod_top/var/mqm"
  PARAM MQNSKOPTPATH "*/home/prod_top/opt/mqm"
  PARAM WMQ^QS^NAME "NEWQS"
  PRI 176
  PROCESS $SWQ5
  PROGRAM $.*.DATA01.SWPRODE.MQQSSVR
```

Chapter 9. Configuring WebSphere MQ
Attributes of the server classes

SECURITY "N"
TMF ON
VOLUME \*.DATA01.SAMPLEQM

These commands change the symbolic name by setting the environment variable WMQ\QS\NAME. They also remove the process name $SWQS, and then set the process name to $SWQ2. The SHOW SERVER command displays the current working set of attributes to validate that the intended changes have been made correctly.

3. Add the new server class to the Pathway configuration of the queue manager. Use the PATHCOM command ADD SERVER to perform this step. The command creates a new server class from the working set of attributes, adds the server class to the Pathway configuration of the queue manager, and assigns a unique name to the server class.

If you are adding a server class that is not a queue server, the name that you choose for the server class is not important. However, to avoid confusion, you might consider following the naming convention used by the corresponding default server class.

If you are adding a new queue server, the name of the server class must begin with the characters MQS-QUEUE in order for the queue server to be started automatically when the queue manager starts. The server class for the default queue server is called MQS-QUEUE00. Therefore, if you are adding two new queue servers, you can call their server classes MQS-QUEUE01 and MQS-QUEUE02. See the following sequence of commands, for example:

= add server mqs-queue01
= start server mqs-queue01
=

These commands add a new server class called MQS-QUEUE01 and start the queue server.

Removing server classes

Use the PATHCOM command DELETE SERVER server_class_name to remove a server class that you no longer require. You can remove a server class only when the server process is not running.

If the queue manager is not running, you must first load the Pathway configuration data for the queue manager as described in “Changing the attributes of a server class when the queue manager is not running” on page 123. You can then remove the server classes. After removing the server classes, you must save the new Pathway configuration data to the Pathway database by using the PATHCOM command SHUTDOWN2.

If you remove a server class while the queue manager is still running, the new configuration is saved in the Pathway database when you next stop the queue manager.

You must not remove any of the default server classes.

Modifying the PATHWAY attributes of the queue manager's Pathway configuration

The following list contains the names of the PATHWAY attributes of a queue manager's Pathway configuration and some sample values of the attributes:
You cannot modify these attributes dynamically. You must extract the complete Pathway configuration into a text file, modify the text file, and then recreate the Pathway database, PATHCTL, from the contents of the text file.

To modify the PATHWAY attributes, follow this procedure while the queue manager is running. In the procedure, PATHMON_process_name is the name of the PATHMON process of the queue manager. This process name is specified by the ProcessName entry in the Pathway stanza in the process management rules configuration file, qmproc.ini. The examples in the procedure use the process name $PM53.

1. At a TACL command prompt, set your default subvolume to the queue manager's subvolume.
   The queue manager's subvolume is specified by the HPNSSGuardianSubvol entry in the QueueManager stanza for the queue manager in the WebSphere MQ configuration file, mqs.ini. The queue manager's subvolume contains the Pathway database, PATHCTL, of the queue manager.
   For example, the following command sets the default subvolume to $DATA01.MQV53:
   ```
   VOLUME $DATA01.MQV53
   ```

2. Create an empty text file in your default subvolume.
   At the TACL command prompt, enter the following command:
   ```
   EDIT MQCFG
   ```
   MQCFG is the name of the file that you are creating, and a file with this name must not already exist in your default subvolume. You can choose your own name for the file but, if you do, remember to replace each occurrence of MQCFG by the name you choose in the steps that follow.
   When the text editor asks you whether you want the file to be created, enter y, and then enter exit to leave the editor.

3. At the TACL command prompt, enter the following command to start the Pathway control program, PATHCOM, connecting to the PATHMON process of the queue manager:
   ```
   PATHCOM PATHMON_process_name
   ```
   For example:
   ```
   PATHCOM $PM53
   ```
4. At the PATHCOM command prompt, enter each of the following commands in turn:

   INFO /OUT MQCFG/ PATHMON, OBEYFORM
   INFO /OUT MQCFG/ PATHWAY, OBEYFORM
   INFO /OUT MQCFG/ TCP *, OBEYFORM
   INFO /OUT MQCFG/ TERM *, OBEYFORM
   INFO /OUT MQCFG/ PROGRAM *, OBEYFORM
   INFO /OUT MQCFG/ SERVER *, OBEYFORM
   EXIT

   These commands extract the complete Pathway configuration of the queue manager into the text file MQCFG and then exit from PATHCOM.

5. From the TACL command prompt, edit the text file MQCFG, making any required changes to the PATHWAY attributes. At the same time, you can change any other attributes in the configuration, except those attributes of the queue manager's server classes that must not be modified. For a list of these attributes, see "Attributes that must have specific values and must not be modified" on page 117.

   After the last SET PATHWAY statement, insert the following statements:
   START PATHWAY COLD!
   LOG1 home_terminal

   The home terminal identified in the second statement can be the home terminal any of the server classes of the queue manager. Typically, all the server classes of a queue manager use the same home terminal.

6. At the TACL command prompt, enter the following command to stop the queue manager:

   endmqm queue_manager_name

7. When the queue manager has stopped, rename the Pathway database, PATHCTL, of the queue manager by entering the following command at the TACL command prompt:

   RENAME PATHCTL PATHO

   PATHO is the new name of the file, and a file with this name must not already exist in your default subvolume. If you wish, you can choose your own name for the file.

8. At the TACL command prompt, enter the following command to start the PATHMON process of the queue manager in the background:

   PATHMON /NAME PATHMON_process_name, NOWAIT/

   For example:
   PATHMON /NAME $PM53, NOWAIT/

9. At the TACL command prompt, enter the following command to create a new Pathway database, PATHCTL, for the queue manager from the contents of the text file MQCFG:

   PATHCOM /IN MQCFG/ PATHMON_process_name

   For example:
   PATHCOM /IN MQCFG/ $PM53

10. At the TACL command prompt, enter the following command to start the Pathway control program, PATHCOM, connecting to the PATHMON process of the queue manager:

    PATHCOM PATHMON_process_name
For example:
PATHCOM $PM53

11. Verify the changes you have made to the Pathway configuration of the queue manager.

At the PATHCOM command prompt, enter the following command to display the PATHWAY attributes of the queue manager’s Pathway configuration:
INFO PATHWAY

If you changed any other attributes of the configuration in step 5, such as the TERM attributes, you can display these attributes as well, as in the following example:
INFO TERM *

12. At the PATHCOM command prompt, enter the following command to stop the PATHMON process of the queue manager:
SHUTDOWN2

13. At the PATHCOM command prompt, enter the following command to exit from PATHCOM:
EXIT

14. At the TACL command prompt, enter the following command to verify that the PATHMON process has ended:
STATUS PATHMON_process_name

For example:
STATUS $PM53

If the PATHMON process has ended, the following response is displayed:
(Process does not exist)

If you do not receive this response, wait a short time and enter the command again. If the PATHMON process has still not ended, stop the process by using the STOP command, as in the following example:
STOP $PM53

15. At the TACL command prompt, enter the following command to start the queue manager again:
strmqm queue_manager_name

If you ever need to restore the original Pathway configuration of the queue manager, follow this procedure:
1. At a TACL command prompt, set your default subvolume to the queue manager’s subvolume, as in the following example:
VOLUME $DATA01.MQV53

2. At the TACL command prompt, enter the following command to stop the queue manager:
endmqm queue_manager_name

3. When the queue manager has stopped, rename the Pathway database, PATHCTL, of the queue manager by entering the following command at the TACL command prompt:
RENAME PATHCTL PATHN

PATHN is the new name of the file, and a file with this name must not already exist in your default subvolume. If you wish, you can choose your own name for the file.
Attributes of the server classes

4. At the TACL command prompt, restore the original Pathway database by renaming the file that was saved in step 7 of the previous procedure, as in the following example:
   RENAME PATHO PATHCTL

5. At the TACL command prompt, enter the following command to start the queue manager again:
   strmqm queue_manager_name

Changing configuration information in configuration files

You can change WebSphere MQ configuration attributes within:

- A WebSphere MQ configuration file, mqs.ini, to effect changes for one installation of WebSphere MQ. There is one mqs.ini file for each installation of WebSphere MQ on a NonStop OS system.
- A default process management rules configuration file, proc.ini, whose contents are used to form the process management rules configuration file, qmproc.ini, of a queue manager when you create the queue manager. The default process management rules configuration file contains a set of rules that you can customize to meet the requirements of all the queue managers created in an installation. There is one proc.ini file for each installation of WebSphere MQ on a NonStop OS system.
- A queue manager configuration file, qm.ini, to effect changes for a specific queue manager. There is one qm.ini file for each queue manager in an installation.
- A process management rules configuration file, qmproc.ini, which contains a set of rules that are used by the execution controller to manage those server processes of a queue manager that are not configured as server classes within Pathway. There is one qmproc.ini file for each queue manager in an installation.

A configuration file, or stanza file, contains one or more stanzas, which are groups of entries that together have a common function or define part of a system, such as 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 WebSphere MQ configuration file or a queue manager configuration file do not take effect until you restart the queue manager. Any changes you make to a default process management rules configuration file affect only queue managers that you create in the future. The changes have no effect on existing queue managers. Any changes you make to a process management rules configuration file do not take effect until you restart the queue manager or issue the MQSC command RESET QMGR TYPE(NSPROC), or the equivalent PCF command.

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 entries in configuration files after installation.

If you create an incorrect entry in a WebSphere MQ configuration file or a queue manager configuration file, the entry is ignored and an operator message is issued to indicate the problem. The effect is the same as omitting the entry completely. If you create an incorrect entry in a default process management rules configuration file, the error is discovered only when you next create a queue manager and start the queue manager. If an incorrect entry is encountered in a process management rules configuration file, the entire contents of the configuration file are ignored and an operator message is issued to indicate the problem. The queue manager continues by using a set of safe default rules.

After you create a new queue manager:
- Back up the WebSphere MQ configuration file
- Back up the new queue manager configuration file
- Back up the new process management rules 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 a backup if you can.)
- You need to change your default queue manager. This could happen if you accidentally delete the existing queue manager.
- You need local queue manager agents to run as processes, instead of threads, for certain applications.
- 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**
A WebSphere MQ configuration file, mqs.ini, contains information that is relevant to all the queue managers in an installation. It is created automatically during installation.

The WebSphere MQ configuration file for an installation is in the installation’s var/mqm directory. The file contains:
- The names of the queue managers
- The name of the default queue manager
- The location of the files associated with each of them

[Figure 22 on page 132](#) shows an example of a WebSphere MQ configuration file.
The default process management rules configuration file, proc.ini

When you create a queue manager, the contents of its process management rules configuration file, qmproc.ini, are derived from the default process management rules configuration file for the installation. The default process management rules configuration file is called proc.ini and is in the directory var_installation_path/var/mqm. The default process management rules configuration file is created when you install WebSphere MQ for HP NonStop Server, and contains a default set of rules, which you can customize to meet the requirements of all the queue managers created in the installation.

Figure 23 on page 133 shows an example of a default process management rules configuration file.
#*********************************************************************#
#
WMQ V5.3 for HP NonStop Server
#
# Template process management configuration file
# The contents of this file are copied to create the initial version
# of the qmproc.ini file for each queue manager that is created in
# this installation.
#
#*********************************************************************#

# All Processes stanza
# ---------------------

AllProcesses:

# Agent type stanzas
# ------------------

LQMA:
MCA:

# Application rules
# -----------------

AppRule1-ProcNameMatch:
AppRule2-ProcNameMatch:
AppRule3-ExeNameMatch:
AppRule4-SubvolMatch:
AppRule5-CpuMatch:

# Channel rules
# -------------

ChlRule1-ChannelNameMatch:
ChlRule2-ChannelTypeMatch:

ChlRule3-ChannelProtocolMatch:
  ChannelProtocolMatch=TCP/IP
  TransportName=$ZTC0

ChlRule3-ChannelProtocolMatch:
  ChannelProtocolMatch=SNA
  TransportName=$APC1

# Repository Manager stanza
# --------------------------

RepositoryManager:

# OAM Manager stanza
# -------------------

OamManager:

Figure 23. Example of a default process management rules configuration file

For more information about the default process management rules configuration file, see Chapter 14, “Process management,” on page 197.
Queue manager configuration file

The queue manager configuration file, 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_installation_path/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 18.

Figure 24 shows how groups of attributes might be arranged in a queue manager configuration file.

```
#*******************************************************************#
#* Module Name: qm.ini                                           *#
#* Type       : WebSphere MQ queue manager configuration file    *#
#* Function   : Define the configuration of a single queue manager *#
#*******************************************************************#
ExitPath:  
   ExitsDefaultPath=/home/prod_top/var/mqm/exits

Service:   
   Name=AuthorizationService  
   EntryPoints=10

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

CHANNELS:  
   MaxChannels=20 ; Maximum number of Channels allowed.  
   MaxActiveChannels=100 ; Maximum number of Channels allowed to  
                         ; be active at any time.

TCP:  
   KeepAlive=Yes ; Switch KeepAlive on
```

Figure 24. Example queue manager configuration file

The process management rules configuration file, qmproc.ini

For a complete description of the process management rules configuration file, including examples, see Chapter 14, “Process management,” on page 197.

The contents of a WebSphere MQ configuration file

You can change the configuration information in a WebSphere MQ configuration file by editing the file. The following sections describe in detail the contents of each of the stanzas in a WebSphere MQ configuration file.
All queue managers

Use the AllQueueManagers stanza in the mqs.ini file to specify the following information about all queue managers.

**DefaultPrefix=directory_name**
The path to the qmgrs directory where the OSS files of each queue manager are stored.

**HPNSSGuardianSubvol=volume_name.subvolume_name**
The fully qualified local name of the subvolume that contains the WebSphere MQ libraries and executable files for the NonStop OS environment.

**HPNSSQMDefaultGuardianVol=volume_name**
The name of the volume where, by default, the NonStop OS files of a queue manager are stored. When you create a queue manager, you can specify a different volume by using the -ns parameter on the **crtmqm** command.

**HPNSSSegidRange=lower_limit-upper_limit**
The range from which WebSphere MQ selects a segment ID when allocating a NonStop OS memory segment.

When an application is connected to a queue manager, certain WebSphere MQ code runs in the application’s process. When this code allocates a NonStop OS memory segment, by default, WebSphere MQ selects a segment ID in the range 500-1023. You must therefore ensure that the application code running in the same process does not allocate memory segments with segment IDs in the same range.

You can change the default range of segment IDs by including this entry in the WebSphere MQ configuration file. The new range must contain at least ten segment IDs and be wholly contained within the range 0-1023 permitted by NonStop OS.

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

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

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

NL_TO_LF is the default.

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

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

**ISO**
Convert:
- ISO CCSIDs using the TABLE method
- All other CCSIDs using the NL_TO_CF method
Contents of a WebSphere MQ configuration file

Possible ISO CCSIDs are shown in Table 8.

<table>
<thead>
<tr>
<th>CCSID</th>
<th>Code Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>819</td>
<td>ISO8859-1</td>
</tr>
<tr>
<td>912</td>
<td>ISO8859-2</td>
</tr>
<tr>
<td>915</td>
<td>ISO8859-5</td>
</tr>
<tr>
<td>1089</td>
<td>ISO8859-6</td>
</tr>
<tr>
<td>813</td>
<td>ISO8859-7</td>
</tr>
<tr>
<td>916</td>
<td>ISO8859-8</td>
</tr>
<tr>
<td>920</td>
<td>ISO8859-9</td>
</tr>
<tr>
<td>1051</td>
<td>roman8</td>
</tr>
</tbody>
</table>

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.

Default queue manager

Use the DefaultQueueManager stanza to specify the default queue manager for the installation.

**Name=default_queue_manager_name**

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 ExitProperties stanza 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 (amqzIwa0) 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.
Contents of a WebSphere MQ configuration file

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.

API exits
Use the ApiExitCommon and ApiExitTemplate stanzas to identify API exit routines for all queue managers in the installation. (To identify API exit routines for individual queue managers, use the ApiExitLocal stanza, as described in "API exits" on page 141.)

For a complete description of the attributes for these stanzas, see "Configuring API exits" on page 373.

Queue managers
There is one QueueManager stanza for every queue manager. The entries in a QueueManager stanza specify where the files of the queue manager are stored.

Name=queue_manager_name
The name of the queue manager.

Prefix=prefix
The path to the qmgs directory where the OSS files of the queue manager are stored. By default, this path is the same as that specified by the DefaultPrefix entry in the AllQueueManagers stanza.

HPNSSGuardianSubvol=volume_name.subvolume_name
The fully qualified local name of the subvolume where the NonStop OS files of the queue manager are stored. The subvolume component of the name is based on the queue manager name, but it can be a transformed name if a subvolume with the same name as the queue manager already exists, or if the queue manager name is not a valid subvolume name.

Directory=directory_name
The name of the directory within the prefix/qmgrs directory where the queue manager files are stored. This name is based on the queue manager name, but it can be a transformed name if a directory with the same name as the queue manager already exists, or if the queue manager name is not a valid directory name. For more information about name transformation, see "Understanding WebSphere MQ file names” on page 18.

The contents of a queue manager configuration file
The attributes described here modify the configuration of an individual queue manager. They override any settings in the WebSphere MQ configuration file. You can change the configuration information in a queue manager configuration file by editing the file.
Installable services

Use the Service stanza in the queue manager configuration file to specify information about an installable service. There must be one Service stanza for every service used.

**Name=AuthorizationService|NameService**

The name of the required service.

**AuthorizationService**

The Service stanza for the authorization service is generated automatically when the queue manager is created.

**NameService**

To enable a name service, you must add a Service stanza manually.

**EntryPoints=number_of_entries**

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

Service components

For each component of an installable service, you must use a ServiceComponent stanza to specify the name and path of the module containing the code for that component.

The ServiceComponent stanza for the default authorization service component, the Object Authority Manager (OAM), is generated automatically when the queue manager is created. You must add any other ServiceComponent stanzas manually.

**Service=service_name**

The name of the required service. This must match the value specified by the Name entry in the Service stanza.

**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 API exits,” on page 297.

Restricted mode

The RestrictedMode stanza is generated 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 generated.

**ApplicationGroup**

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

- Run MQI applications
- Change the contents of some queue manager directories
Channels

Use the Channels stanza 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 | STANDARD
The binding for channels:

FASTPATH
Channels use FASTPATH binding.

STANDARD
Channels use STANDARD binding.

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 | CLUSRRCVR | ALL | FASTPATH
If WebSphere MQ receives a request to start a channel, but finds that an MCA 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.
Contents of a queue manager configuration file

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:

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.
AdoptNewMCACheck=NAME,ADDRESS,QM is the default for FAP4 or later.

TCP
Use the TCP stanza to specify information about TCP/IP channels.

Port=1414|port_number
The default port number for TCP/IP connections. The well known port number for WebSphere MQ is 1414.

KeepAlive=YES|NO
Switch the TCP/IP keep alive 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=5|number
The maximum number of connection requests that can be waiting to be accepted by a TCP/IP listener. This limit applies if you do not specify a
different limit by using the -b parameter on the `runmqslr` command. If the TCP stanza does not contain this entry, the default limit is 5.

If the number of outstanding connection requests reaches the maximum value, the next connection request to arrive is rejected and the channel fails to start. A message channel then enters the RETRYING state. For a WebSphere MQ client application that is trying to connect to a queue manager over an MQI channel, the MQCONN or MQCONNX call fails with reason code MQRC_Q_MGR_NOTAVAILABLE. The client application must then try to connect at a later time.

**Exit path**

Use the ExitPath stanza to specify the path for user exit programs.

**ExitDefaultPath**=string

The ExitDefaultPath attribute specifies the location of channel exits and data conversion exits.

**API exits**

Use the ApiExitLocal stanza to identify API exit routines for a queue manager. (To identify API exit routines for all queue managers, use the ApiExitCommon and ApiExitTemplate stanzas, as described in “API exits” on page 137.)

For a complete description of the attributes for these stanzas, see “Configuring API exits” on page 373.
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 queues, processes, and namelists, 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 also need to ensure that channels can only be manipulated by authorized users.

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 perform the following tasks:
- Use 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. All members of this group have full access to all WebSphere MQ resources. This level of access can be revoked only by removing a user from the MQM group. User IDs that belong to the MQM group must not be available to non-privileged users.

You must create the MQM group before you run the installation script, instmqm, and the primary group of the user who runs the installation script must be MQM. However, any user who belongs to the MQM group can be an administrator for the installation; the primary group of the user does not have to be MQM, and the user does not have to be the one who ran the installation script.

You do not need to be a member of the MQM group to do the following:
- Issue commands from an application that issues PCF commands, or MQSC commands within an Escape PCF command, unless the commands manipulate channels, channel initiators, or channel listeners. (These commands are described in “Channel security” on page 153.)
Administration authority

- Issue MQI calls from an application (unless you want to use the fastpath bindings on the MQCONNX call).
- Use the `crtmqcvx` command to create a fragment of code that performs data conversion on data type structures.
- 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. They might also need to remove users who no longer need this authority. These tasks are described in "Creating and managing groups" on page 148.

Authority to work with WebSphere MQ objects

Queue managers, queues, processes, and namelists 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 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 the 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 applications that use full MQI context options
- Creating and deleting application queues

The user who creates a WebSphere MQ object is granted full control of that object, as are members of the MQM group.

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 call 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 call. 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 159.

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 call results 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 NonStop OS and Safeguard. 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 “Preventing access control checks” on page 153). 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 NonStop OS and Safeguard, using their user IDs and group names. Users can access WebSphere MQ objects only if they have the correct authority. “Using the OAM to control access to objects” on page 149 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. WebSphere MQ supplies a command to create and maintain access control lists; do not update this queue in any other way. This is described in “Using the OAM to control access to objects” on page 149.

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
Administration authority

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

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.

On NonStop OS, every queue manager has a principal database. Each entry in the principal database maps a WebSphere MQ principal to a NonStop OS user ID.

The crtmqm command automatically creates an entry in the principal database for the user who ran the installation script, instmqm. The principal created is always mqm, for compatibility with other WebSphere MQ implementations.

After you have created a queue manager, you can create entries in the principal database for the other users of the queue manager. To create an entry in the principal database, use the altmqusr command. For example, to create an entry for the WebSphere MQ principal mquser that maps to the NonStop OS user ID MQM.MQUSER, enter the following command:

```
altmqusr -m saturn.queue.manager -p mquser -u MQM.MQUSER
```

You can also use the altmqusr control command to alter or delete an entry in the principal database.

To display entries in the principal database, use the dspmqusr command. For example, to display information about all the principals that have entries in the principal database, enter the following command:

```
dspmqusr -m saturn.queue.manager
```

You can use the altmqusr and dspmqusr commands at any time, even when the queue manager is not running.

When an application attempts to connect to a queue manager, the queue manager obtains from NonStop OS the user ID under which the application is running. The queue manager then queries the principal database to determine the corresponding principal, and passes this principal to the OAM for all subsequent authority checks on behalf of the application. These authority checks include checking whether the principal has the authority to connect to the queue manager. If the user ID has no entry in the principal database, the queue manager assigns the principal nobody to the user ID.

When an application sends a message, the queue manager sets the UserIdentifier field in the message descriptor to the principal associated with the application. The principal therefore travels with the message as the means of identifying the user who sent the message.

The UserIdentifier field is one of the message context fields in a message descriptor. Normally, the queue manager sets these fields in a message, but an application can set the fields provided it has the authority to do so. For more information about the message context fields, see “Context authority” on page 148.

Principals and groups

Users can belong to groups. You can grant access to a particular resource to groups, rather than to individual users, in order to reduce the amount of administration required. For example, you might create 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 and to the principal database. For information about creating a group and managing the group subsequently, see “Creating and managing groups” on page 148.

A user 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 user’s group membership are not recognized until the queue manager is restarted or you issue the MQSC command REFRESH SECURITY (or the PCF equivalent).

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 corresponding principal, 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.

All users, including users who have been assigned the principal nobody, are considered to be members of the group called nobody. By default, this group has no authorities. However, you can grant authorities to the group nobody in order to provide access to WebSphere MQ resources to those users who cannot access the resources by any other means. These users comprise those who belong to groups that do not have authority to access the resources, and those who have been assigned the principal nobody.

**Alternate user authority**

One user 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 useful where a server application receives a request from another application and wants to ensure that the other application has the required authority for the request.

For example, assume that a server application running under user ID PAYSERV retrieves a request message from a queue that was put in the queue by an application running under user ID USER1. When the server application gets the request message, it processes the request and puts the reply into the reply-to queue specified by the request message. Instead of using the principal associated with its own user ID (PAYSERV) to access the reply-to queue, the server application can use the principal associated with user ID USER1. The server application sets this alternative principal in the AlternateUserId field of the object descriptor when attempting to open the reply-to queue. The server application succeeds in opening the reply-to only if both the following conditions are met:

- The principal associated with the server application has alternate user authority.
- The alternative principal has the authority to open the reply-to queue for the types of operation requested.

Typically, the server application derives the alternative principal from the contents of the UserId field in the message descriptor of the original request message. For more information about the UserId field, see “Context authority” on page 148.
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 information when either an MQPUT or MQPUT1 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 applications to check the identity of the requester, testing whether the message came from an application running under an authorized user ID.

An application must have authority to 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.

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 user’s group membership are not recognized until the queue manager is restarted or you issue the MQSC command REFRESH SECURITY (or the PCF equivalent).

On NonStop OS, use the Safeguard command interpreter, SAFECOM, to manage groups and users. For detailed information about how to use SAFECOM commands, see the Safeguard Administrator’s Manual.

Creating a group

Use the ADD GROUP command to create a user group. You must create a group before you create any user ID that has this group as its primary group. If you are going to use a group as a primary group, the name of the group must be in upper case and have a maximum of 8 characters. If you are not going to use a group as a primary group, the name can be in mixed case and up to 32 characters in length.

You must assign a unique number within the system to each group that you create. You can determine which group numbers are already in use by using the INFO GROUP command. If you are going to use a group as a primary group, its group number must be in the range 1 to 254 (0 and 255 are reserved group numbers). If you are not going to use a group as a primary group, its group number can be any number in the range 1 to 32767, but excluding 255.

You can grant WebSphere MQ authorities to any valid group. The group does not have to be a primary group.
Using the Giving, Removing, Displaying, Adding, OAM

As an example, the command:

```
ADD GROUP testgrp, NUMBER 762
```

creates a group called testgrp with group number 762. You cannot use this group as a primary group.

**Adding a user to a group**

Use the ALTER GROUP command to add a user to a group. For example, the command:

```
ALTER GROUP mqbrkrs, MEMBER MQM.MANAGER
```

adds the user ID MQM.MANAGER to a group called mqbrkrs.

You can add Safeguard aliases to a group, as well as user IDs.

**Displaying who is in a group**

Use the INFO GROUP command to display the members of a group, or use the INFO USER or INFO ALIAS commands to see which groups a user ID or Safeguard alias belongs to. For example, the command:

```
INFO GROUP mqbrkrs, DETAIL
```

displays a list of the members of the group called mqbrkrs.

**Removing a user from a group**

Use the ALTER GROUP command to remove a user from a group. For example, the command:

```
ALTER GROUP mqbrkrs, MEMBER - MQM.MANAGER
```

removes the user ID MQM.MANAGER from the group called mqbrkrs.

**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 143. Users who 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` command to give users, and groups of users, access to WebSphere MQ objects. For a full definition of the command and its syntax, see "setmqaut (grant or revoke authority)" on page 285. The queue manager must be running to use this command. When you have changed access for a principal or a group, the changes are reflected immediately by the OAM.

To grant users authorities to access objects, or revoke authorities previously granted, you need to specify the following information:

- 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.
- A profile, which is either the full name of an object or a generic name that contains wildcard characters. You can use a generic profile to identify more than
Administration authority

one object in a single `setmqaut` command. For a detailed description of generic profiles, and the use of wildcard characters within them, see the "Using OAM generic profiles" on page 151.

- The type of the objects. This is required to identify the objects uniquely.
- One or more principals and group names for which you are granting or revoking authorities. Each principal must have an entry in the principal database of the queue manager. You cannot specify the corresponding NonStop OS user IDs.
- A list of authorities. Each item in the list specifies a type of access that is to be granted to the users or revoked from them. Each authority in the list is represented by a keyword, which is prefixed by a plus sign (+) or a minus sign (−). A plus sign is used to grant an authority, and a minus sign to revoke an authority. There must be no blanks between a plus or minus sign and the keyword.

You can specify any number of authorities in a single command. For example, the following list grants authority to put messages in a queue and to browse messages, but revokes the authority to get messages:

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

Examples of using the command

The following example shows you how to use the `setmqaut` command to grant and revoke authorities for a single object:

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

In this example:

- `sатурн.queue.manager` is the queue manager name.
- `queue` is the object type.
- `RED.LOCAL.QUEUE` is the name of the object.
- `groupa` is the name of the group whose authorities are to change.
- `+browse -get +put` is the authority list for the specified queue:
  - `+browse` grants authority to browse messages in the queue (that is, to issue MQGET calls with the browse option).
  - `-get` revokes authority to get messages from the queue.
  - `+put` grants authority to put messages in the queue.

The following command revokes from principal `fvuser`, and groups `groupa` and `groupb`, the authority to put messages on the queue called `MyQueue`. The authority is actually revoked from the primary group of the NonStop OS user ID to which the principal `fvuser` maps. This means that the authority is effectively revoked from all members of that group.

```
setmqaut -m saturn.queue.manager -t queue -n MyQueue -p fvuser
       -g groupa -g groupb -p
```

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”
- “Profile priorities”
- “Dumping profile settings” on page 152

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.DE.GJKL.

  **      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: If you enter a control command containing a profile name at an OSS shell command prompt, any wildcard characters in the profile name are intercepted by the OSS shell and interpreted in a different way. To prevent these characters from being intercepted by the OSS shell, precede each occurrence of these characters by a backslash (\), which acts as an escape character, or enclose any profile name containing these characters within single 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 entered the following commands at an OSS shell command prompt:
The following commands, entered at a TACL command prompt, perform the same function:

```
setmqaut -n 'AB.*' -t q +put -p fred
setmqaut -n 'AB.C*' -t q +get -p fred
```

The first command grants put authority to all queues whose names match the profile AB.*. The second command grants get authority to all queues whose names match the profile AB.C*.

Suppose that you now create a queue called AB.CD. According to the rules for wildcard matching, either `setmqaut` command could apply to that queue. So, does the queue 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:

1. ?
2. *
3. **

**Dumping profile settings**

The `dmpmqaut` command enables you to dump the current authorizations associated with a specified profile.

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 user group MQIB:

   ```
   dmpmqaut -m SAMPLE_QMGR -n A.B.C -t q -g MQIB
   ```

   The resulting dump looks something like this:

   profile:  A.B.C
   object type: queue
   entity: MQIB
   entity type: group
   authority: get browse put inq

2. This example dumps all authority records with a profile that matches queue A.B.C:

   ```
   dmpmqaut -m SAMPLE_QMGR -n A.B.C -t q
   ```

   The resulting dump looks something like this:

   profile:  A.B.C
   object type: queue
   entity: MQM
   entity type: group
   authority: amqi dlt chg dsp clr
   - - - - - - - - - - -
   profile:  A.B.C
Administration authority

```plaintext
object type: queue
entity: MQIB
entity type: group
authority: get browse put inq
```

For detailed information about the command, see “dspmqaut (dump authority)” on page 257.

### Displaying access settings

Use the `dspmqaut` command to display the authorities that a specific principal or group has for a particular object. When you change access for a principal or a group using the `setmqaut` command, the changes are reflected immediately by the OAM. The `dspmqaut` command displays authorities as keywords, which are the same keywords that are used to represent authorities on a `setmqaut` command. You can display the authorities for only one principal or group at a time. See “dspmqaut (display authority)” on page 260 for a formal specification of this command.

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

```
dspmqaut -m QueueMan1 -t process -n Annuities -g GpAdmin
```

### Changing and preventing access to a WebSphere MQ object

To change the level of access that a user or group has to a WebSphere MQ object, use the `setmqaut` command. To prevent a particular user from accessing an object, where the user is a member of a group that has access, remove the user from the group as described in “Creating and managing groups” on page 148. This change to the membership of the group is not recognized until the queue manager is restarted or you issue the MQSC command REFRESH SECURITY (or the PCF equivalent).

### Preventing access control checks

If you decide that you want to prevent access control checks (in a test environment, for example), you can disable the OAM in one of two ways:

- Set the operating system environment variable MQSNOAUT to the value yes, before you create a queue manager:
  ```
  export MQSNOAUT=yes
  ```
  This method permanently disables the OAM for the queue manager.

- Edit the queue manager configuration file, qm.ini, to remove the ServiceComponent stanza for the OAM.

### Channel security

Message channel agents (MCAs) are WebSphere MQ applications that use the MQI to access WebSphere MQ resources. The principals associated with MCAs therefore need authority to access these resources.

An MCA must be able to connect to a queue manager and open the dead letter queue. If it is a sending MCA, it must be able to open the transmission queue for the channel. If it is a receiving MCA, it must be able to open destination queues.
Administration authority

and set context information in the messages that it puts in those queues. Channel exit programs called by the MCA might also need to access queue manager resources.

If the PUTAUT parameter is set to CTX in the channel definition at the receiving end of a channel, the principal in the UserIdentifier field in the message descriptor of each incoming message needs authority to open the destination queue for the message. In addition, the principal associated with the receiving MCA needs alternate user authority to open the destination queue using the authority of a different principal.

On an MQI channel, the principal associated with the server connection MCA needs authority to issue MQI calls on behalf of the client application.

The principal that is used for authority checks depends on whether the MCA is connecting to a queue manager or accessing queue manager resources after it has connected to the queue manager:

The principal for connecting to a queue manager
The principal whose authority is checked when an MCA connects to a queue manager is the one associated with the user ID under which the MCA is running. This user ID is known as the default user ID of the MCA, and is user ID of the user who ran the installation script, instmqm.

After the MCA has connected to the queue manager, it accesses certain queue manager resources as part of its initialization processing. The principal associated with the default user ID of the MCA is also used for the authority checks when it opens these resources.

Because the primary group of the user who ran the installation script is MQM, the principal associated with the default user ID automatically has all the authority it needs to connect to the queue manager and access its resources.

The principal for subsequent authority checks
After a sending MCA has connected to a queue manager, the principal whose authority is checked when the MCA accesses queue manager resources subsequently is always the principal associated with the default user ID.

For a receiving or server connection MCA, the principal whose authority is checked when the MCA accesses queue manager resources subsequently might be different to the one that was checked when the MCA connected to the queue manager. Here are some examples of the other principals that might be used depending on how you configure WebSphere MQ:

• The principal specified by the MCAUSER parameter in the channel definition.
• The principal set by a security exit called by the MCA.
• The principal in the UserIdentifier field in the message descriptor of each incoming message. This principal needs authority to open the destination queue for the message. This applies only to a receiving MCA for which the PUTAUT parameter is set to CTX in the channel definition at the receiving end of the channel.
• The principal that is received from the client system when a WebSphere MQ client application issues an MQCONN call. This applies only to a server connection MCA.
If you configure WebSphere MQ to use different principals for these authority checks, the simplest solution is to ensure that the corresponding user IDs are members of the MQM group. However, this solution might not be appropriate in all cases. For example, if the PUTAUT parameter is set to CTX in the channel definition at the receiving end of a message channel, you might consider it more secure for the principal in each incoming message to have only sufficient authority to open the destination queue for the message.

If you do not configure WebSphere MQ to use different principals for these authority checks, the principal associated with the default user ID is used instead.

For more information about configuring security on message channels, see WebSphere MQ Intercommunication. For more information about configuring security on MQI channels, see WebSphere MQ Clients.

Operating on channels, channel initiators, and listeners

Channels, channel initiators, and listeners are not WebSphere MQ objects; access to them is not controlled by the OAM. WebSphere MQ does not allow a user or application to operate on these objects, unless the user ID is a member of the MQM group. To use any of the PCF commands listed below, the user ID associated with the principal that is specified in the UserIdentifier field in the message descriptor of the PCF command message must be a member of the MQM group on the system on which the target queue manager runs:

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

The same rule applies to an Escape PCF command message encapsulating an equivalent MQSC command.

Transmission queues

A queue manager automatically puts messages destined for a remote queue manager on a transmission queue. No special authority is required to do this. However, an application requires special authority to put a message directly on a transmission queue. For more information about this special authority, see Table 10 on page 158 and Table 11 on page 158.

Channel exit programs

Channel exit programs are programs that are called at defined places in the processing sequence of an MCA. Users and vendors can write their own channel exit programs. Some are supplied by IBM.
There are several types of channel exit program, but only four have a role in providing channel security:

- Security exit
- Message exit
- Send exit
- Receive exit

For more information about channel exit programs and their role in providing channel security, see WebSphere MQ Security and WebSphere MQ Intercommunication.

Protecting channels with SSL

The Secure Sockets Layer (SSL) protocol provides channel security, with protection against eavesdropping, tampering, and impersonation. Using the WebSphere MQ support for SSL, you can specify in the channel definition that a channel must use SSL. You can also specify various configuration information, such as the encryption algorithm you want to use.

The WebSphere MQ SSL support includes the following parameter on the ALTER QMGR command:

SSLKEYR
The fully qualified path name of the directory in the OSS file system that contains the following three files:

- The certificate store
- The pass phrase stash file
- The certificate revocation list file (optional)

The default value is `var_installation_path/var/mqm/qmgrs/qmname/ssl`, where `qmname` is the name of the queue manager’s directory.

The WebSphere MQ SSL support includes the following parameters on the DEFINE CHANNEL or ALTER CHANNEL command:

SSLCIPH
The CipherSpec to be used on the channel, for example NULL_MD5 or RC4_MD5_US. The same CipherSpec must be specified at both ends of the channel.

If you do not specify a CipherSpec at one end or both ends of a channel, the channel does not attempt to use SSL.

SSLCAUTH
Whether the SSL server must authenticate the SSL client, or whether authenticating the SSL client is optional. By default, the SSL server must authenticate the SSL client.

SSLPEER
A pattern with which WebSphere MQ compares the distinguished name in a digital certificate received from the peer queue manager or client at the other end of the channel. If the distinguished name does not match the pattern, the channel does not start. By default, the distinguished name is not checked when the channel starts.

For more information about these parameters, see the `WebSphere MQ Script (MQSC) Command Reference`. For information about the equivalent parameters on PCF commands, see WebSphere MQ Programmable Command Formats and Administration Interface.
Administration authority

For an overview of channel security using SSL, see WebSphere MQ Security.

For detailed information about the SSL support provided by WebSphere MQ for HP NonStop Server, see Chapter 11, “Working with the WebSphere MQ Secure Sockets Layer (SSL) support,” on page 163.

How authorizations work

The authorization specification tables starting on page 158 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, or namelist.

**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 +seta11, 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 principal associated with the user ID under which the application 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 appears 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 9 on page 158, Table 10 on page 158, Table 11 on page 158, and Table 12 on page 158 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.
Authorization specification tables

**Note:** Namelists are not included 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 9. Security authorization needed for MQCONN calls

<table>
<thead>
<tr>
<th>Authorization required for</th>
<th>Queue object</th>
<th>Process object</th>
<th>Queue manager object</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQCONN</td>
<td>Not applicable</td>
<td>Not applicable</td>
<td>MQZAO_CONNECT</td>
</tr>
</tbody>
</table>

Table 10. Security authorization needed for MQOPEN calls

<table>
<thead>
<tr>
<th>Authorization required for</th>
<th>Queue object</th>
<th>Process object</th>
<th>Queue manager object</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQOO_INQUIRE</td>
<td>MQZAO_INQUIRE</td>
<td>MQZAO_INQUIRE</td>
<td>MQZAO_INQUIRE</td>
</tr>
<tr>
<td>MQOO_BROWSE</td>
<td>MQZAO_BROWSE</td>
<td>Not applicable</td>
<td>No check</td>
</tr>
<tr>
<td>MQOO_INPUT_*</td>
<td>MQZAO_INPUT</td>
<td>Not applicable</td>
<td>No check</td>
</tr>
<tr>
<td>MQOO_SAVE_ALL_CONTEXT</td>
<td>MQZAO_INPUT</td>
<td>Not applicable</td>
<td>Not applicable</td>
</tr>
<tr>
<td>MQOO_OUTPUT (Normal queue)</td>
<td>MQZAO_OUTPUT</td>
<td>Not applicable</td>
<td>Not applicable</td>
</tr>
<tr>
<td>MQOO_PASS_IDENTITY_CONTEXT</td>
<td>MQZAO_PASS_IDENTITY_CONTEXT</td>
<td>Not applicable</td>
<td>No check</td>
</tr>
<tr>
<td>MQOO_PASS_ALL_CONTEXT</td>
<td>MQZAO_PASS_ALL_CONTEXT</td>
<td>Not applicable</td>
<td>No check</td>
</tr>
<tr>
<td>MQOO_SET_IDENTITY_CONTEXT</td>
<td>MQZAO_SET_IDENTITY_CONTEXT</td>
<td>Not applicable</td>
<td>MQZAO_SET_IDENTITY_CONTEXT</td>
</tr>
<tr>
<td>MQOO_SET_ALL_CONTEXT</td>
<td>MQZAO_SET_ALL_CONTEXT</td>
<td>Not applicable</td>
<td>MQZAO_SET_ALL_CONTEXT</td>
</tr>
<tr>
<td>MQOO_OUTPUT (Transmission queue)</td>
<td>MQZAO_SET_ALL_CONTEXT</td>
<td>Not applicable</td>
<td>MQZAO_SET_ALL_CONTEXT</td>
</tr>
<tr>
<td>MQOO_SET</td>
<td>MQZAO_SET</td>
<td>Not applicable</td>
<td>No check</td>
</tr>
<tr>
<td>MQOO_ALTERNATE_USER_AUTHORITY</td>
<td>Not applicable</td>
<td>Not applicable</td>
<td>MQZAO_ALTERNATE_USER_AUTHORITY</td>
</tr>
</tbody>
</table>

Table 11. Security authorization needed for MQPUT1 calls

<table>
<thead>
<tr>
<th>Authorization required for</th>
<th>Queue object</th>
<th>Process object</th>
<th>Queue manager object</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQPMO_PASS_IDENTITY_CONTEXT</td>
<td>MQZAO_PASS_IDENTITY_CONTEXT</td>
<td>Not applicable</td>
<td>No check</td>
</tr>
<tr>
<td>MQPMO_PASS_ALL_CONTEXT</td>
<td>MQZAO_PASS_ALL_CONTEXT</td>
<td>Not applicable</td>
<td>No check</td>
</tr>
<tr>
<td>MQPMO_SET_IDENTITY_CONTEXT</td>
<td>MQZAO_SET_IDENTITY_CONTEXT</td>
<td>Not applicable</td>
<td>MQZAO_SET_IDENTITY_CONTEXT</td>
</tr>
<tr>
<td>MQPMO_SET_ALL_CONTEXT</td>
<td>MQZAO_SET_ALL_CONTEXT</td>
<td>Not applicable</td>
<td>MQZAO_SET_ALL_CONTEXT</td>
</tr>
<tr>
<td>(Transmission queue)</td>
<td>MQZAO_SET_ALL_CONTEXT</td>
<td>Not applicable</td>
<td>MQZAO_SET_ALL_CONTEXT</td>
</tr>
<tr>
<td>MQPMO_ALTERNATE_USER_AUTHORITY</td>
<td>Not applicable</td>
<td>Not applicable</td>
<td>MQZAO_ALTERNATE_USER_AUTHORITY</td>
</tr>
</tbody>
</table>

Table 12. Security authorization needed for MQCLOSE calls

<table>
<thead>
<tr>
<th>Authorization required for</th>
<th>Queue object</th>
<th>Process object</th>
<th>Queue manager object</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Authorization specification tables

<table>
<thead>
<tr>
<th>MQCO_DELETE</th>
<th>MQZAO_DELETE</th>
<th>Not applicable</th>
<th>Not applicable</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQCO_DELETE_PURGE</td>
<td>MQZAO_DELETE</td>
<td>Not applicable</td>
<td>Not applicable</td>
</tr>
</tbody>
</table>

### 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 principal associated with the user ID that is 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 5).
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 alternative principal for the specific named object authority, and the current application authority for the MQZAO_ALTERNATE_USER_IDENTIFIER check.
10. This authorization allows any *AlternateUserld* 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 alternative principal 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.
Authorization specification tables

Authorizations for MQSC commands in escape PCFs

Table 13 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 principal associated with 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

Table 13. MQSC commands and security authorization needed

<table>
<thead>
<tr>
<th>Authorization required for:</th>
<th>Queue object</th>
<th>Process object</th>
<th>Queue manager object</th>
<th>Namelist object</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALTER object</td>
<td>MQZAO_CHANGE</td>
<td>MQZAO_CHANGE</td>
<td>MQZAO_CHANGE</td>
<td>MQZAO_CHANGE</td>
</tr>
<tr>
<td>CLEAR QLOCAL</td>
<td>MQZAO_CLEAR</td>
<td>Not applicable</td>
<td>Not applicable</td>
<td>Not applicable</td>
</tr>
<tr>
<td>DEFINE object NOREPLACE</td>
<td>MQZAO_CREATE</td>
<td>MQZAO_CREATE</td>
<td>Not applicable</td>
<td>MQZAO_CREATE</td>
</tr>
<tr>
<td>DEFINE object REPLACE</td>
<td>MQZAO_CHANGE</td>
<td>MQZAO_CHANGE</td>
<td>Not applicable</td>
<td>MQZAO_CHANGE</td>
</tr>
<tr>
<td>DELETE object</td>
<td>MQZAO_DELETE</td>
<td>MQZAO_DELETE</td>
<td>Not applicable</td>
<td>MQZAO_DELETE</td>
</tr>
<tr>
<td>DISPLAY object</td>
<td>MQZAO_DISPLAY</td>
<td>MQZAO_DISPLAY</td>
<td>MQZAO_DISPLAY</td>
<td>MQZAO_DISPLAY</td>
</tr>
</tbody>
</table>

Notes for Table 13

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

Table 14 on page 161 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 principal associated with 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 Table 14 on page 161 that are relevant to the object type, except MQZAO_CREATE, which is not specific to a particular object or object type.
Table 14. PCF commands and security authorization needed

<table>
<thead>
<tr>
<th>Authorization required for:</th>
<th>Queue object</th>
<th>Process object</th>
<th>Queue manager object</th>
<th>Namelist object</th>
</tr>
</thead>
<tbody>
<tr>
<td>Change object</td>
<td>MQZAO_CHANGE</td>
<td>MQZAO_CHANGE</td>
<td>MQZAO_CHANGE</td>
<td>MQZAO_CHANGE</td>
</tr>
<tr>
<td>Clear Queue</td>
<td>MQZAO_CLEAR</td>
<td>Not applicable</td>
<td>Not applicable</td>
<td>Not applicable</td>
</tr>
<tr>
<td>Copy object (without replace)</td>
<td>MQZAO_CREATE</td>
<td>MQZAO_CREATE</td>
<td>Not applicable</td>
<td>MQZAO_CREATE</td>
</tr>
<tr>
<td>Copy object (with replace)</td>
<td>MQZAO_CHANGE</td>
<td>MQZAO_CHANGE</td>
<td>Not applicable</td>
<td>MQZAO_CHANGE</td>
</tr>
<tr>
<td>Create object (without replace)</td>
<td>MQZAO_CREATE</td>
<td>MQZAO_CREATE</td>
<td>Not applicable</td>
<td>MQZAO_CHANGE</td>
</tr>
<tr>
<td>Create object (with replace)</td>
<td>MQZAO_CHANGE</td>
<td>MQZAO_CHANGE</td>
<td>Not applicable</td>
<td>MQZAO_CHANGE</td>
</tr>
<tr>
<td>Delete object</td>
<td>MQZAO_DELETE</td>
<td>MQZAO_DELETE</td>
<td>Not applicable</td>
<td>MQZAO_DELETE</td>
</tr>
<tr>
<td>Inquire object</td>
<td>MQZAO_DISPLAY</td>
<td>MQZAO_DISPLAY</td>
<td>MQZAO_DISPLAY</td>
<td>MQZAO_DISPLAY</td>
</tr>
<tr>
<td>Inquire object names</td>
<td>No check</td>
<td>No check</td>
<td>No check</td>
<td>No check</td>
</tr>
<tr>
<td>Reset queue statistics</td>
<td>MQZAO_DISPLAY and MQZAO_CHANGE</td>
<td>Not applicable</td>
<td>Not applicable</td>
<td>Not applicable</td>
</tr>
</tbody>
</table>

Notes for Table 14

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.
Authorization specification tables
Chapter 11. Working with the WebSphere MQ Secure Sockets Layer (SSL) support

This chapter describes the Secure Sockets Layer (SSL) support supplied with WebSphere MQ for HP NonStop Server. The WebSphere MQ SSL support provides the following security services on message and MQI channels:

- Authentication of the SSL server and, optionally, authentication of the SSL client
- Encryption and decryption of the data flowing across a channel
- Integrity checks on the data flowing across a channel

The SSL support supplied with WebSphere MQ for HP NonStop Server comprises the following components:

- The OpenSSL library and openssl command. These are supplied as object code only.
- The entropy daemon, which is the source of random data for OpenSSL.
- Sample shell scripts and MQSC command files that illustrate how to configure SSL channels.

WebSphere MQ supports Version 3.0 of the SSL protocol.

The chapter contains the following sections:

- “Introduction to OpenSSL”
- “Where the files containing the WebSphere MQ SSL support code are installed” on page 164
- “The entropy daemon” on page 165
- “Preparing to use the WebSphere MQ SSL support” on page 166
- “Working with keys and digital certificates” on page 167
- “A sample configuration for testing” on page 172

To understand this chapter, you need a good knowledge of the concepts and terminology associated with SSL, cryptography, and a public key infrastructure (PKI). You also need an overall understanding of how SSL provides security services on WebSphere MQ channels. If you need an introduction to any of these topics, see WebSphere MQ Security.

Introduction to OpenSSL

The OpenSSL toolkit is an open source implementation of the Secure Sockets Layer (SSL) and Transport Layer Security (TLS) protocols for secure communications over a network. The toolkit has been developed by the OpenSSL Project. For information about the OpenSSL Project, see www.openssl.org.

WebSphere MQ for HP NonStop Server contains modified versions of the OpenSSL library and openssl command. The library and command have been ported from the OpenSSL toolkit 0.9.7d and are supplied as object code only. No source code is provided. Only these modified versions of the OpenSSL library and openssl command are supported.
Working with the WebSphere MQ SSL support

You can install WebSphere MQ for HP NonStop Server with or without the SSL support. If you choose to install the SSL support, the OpenSSL library is integrated with the installed WebSphere MQ libraries that use SSL function. If you choose not to install the SSL support, versions of the WebSphere MQ libraries that do not contain the OpenSSL library are installed instead.

Using the OSS command `openssl`, you can create and manage keys and digital certificates using a variety of common data formats, and perform simple certification authority (CA) tasks.

The default format for key and certificate data processed by OpenSSL is the Privacy Enhanced Mail (PEM) format. Data in PEM format is base64 encoded ASCII data. The data can therefore be transferred using text based systems such as e-mail and can be cut and pasted using text editors and Web browsers. PEM is an Internet standard for text based cryptographic exchanges and is specified in Internet RFCs 1421, 1422, 1423, and 1424. WebSphere MQ assumes that a file with extension .pem contains data in PEM format. A file in PEM format can contain multiple certificates and other encoded objects, and can include comments.

The WebSphere MQ SSL support on other platforms might require key and certificate data in files to be encoded using Distinguished Encoding Rules (DER). DER is a set of encoding rules for using the ASN.1 notation in secure communications. Data encoded using DER is binary data, and the format of key and certificate data encoded using DER is also known as PKCS#12 or PFX. A file containing this data commonly has an extension of .p12 or .pfx. The `openssl` command can convert between PEM and PKCS#12 format.

The `openssl` command has many options, but this chapter describes only the options you need in order to use the SSL support in WebSphere MQ. For a detailed specification of the `openssl` command and its options, see www.openssl.org/docs/apps/openssl.html.

OpenSSL requires a source of random data. When using the SSL support supplied with WebSphere MQ for HP NonStop Server, a server process called the entropy daemon provides this source of random data.

Where the files containing the WebSphere MQ SSL support code are installed

When you transfer the files from the delivery medium to NonStop OS during installation, the files containing the WebSphere MQ SSL support code are stored in the directory `/usr/ibm/wmq/GA/opt/mqm/ssl`. Subsequently, when you run the installation script, instmqm, these files are copied to the directories from where they are run.

If you choose to install the WebSphere MQ SSL support, the OpenSSL library is integrated with the following TCP/IP communications DLLs, which are installed in the directory `opt_installation_path/opt/mqm/lib`:

- `amqctca`
  - Used by an MCA that runs as a process

- `amqctca_r`
  - Used by an MCA that runs as a thread within a process
Working with the WebSphere MQ SSL support

A separate OpenSSL library is not supplied. The following DLLs, which do not contain the OpenSSL library, are also installed in the directory
opt_installation_path/opt/mqm/lib:

amqcctca_nossll
Used by an MCA that runs as a process

amqcctca_r_nossll
Used by an MCA that runs as a thread within a process

If you choose not to install the WebSphere MQ SSL support, versions of the DLLs amqcctca and amqcctca_r that do not contain the OpenSSL library are installed in the directory opt_installation_path/opt/mqm/lib instead, and the DLLs amqcctca_nossll and amqcctca_r_nossll are not installed.

The openssl command is installed in the directory opt_installation_path/opt/mqm/bin, and the sample shell scripts and MQSC command files are installed in the directory opt_installation_path/opt/mqm/samp/ssl.

The entropy daemon is installed in the directory opt_installation_path/opt/mqm/bin, but you might want to copy it to another directory on your system. For information about how to configure and run the entropy daemon, see "Configuring and running the entropy daemon."

The entropy daemon

The entropy daemon is the source of random data for OpenSSL. The random data is provided through a socket in the OSS file system called /etc/egd-pool. OpenSSL uses the random data as unpredictable starting points for generating symmetric and asymmetric keys. Keys generated in this way are more difficult to break. The entropy daemon is not a component of the OpenSSL toolkit.

Only one entropy daemon can run in a NonStop OS system at any one time. The entropy daemon must run continuously whenever SSL channels are used, or whenever you use the openssl command.

Configuring and running the entropy daemon

The entropy daemon logs operator messages to EMS using the OSS syslog emulation facility. After logging a successful startup and initialization, the entropy daemon produces no further logging during normal operation. The entropy daemon requires no operator action apart from monitoring it occasionally to ensure that, if a failure occurs, it is restarted promptly.

Run the entropy daemon as a persistent generic process that is managed by the Kernel subsystem. You can use the persistence manager of the Kernel subsystem to create, configure, and manage daemon processes, including being able to restart a daemon process automatically in case of failure. For detailed information about how to use the Kernel subsystem, see the SCF Reference Manual for the Kernel Subsystem.

The executable file for the entropy daemon is in the directory opt_installation_path/opt/mqm/bin. The name of the file is amqjndm0. Copy this file to a suitable system-wide OSS directory from where it can be run independently of any WebSphere MQ installation and its SSL support. A typical suitable directory is /usr/local/bin.
The following sample shell script contains the SCF commands to configure and start the entropy daemon for a NonStop OS system. This script is supplied in the file opt_installation_path/opt/mqm/samp/ssl/etpdcfg.

CONFIRM ON
ALLOW 10 ERRORS
ABORT PROCESS $ZZKRN.#SSL-ENTROPY-DAEMON
DELETE PROCESS $ZZKRN.#SSL-ENTROPY-DAEMON
ADD PROCESS $ZZKRN.#SSL-ENTROPY-DAEMON, &
  NAME $ETPD1, &
  AUTORESTART 10, &
  PRIMARYCPU 2, &
  STARTMODE MANUAL, &
  USERID MQM.MANAGER, &
  PROGRAM $SYSTEM.SYSTEM.OSH, &
  ASSOCPROC $ETPD2, &
  STARTUPMSG "-ls -name /G/ETPD2 -osstty &
    -p /usr/local/bin/amqjkdm0"

START PROCESS $ZZKRN.#SSL-ENTROPY-DAEMON

This script causes the entropy daemon to run under the user ID MQM.MANAGER with the process name $ETPD2. The script also causes the Kernel subsystem to monitor the entropy daemon and its parent osh process, with the process name $ETPD1, and restart both processes in the event of a failure.

Add this script, or a similar script, to your system startup scripts so that it runs after OSS has started. In this way, the entropy demon restarts automatically whenever NonStop OS restarts.

Because the entropy daemon is a critical resource for the WebSphere MQ SSL support, restrict access to the daemon by allowing only certain specified users to configure, run, and manage the daemon.

### Stopping the entropy daemon

Use the SCF command ABORT PROCESS to stop the entropy daemon. For example, the following command stops the entropy daemon and parent osh process that are started by the sample shell script described in Configuring and running the entropy daemon on page 165.

ABORT PROCESS $ZZKRN.#SSL-ENTROPY-DAEMON

### Preparing to use the WebSphere MQ SSL support

Before starting to configure and use the WebSphere MQ SSL support, perform the following tasks:

1. Verify that the WebSphere MQ SSL support is installed.
2. Verify that the entropy daemon is running on your system.
3. Decide how to specify the configuration file for the openssl command.

The following sections tell you how to perform these tasks.

### Verifying that the WebSphere MQ SSL support is installed

You can verify that the WebSphere MQ SSL support is installed in either of the following ways:

- Inspect the response file for your installation and make sure that the InstallComponent entry in the Install stanza specifies SSL or ALL.
- Make sure that your installation contains the following files:

  opt_installation_path/opt/mqm/lib/amqcctca_nossl
Verifying that the entropy daemon is running

You can verify that the entropy daemon is running in any of the following ways:

- If you have configured the entropy daemon to be managed by the Kernel subsystem, use the SCF command STATUS PROCESS $ZZKRN.#entropy_daemon_name to determine whether the entropy daemon and its parent osh process are running.
- If you know the process name that you assigned to the entropy daemon, use the TACL command STATUS process_name.
- If you did not configure the entropy daemon with a fixed process name, use the OSS command ps to look for a process whose executable file is called amqjdkm0, as in the following example:

  ps –A | grep amqjdkm0

If the entropy daemon is not running, use the information in “Configuring and running the entropy daemon” on page 165 to start the entropy daemon.

Deciding how to specify the configuration file for the openssl req command

The openssl command is the only component of OpenSSL that you need to consider when establishing the environment of an OSS shell. It is particularly important to understand this if other implementations of OpenSSL might be present on your system.

The openssl req command uses a configuration file. You can specify the name and location of the configuration file in either of the following ways, in order of precedence:

1. By using the command line option –conf file_name
2. By setting the environment variable OPENSSL_CONF to the fully qualified name of the configuration file

If you use neither of these two ways, the openssl req command looks for a configuration file with the name /usr/local/ssl/lib/openssl.cnf.

The sample shell scripts supplied with WebSphere MQ use the first way of specifying the name and location of the configuration file so that there is no ambiguity about where the configuration information comes from.

Working with keys and digital certificates

WebSphere MQ for HP NonStop Server uses only a subset of the function provided by the OpenSSL toolkit, and this section describes only that subset. For a complete description of all the function provided by the OpenSSL toolkit, see www.openssl.org.

Typically, the following keys and digital certificates are required by a queue manager:

- The queue manager’s private key
- The queue manager’s personal certificate
Working with the WebSphere MQ SSL support

- All the CA certificates that are needed to validate the personal certificates received by the queue manager. For each personal certificate that a queue manager receives, the queue manager needs all the CA certificates in the certificate chain that begins at the personal certificate.

The private key and digital certificates for a queue manager are held in the queue manager’s certificate store. For WebSphere MQ for HP NonStop Server, the key and certificate data must be in Privacy Enhanced Mail (PEM) format.

You can use the `openssl req` command to generate keys and certificate requests in PEM format. Alternatively, you can use the facilities provided on other platforms to generate keys and certificate requests in either PEM or PKCS#12 format. If you receive key and certificates in PKCS#12 format, you can use the `openssl pkcs12` command to import them, converting them to PEM format. Conversely, you can also use the `openssl pkcs12` command to export keys and certificates in PEM format, converting them to PKCS#12 format for use on other platforms.

The following tasks are the basic tasks you need to perform in order to manage keys and digital certificates for the WebSphere MQ SSL support:

- Use the `openssl req` command to generate the public and private keys for a queue manager, and a request for a personal certificate. The request is then sent to a CA. For information about how to perform this task, see “Generating public and private keys, and a request for a personal certificate.”

- Use the `openssl pkcs12` command to import a digital certificate in PKCS#12 format, converting it to PEM format. For information about how to perform this task, see “Importing digital certificates” on page 170.

- Prepare the queue manager’s SSL files. For information about how to perform this task, see “Preparing the queue manager’s SSL files” on page 170.

The `openssl` command also contains function that you can use for testing, or for supporting SSL channels that are internal to your organization. You can use the `openssl` command to generate a self-signed personal certificate, which is useful only for testing purposes. To support SSL channels that are internal to your organization, you can use the `openssl` command to provide the basic CA functions for setting up and using a Public Key Infrastructure (PKI). These are some of the tasks that you can perform using the `openssl` command:

- Generate a CA certificate, including a root CA certificate
- Generate a personal certificate
- Export a digital certificate, converting it to PKCS#12 format
- Generate a certificate revocation list (CRL)

This is not an exhaustive list of all the tasks that you might need to perform. The sample shell scripts in the directory `opt_installation_path/opt/mqm/samp/ssl` perform some of these tasks.

Generating public and private keys, and a request for a personal certificate

To generate the public and private keys for a queue manager, and a request for a personal certificate, use the `openssl req` command with the following options:

- `-newkey key_specifier`
- `-out certificate_request_file_name`
- `-keyout private_key_file_name`
Working with the WebSphere MQ SSL support

The `-newkey` option causes the OpenSSL `req` command to generate new public and private keys, and a certificate request. The option also specifies the type of keys required. The command stores the certificate request in the file specified by the `-out` option and the encrypted private key in the file specified by the `-keyout` option. By default, the contents of both files are in PEM format. Never distribute the file containing the private key. Set its file permissions so that unauthorized users cannot access it.

Typically, you might also need to specify the following information on the OpenSSL `req` command:

- The name of the configuration file to be used by the OpenSSL `req` command. This overrides any configuration file specified by the OPENSSL_CONF environment variable, or the default configuration file. Use the `-config` option to specify the name of the configuration file.

For information about the contents of a configuration file, see [www.openssl.org/docs/apps/config.html](http://www.openssl.org/docs/apps/config.html). See also the configuration file format section of the specification of the OpenSSL `req` command at [www.openssl.org/docs/apps/req.html#CONFIGURATION_FILE_FORMAT](http://www.openssl.org/docs/apps/req.html#CONFIGURATION_FILE_FORMAT). The directory `opt_installation_path/opt/mqm/samp/ssl` contains some sample configuration files.

- The message digest algorithm to be used for signing the certificate request. This overrides any message digest algorithm specified in the configuration file. You can specify one of the following options: `md5`, `-sha1`, `-md2`, or `-mdc2`.

- You can use the configuration file to supply certain information for the certificate request, such as the distinguished name of the queue manager. If you do not specify this information in a configuration file, use the `-new` option and the OpenSSL `req` command prompts you to enter the required information.

The OpenSSL `req` command prompts you twice to enter a pass phrase, which the command uses to encrypt the new private key. Choose a pass phrase that is difficult to guess and keep the pass phrase secret. You can specify a pass phrase by using the `-passin` option instead, but doing it this way might compromise the security of the private key because the pass phrase is displayed on the screen.

Here is an example of an OpenSSL `req` command:

```bash
openssl req -newkey rsa:512 -sha1 -keyout ALICE_key.pem \
    -out ALICE_req.pem -config openssl_ALICEreq.cnf
```

This command generates a 512 bit RSA private key and stores the encrypted private key in the file `ALICE_key.pem`. The command also creates a certificate request in the file `ALICE_req.pem`, and signs the request using the SHA1 message digest algorithm. The configuration file `openssl_ALICEreq.cnf` supplies certain information for the certificate request, such as the distinguished name of the queue manager.

To obtain a personal certificate, you can send a certificate request generated by the OpenSSL `req` command to a CA. Alternatively, for testing purposes, or to support SSL channels that are internal to your organization, you can use the OpenSSL `ca` or OpenSSL `x509` command to generate a personal certificate without sending the request to a CA.

When you use the OpenSSL `req` command, you might receive the following error message, or a message similar to it:

unlable to load 'random state'
Working with the WebSphere MQ SSL support

This message means that the entropy daemon is not running on your system.

For a complete description of the `openssl req` command and its options, see www.openssl.org/docs/apps/req.html.

Importing digital certificates

If you receive a digital certificate in PKCS#12 format, you must convert it to PEM format before the WebSphere MQ SSL support can use it.

To create a digital certificate in PEM format from a digital certificate in PKCS#12 format, use the `openssl pkcs12` command. Typically, you enter the command with the following options:

- `-in PKCS#12_file_name`
- `-out PEM_file_name`

The `-in` option specifies the name of the input file containing a digital certificate in PKCS#12 format. The `-out` option specifies the name of the output file containing the digital certificate in PEM format.

Here is an example of an `openssl pkcs12` command:

```bash
openssl pkcs12 -in BOB.p12 -out BOB.pem
```

This command parses a file called BOB.p12 containing a digital certificate in PKCS#12 format, and creates a file called BOB.pem containing the digital certificate in PEM format.

For a complete description of the `openssl pkcs12` command and its options, see www.openssl.org/docs/apps/pkcs12.html.

Preparing the queue manager’s SSL files

The WebSphere MQ SSL support uses a set of files called the queue manager’s SSL files. These files are in a directory specified by the `SSLKeyRepository` attribute of the queue manager object. The default value of the attribute is `var_installation_path/var/mqm/qmgrs/qmname/ssl`, where `qmname` is the name of the queue manager’s directory.

The queue manager’s SSL files are the following files. The names of the files are case sensitive.

- **cert.pem**: The certificate store. This file contains the queue manager’s personal certificate, the encrypted private key of the queue manager, and all the CA certificates that are needed to validate the personal certificates received by the queue manager. The certificates and the encrypted private key are in PEM format. The pass phrase for the encrypted private key is stored in the pass phrase stash file.

- **Stash.sth**: The pass phrase stash file. This file contains the encrypted pass phrase for the private key of the queue manager. The private key itself is stored in the certificate store.

- **crl.pem**: The certificate revocation list file. This file contains the CRLs that the queue manager uses to validate digital certificates. The CRLs are in PEM format.
To use SSL channels, the certificate store and pass phrase stash files must be present in the specified directory, but the certificate revocation list file is optional. If no certificate revocation list file is present, the queue manager does not check whether a digital certificate has been revoked, although the queue manager still performs other checks to validate the certificate.

Set the file permissions of the queue manager’s SSL files so that only members of the MQM group can read the files and write to them.

**Creating the queue manager’s SSL files**

To create a certificate store, concatenate the following files, in the specified order. The contents of all the files must be in PEM format.

1. The file containing the queue manager’s personal certificate.
2. The file containing the encrypted private key of the queue manager.
3. The files containing all the CA certificates that are needed to validate the personal certificates received by the queue manager. For each personal certificate that a queue manager receives, the queue manager needs all the CA certificates in the certificate chain that begins at the personal certificate.

To create a pass phrase stash file, use the `openssl pkcs12` command to export the certificate store. For example, the following command creates the pass phrase stash file, Stash.sth, for the private key in the certificate store called ALICE.pem:

```
openssl pkcs12 -export -in ALICE.pem
 -inkey ALICE_key.pem -out ALICE.p12
 -passin pass:alicekey -passout pass:alicekey
```

The command also converts the contents of the certificate store to PKCS#12 format, and creates a new file called ALICE.p12 containing the results of the conversion. But this is not the primary purpose of the command.

After you have created the pass phrase stash file, copy it to another directory so that any subsequent export operations do not overwrite it.

If you want the queue manager to check whether digital certificates have been revoked, create a certificate revocation list file by concatenating files containing CRLs. The contents of all the files must be in PEM format.

Install the queue manager’s SSL files by copying the certificate store, pass phrase stash file, and the certificate revocation list file to the directory specified by the `SSLKeyRepository` attribute of the queue manager object.

**Finding out where the queue manager’s SSL files are stored**

To find out where the queue manager’s SSL files are stored, issue the MQSC command `DISPLAY QMGR SSLKEYR` to display the value of the `SSLKeyRepository` attribute of the queue manager object.

**Changing the directory where the queue manager’s SSL files are stored**

To change the directory where the queue manager’s SSL files are stored, use the following procedure:

1. Copy the existing SSL files from the old directory to the new directory, or recreate them in the new directory.
2. Use the MQSC command `ALTER QMGR` with the `SSLKEYR` parameter to change the value of the `SSLKeyRepository` attribute of the queue manager.
Working with the WebSphere MQ SSL support

object. For example, the following command specifies that the queue manager’s SSL files are now stored in the directory /home/mqm/ssl:

```
ALTER QMGR SSLKEYR('/home/mqm/ssl')
```

**When changes to the queue manager’s SSL files become effective**

An MCA that runs as a process reads the contents of the queue manager’s SSL files when the MCA is used to implement an SSL channel for the first time. The MCA does not attempt to read this information again subsequently even if the MCA is returned to the pool and reused.

For an MCA process that contains multiple threads, where each thread is an individual MCA, the MCA process reads the contents of the queue manager’s SSL files when the first of its MCAs is used to implement an SSL channel. The MCA process does not attempt to read this information again subsequently even if more of its MCAs are used to implement SSL channels, or an MCA is returned to the pool and reused.

If you change the directory where the queue manager’s SSL files are stored, or if you change the contents of the queue manager’s SSL files, MCA processes that have already read the value of the *SSLkeyRepository* attribute of the queue manager object and the contents the queue manager’s SSL files do not notice these changes. For the changes to become effective immediately, restart the queue manager, and start all the channels again. Alternatively, use the following procedure:

1. In the process management rules configuration file, qmproc.ini, set the maximum reuse count for MCA processes to 1.
   For MCAs that run as processes, the MCA stanza must contain the following entry:
   ```
   MaxAgentUse=1
   ```
   For MCA processes with threads, the MCA stanza must contain the following entry:
   ```
   MaxThreadedAgentUse=1
   ```
   For more information about the contents of a process management rules configuration file, see Chapter 14, “Process management,” on page 197

2. Issue the MQSC command `RESET QMGR TYPE(NSPROC)`, so that the changes to the process management rules configuration file take effect.
3. Stop all the channels and start them again.

**A sample configuration for testing**

The sample configuration described in this section is an example of how to use the SSL support supplied with WebSphere MQ for HP NonStop Server. The configuration does not use personal and CA certificates that are issued and signed by a real CA and is therefore not suitable for use in production. However, the steps in the procedure to build the sample configuration are essentially the same as those needed to build a production configuration.

The sample configuration consists of two queue managers, ALICE and BOB, which belong to an organization called BIGCO and run on the same system. The queue managers are configured so that they can exchange messages over SSL channels.
Working with the WebSphere MQ SSL support

This section first describes the sample shell scripts and MQSC command files that are used by the procedure to build and verify the sample configuration, and then describes the procedure itself. The section concludes by providing some guidance on how to adapt the sample configuration so that one of the queue managers runs on a different system.

The sample shell scripts and MQSC command files

The procedure to build and verify the sample configuration uses the sample shell scripts and MQSC command files in the directory opt_installation_path/opt/mqm/samp/ssl.

The procedure uses the following shell scripts to prepare the keys and digital certificates required by queue managers ALICE and BOB:

create_root_cert.sh

This shell script requests and creates a root CA certificate. The flow of control within the script has the following steps:

1. The openssl req command is used to generate the public and private keys for the root CA, and a request for a root CA certificate. Because the integrity of the root CA certificate is fundamental to all certificates in the chain, 2048 bit RSA keys are generated for the root CA. The -passout option on the command supplies a pass phrase, which the command uses to encrypt the root CA’s private key.

2. The openssl x509 command is used to process the certificate request and create a root CA certificate. The -signkey option on the command supplies the root CA’s private key, which is used to sign the certificate. The certificate that is created is therefore self-signed. The -passin option on the command supplies the pass phrase for the root CA’s private key, which the command uses to decrypt the private key.

3. The root CA certificate and the encrypted private key of the root CA are concatenated to form a single file.

4. The openssl x509 command is used to validate the contents of the file created in the previous step.

create_bigco_cert.sh

This shell script requests and creates a CA certificate for the BIGCO CA. The CA certificate is signed by the root CA. The flow of control within the script has the following steps:

1. The openssl req command is used to generate the public and private keys for the BIGCO CA, and a request for a CA certificate. 1024 bit RSA keys are generated for the BIGCO CA. The -passout option on the command supplies a pass phrase, which the command uses to encrypt the BIGCO CA’s private key.

2. The openssl x509 command is used to process the certificate request and create a CA certificate. The -CAkey option on the command supplies the root CA’s private key, which is used to sign the certificate. The -passin option on the command supplies the pass phrase for the root CA’s private key, which the command uses to decrypt the private key.

3. The CA certificate of the BIGCO CA, the encrypted private key of the BIGCO CA, and the root CA certificate are concatenated to form a single file.

4. The openssl x509 command is used to validate the contents of the file created in the previous step.
Working with the WebSphere MQ SSL support

create_ALICE_cert.sh and create_BOB_cert.sh

The shell script create_ALICE_cert.sh requests and creates a personal certificate for ALICE. The shell script create_BOB_cert.sh performs the same function for BOB. Both personal certificates are signed by the BIGCO CA, and the basic logic of each script is the same. The flow of control within each script has the following steps:

1. The openssl req command is used to generate the public and private keys for the queue manager, and a request for a personal certificate. 512 bit RSA keys are generated for the queue manager. The -passout option on the command supplies a pass phrase, which the command uses to encrypt the private key of the queue manager.

2. The openssl x509 command is used to process the certificate request and create a personal certificate. The -CAkey option on the command supplies the BIGCO CA’s private key, which is used to sign the certificate. The -passin option on the command supplies the pass phrase for the BIGCO CA’s private key, which the command uses to decrypt the private key.

3. The personal certificate of the queue manager, the encrypted private key of the queue manager, the CA certificate of the BIGCO CA, and the root CA certificate are concatenated to form a single file. For ALICE, this file is called ALICE.pem and is the certificate store for ALICE. For BOB, the file is called BOB.pem.

4. The openssl x509 command is used to validate the contents of the file created in the previous step.

exportcerts.sh

This shell script creates the pass phrase stash files for ALICE and BOB. The flow of control within the script has the following steps:

1. The openssl pkcs12 command is used to export the file BOB.pem, which is created by the shell script create_BOB_cert.sh. The command creates a pass phrase stash file for BOB called Stash.sth.

   The command also converts the contents of BOB.pem to PKCS#12 format and stores the converted data in a file called BOB.p12. But this is not the primary purpose of the command.

   The -passin option on the command supplies the pass phrase for the BOB’s private key, which the command uses to decrypt the private key stored in PEM format in BOB.pem. The -passout option on the command supplies a second pass phrase for BOB’s private key, which the command uses to encrypt the private key stored in PKCS#12 format in BOB.p12. Although the two pass phrases can be different, the command uses two pass phrases that are the same.

2. The pass phrase stash file for BOB is renamed BOB_Stash.sth.

3. The openssl pkcs12 command is used to export the file ALICE.pem, which is created by the shell script create_ALICE_cert.sh. The command creates a pass phrase stash file for ALICE called Stash.sth.

   The command also converts the contents of ALICE.pem to PKCS#12 format and stores the converted data in a file called ALICE.p12. But this is not the primary purpose of the command.

   The -passin option on the command supplies the pass phrase for the ALICE’s private key, which the command uses to decrypt the private key stored in PEM format in ALICE.pem. The -passout option on the command supplies a second pass phrase for ALICE’s private key, which the command uses to encrypt the private key stored in PKCS#12 format
Working with the WebSphere MQ SSL support

in ALICE.p12. Although the two pass phrases can be different, the command uses two pass phrases that are the same.

4. The pass phrase stash file for ALICE is renamed ALICE_Stash.sth.

createcerts.sh
This shell script calls the following shell scripts in the stated sequence:
1. create_root_cert.sh
2. create_bigco_cert.sh
3. create_ALICE_cert.sh
4. create_BOB_cert.sh

The procedure uses the following shell scripts and MQSC command files to configure queue manager ALICE, including the two channels that ALICE needs in order to exchange messages with queue manager BOB:

ALICE.sh
This shell script creates and starts queue manager ALICE, and starts a TCP/IP listener for ALICE. You might need to modify the parameters of the runmqslsr command for your network. The script also uses the runmqsc command to run the MQSC commands in the MQSC command file ALICE.mqsc.

ALICE.mqsc
This MQSC command file contains the MQSC commands to create the queues and channels that ALICE needs in order to exchange messages with BOB. The MQSC commands do not configure the channels for using SSL.

ALICE_secure.sh
This shell script uses the runmqsc command to run the MQSC commands in the MQSC command file ALICE_secure.mqsc.

ALICE_secure.mqsc
This MQSC command file contains the MQSC commands to convert the channels to SSL channels. The channel ALICE.TO.BOB uses the RC4_SHA_US CipherSpec, which means that it uses the RC4 encryption algorithm and the SHA message digest algorithm. The channel BOB.TO.ALICE uses the NULL_MD5 CipherSpec, which means that it uses the MD5 message digest algorithm but does not encrypt messages that flow across the channel. By default, client authentication is required on the channel BOB.TO.ALICE.

A similar set of sample shell scripts and MQSC command files are provided for configuring queue manager BOB.

The procedure also uses the following shell scripts:

installcerts.sh
This shell script copies ALICE’s certificate store and pass phrase stash file to the default directory for ALICE’s SSL files. ALICE’s certificate store is created by the shell script create_ALICE_cert.sh, and ALICE’s pass phrase stash file is created by the shell script exportcerts.sh.

The script also performs the equivalent function for BOB’s certificate store and pass phrase stash file.

setup.sh
This shell script calls the following shell scripts in the stated sequence:
1. ALICE.sh
2. BOB.sh
3. createcerts.sh
Working with the WebSphere MQ SSL support

4. exportcerts.sh
5. installcerts.sh

Building and verifying the sample configuration
Use the following procedure to build and verify the sample configuration:
1. Examine the sample shell scripts and MQSC command files and modify them if necessary for your network.
   In particular, check the ports numbers specified by DEFINE CHANNEL commands in ALICE.mqsc and BOB.mqsc, and by the runmqslr commands in ALICE.sh and BOB.sh. Make sure that these port numbers are not already in use on your system.
2. Run the shell script setup.sh. In summary, the script fully configures queue managers ALICE and BOB by performing the following function:
   a. Creates and starts ALICE, and starts a listener for ALICE.
   b. Creates the queues and channels that ALICE needs in order to exchange messages with BOB. At this stage, the channels are not configured for using SSL.
   c. Creates and starts BOB, and starts a listener for BOB.
   d. Creates the queues and channels that BOB needs in order to exchange messages with ALICE. At this stage, the channels are not configured for using SSL.
   e. Creates and installs the certificate stores and pass phrase stash files for ALICE and BOB.
3. Start the channels between ALICE and BOB using the control command runmqch1 or the MQSC command START CHANNEL.
4. Verify that ALICE and BOB can exchange messages by using sample applications such as amqqput and amqqsget. Look in the MQSC command files ALICE.mqsc and BOB.mqsc to find out which queues to use in order to send and receive messages.
5. Stop the channels between ALICE and BOB, or wait for each channel to end automatically when its disconnect interval elapses.
6. Run the shell scripts ALICE_secure.sh and BOB_secure.sh to convert the channels to SSL channels.
7. Start the channels between ALICE and BOB again, and verify that ALICE and BOB can exchange messages.

Running one of the queue managers on another system
If you want to run one of the queue managers on a different NonStop OS system, you can modify the sample shell scripts and MQSC command files so that ALICE is created and configured on one system and BOB on the other.

If you want to run one of the queue managers, BOB say, on a different platform, using a different WebSphere MQ product, you can still use the sample shell scripts to prepare the key and certificate data for BOB on the NonStop OS system. The script create_BOB_cert.sh creates a file called BOB.pem, which contains the key and certificate data in PEM format. The script exportcerts.sh then converts the contents of BOB.pem to PKCS#12 format and stores the converted data in a file called BOB.p12.

If the key repository on the system on which BOB runs requires key and certificate data in PEM format, you can transfer the file BOB.pem to that system. If the key repository requires key and certificate data in PKCS#12 format, you can transfer
the file BOB.p12 instead. You can then install the key and certificate data in
BOB.pem or BOB.p12 using the appropriate procedure in [WebSphere MQ Security] If
you transfer BOB.p12 to a Windows system, you might need to rename it BOB.pfx.

You might be able to use modified versions of the shell scripts BOB.sh and
BOB_secure.sh, and the MQSC command files BOB.mqsc and BOB_secure.mqsc, to
create and start BOB, start a listener for BOB, and create the queues and channels
that BOB needs to exchange messages with ALICE. However, if BOB runs on a
system that is not a UNIX system, you must rewrite the shell scripts BOB.sh and
BOB_secure.sh in a script language that can be used on that system.
Chapter 12. 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 in conjunction with the information in the WebSphere MQ Application Programming Guide.

The chapter contains the following sections:
- “Introducing units of work”
- “Using TMF for local and global units of work” on page 180
- “Configuring TMF for WebSphere MQ” on page 184

Introducing units of work

This section introduces the concepts and terminology associated with units of work. If you are already familiar with units of work, you can omit this section.

A resource manager is a computer subsystem that owns and manages resources that can be accessed and updated by applications. A WebSphere MQ queue manager is an example of a resource manager, and the resources of the queue manager are its queues.

When an application updates the resources of one or more resource managers, there might be a business requirement to ensure that certain updates all complete successfully as a group, or none of them complete. The reason for this kind of requirement is that the business data would be left in an inconsistent state if some of the updates completed successfully, but others did not.

Updates to resources that are managed in this way are said to occur within a unit of work, or a transaction. During a unit of work, an application issues requests to resource managers to update their resources. The unit of work ends when the application issues a request to commit all the updates. Until the updates are committed, none of them become visible to other applications that are accessing the same resources. Alternatively, if the application decides that it cannot complete the unit of work for any reason, it can issue a request to back out all the updates it has requested up to that point. In this case, none of the updates ever become visible to other applications.

The point in time when all the updates within a unit of work are either committed or backed out is called a syncpoint. An update within a unit of work is said to occur within syncpoint control. If an application requests an update that is outside of syncpoint control, the resource manager commits the update immediately, even if there is a unit of work in progress, and the update cannot be backed out subsequently.

The computer subsystem that manages units of work is called a transaction manager, or a syncpoint coordinator. A transaction manager is responsible for ensuring that all updates to resources within a unit of work complete successfully, or none of them complete. It is to a transaction manager that an application issues a request to commit or back out a unit of work.
Units of work

WebSphere MQ distinguishes between a local unit of work and a global unit of work:

Local unit of work
In a local unit of work, an application updates only the resources of the WebSphere MQ queue manager to which it is connected.

Global unit of work
In a global unit of work, an application can update WebSphere MQ resources and the resources of other resource managers, such as the tables of a relational database.

Using TMF for local and global units of work

The Transaction Management Facility (TMF) is the native transaction manager on NonStop OS and is integrated with the file system and the relational database managers, SQL/MP and SQL/MX. WebSphere MQ for HP NonStop Server uses TMF in the following ways:

- To coordinate local units of work.

  An application updates WebSphere MQ resources within a local unit of work by issuing MQPUT, MQPUT1, and MQGET calls within syncpoint control. The application can then commit the local unit of work by calling MQCMIT, or back it out by calling MQBACK. Each connection to a queue manager can have only one local unit of work in progress at a time.

  As far as the application is concerned, the queue manager to which it is connected acts as the transaction manager. Internally, however, the queue manager exploits the facilities of TMF to perform this role. The use of TMF for this purpose is transparent to the application, and the application does not use TMF API calls.

- To coordinate global units of work.

  TMF acts as the transaction manager, and an application must use the API provided by TMF to start, commit, and back out global units of work.

  An application starts a global unit of work by calling BEGINTRANSACTION, and then updates WebSphere MQ resources within the global unit of work by issuing MQPUT, MQPUT1, and MQGET calls within syncpoint control. The application can then commit the global unit of work by calling ENDTRANSACTION, or back it out by calling ABORTTRANSACTION.

  Because an application can participate in multiple global units of work at the same time, the application must also ensure that each MQPUT, MQPUT1, or MQGET call is performed within the correct unit of work. A global unit of work is not associated with a specific connection to a queue manager.

  Within a global unit of work, as well as updating WebSphere MQ resources, an application can update Enscribe files, SQL/MP databases, or SQL/MX databases.

  - To ensure the integrity of critical internal databases such as the object catalog. The use of TMF for this purpose is transparent to applications.
  
  - To ensure the integrity of a persistent message that is put on a queue, or removed from a queue, outside of syncpoint control. WebSphere MQ uses a TMF transaction to update the files containing the message data for the persistent message. The use of TMF for this purpose is transparent to the application.

Note that, if an application calls MQPUT or MQPUT1 without either of the options MQPMO_SYNCPOINT or MQPMO_NO_SYNCPOINT, or calls MQGET without any of the options MQGMO_SYNCPOINT, MQGMO_SYNCPOINT_IF_PERSISTENT, or MQGMO_NO_SYNCPOINT, the call is
Using global units of work

A global unit of work is implemented as a TMF transaction. An application starts a global unit of work by calling BEGINTRANSACTION, and either commits the unit of work by calling ENDTRANSACTION or backs out the unit of work by calling ABORTTRANSACTION. An application can use other TMF API calls as well.

A server application can inherit a TMF transaction from a client application. The server application can perform work within the transaction before replying and passing the transaction back to the client application for further processing. Both the client and the server application can therefore participate in the same global unit of work that involves updates to WebSphere MQ queues and updates to files and databases. The ability to pass a TMF transaction between applications means that several WebSphere MQ applications can perform messaging operations within the same global unit of work.

An application can manage and control multiple active TMF transactions at the same time. The transactions can be started by the application itself, or inherited from other applications, or both. This means that an application can participate in multiple global units of work at the same time.

The maximum number of concurrent active TMF transactions per process is 1000, which is an architectural limit. If an application is managing multiple TMF transactions, only one transaction can be current at any point in time. Alternatively, none of the transactions can be current. The application can use TMF API calls such as RESUMETRANSACTION, ACTIVATERECEIVETRANSID, and TMF_SET_TX_ID to move the state of being current from one transaction to another, or to designate that no transaction is current. The application uses this level of control to determine whether a messaging operation is performed within a local unit of work, a global unit of work, or outside of syncpoint control:

- If an application calls MQPUT, MQPUT1, or MQGET within syncpoint control when no TMF transaction is current, WebSphere MQ processes the call within a local unit of work. For more information about using local units of work, see “Using local units of work” on page 182.

- If an application calls MQPUT, MQPUT1, or MQGET within syncpoint control when the application has a current TMF transaction, WebSphere MQ processes the call within the global unit of work implemented by the current TMF transaction.
Using TMF

- If an application calls MQPUT, MQPUT1, or MQGET outside of syncpoint control, WebSphere MQ processes the call outside of syncpoint control, irrespective of whether the application has a current TMF transaction at the time of the call.

WebSphere MQ never changes the state of an application’s TMF transaction during an MQI call, except when a software or hardware failure occurs during processing and WebSphere MQ or the operating system determines that the transaction must be backed out to preserve data integrity. Every MQI call restores the transaction state of the application just before returning control to the application.

Using local units of work

If an application calls MQPUT, MQPUT1, or MQGET within syncpoint control when the application has no current TMF transaction, WebSphere MQ processes the call within a local unit of work. If no local unit of work is in progress at the time of the call, WebSphere MQ starts one. An application commits a local unit of work by calling MQCMIT, or backs out a local unit of work by calling MQBACK.

Each connection to a queue manager can have a maximum of one local unit of work in progress at any one time. However, an application can participate in a local unit of work and one or more global units of work concurrently, if required.

Avoiding long running transactions

Avoid designing applications in which TMF transactions remain active for more than a few tens of seconds. Long running transactions can cause the circular audit trail of TMF to fill up. Because TMF is a critical system wide resource, TMF protects itself by backing out application transactions that are active for too long.

Suppose that the processing within an application is driven by getting messages from a queue, and that the application gets a message from the queue and processes the message within a unit of work. Typically, an application calls MQGET with the wait option and within syncpoint control to get a message from the queue. In this case, WebSphere MQ does not start a TMF transaction for a local unit of work until a message is actually retrieved from the queue.

If the application is using a global unit of work instead, the specified wait interval on the MQGET call must be short in order to avoid a long running transaction. This means that the application might have to issue the MQGET call more than once before it retrieves a message. Alternatively, the application can call MQGET with the set signal option, instead of the wait option. If no suitable message is available at the time of the call, control returns to the application without waiting for a message to arrive. When the application receives notification that a suitable message has arrived, the application can start a global unit of work and reissue the MQGET call to retrieve the message.

Syncpoint limits

The file system can limit the amount of persistent message data that can be put or got within a single transaction by limiting the number of record locks on the physical files that hold persistent message data.

The file system default lock limit per transaction is 5000 locks per disk volume. You can change this using SCF. For example, the following command changes the limit to 10000 locks per transaction per disk volume:

```
ALTER DISK $DISK01, MAXLOCKSPERTCP 10000
```
For messages that are stored in queue overflow files (because their size is below the threshold size for the use of message overflow files) the number and size of messages is limited.

We advise you to set the message overflow threshold size to no more than 200 KB. At this message size, the default record lock limit can accommodate about 100 messages within a single transaction, which is more than adequate for most applications. In addition, the performance benefits of using message overflow files become significant at this message size.

Also note that the use of TMF audit trail is greatly reduced when message overflow files are used instead of queue overflow files, provided the message overflow files are not audited. If you choose to use audited message overflow files, there is no reduction in TMF audit trail usage.

For more information about the differences and benefits of message and queue overflow files, see “Message overflow files” on page 92.

Performing operations on persistent messages outside of syncpoint control

Persistent messages require TMF transactions to be started internally by the queue server in order to update the Enscribe files that hold message data. There is a limit imposed by the NonStop OS file system of 1000 concurrent transactions started by any one process. Therefore, a single queue server can support no more than 1000 concurrent MQPUT, MQPUT1, and MQGET calls that involve persistent messages and are issued outside of syncpoint control.

If this situation occurs, the MQPUT or MQGET call fails with reason code MQRC_SYNCPOINT_LIMIT_REACHED. Reassign queues to alternate queue servers in order to spread the load across multiple processes, or change applications to use different queues hosted by different queue servers.

Performing operations on nonpersistent messages within a unit of work

Because nonpersistent messages are stored in memory managed by a queue server, and not in audited files on disk, they require no audit trail space. WebSphere MQ uses an internal interface of TMF to control the availability of nonpersistent messages that are enqueued or dequeued within a unit of work. Any mixture of persistent and nonpersistent messages can be included within a unit of work. WebSphere MQ ensures that, at the time the TMF transaction completes, operations on the nonpersistent messages are committed or backed out at the same time that operations on the persistent messages are committed or backed out.

The number of concurrent active transactions for an application

The maximum number of concurrent active transactions that a process can use is determined by the depth of the TMF transaction file, or T-file, for the process. The maximum possible depth of a T-file is 1000.

Each active transaction for a process occupies an entry in the T-file. When all the entries in the T-file are occupied, no further transactions can be worked on until a transaction completes and frees up an entry in the T-file. In this situation, attempts to start or use a new transaction fail with the file system error code 83.
Using TMF

WebSphere MQ operation also fails in this situation if it requires a new transaction. The reason code returned is MQRC_SYNCPOINT_NOT_AVAILABLE. It is important to know, therefore, the maximum number of concurrent active transactions required by your application. This number must include the transactions that are started by WebSphere MQ for an application process, as well as the transactions started or inherited by the application code itself.

WebSphere MQ starts a new transaction for an application when it starts a local unit of work. For an application that uses standard binding, this is the only time that WebSphere MQ starts a transaction for an application.

For an application that uses fastpath binding, there are some additional situations when WebSphere MQ starts a transaction for an application:

- The application opens a model queue to create a dynamic queue.
- The application closes a dynamic queue and the dynamic queue is deleted at the same time.
- The application calls MQSET to modify the attributes of a queue.

If your application might perform any of these operations while other transactions are active, you must take them into account when calculating the maximum number of concurrent active transactions required by the application.

For a NonStop OS application, or an OSS application that is not threaded, the T-file is not open initially, and the application process can have only one active TMF transaction at a time. This situation might be sufficient for many applications, such as an application that uses standard binding and needs to participate in only one unit of work, local or global, at a time.

If your application needs more than one concurrent active TMF transaction, the application must call FILE_OPEN_ to open the T-file with a depth that equals or exceeds the maximum number of concurrent active transactions required by the application. The FILE_OPEN_ call must specify the process name $TMP as a parameter, and the application must call FILE_OPEN_ before performing any work involving TMF transactions.

For a threaded OSS application, the pthread library opens the T-file with a default depth of 100. If necessary, an application can then use the pthread library function spt_setTMF ConcurrentTransactions() to adjust the depth.

Configuring TMF for WebSphere MQ

Your NonStop OS system needs to be configured with TMF auditing enabled for all volumes that are to contain queue managers or queues. Use the TMFCOM command status datavols to determine the status of auditing on any volume on your system. (Note that you might have to be SUPER.SUPER to use TMFCOM.) In addition, the TMF audit trails configured for the data volumes that support queue managers must be large enough to allow for the peak rate and size of message traffic expected on all queue managers that use these volumes.

TMF cancels long running transactions automatically, but the default TMF AutoAbort timeout setting is 7200 seconds, or 2 hours. You might want to consider whether this is the most appropriate setting for the transaction processing requirements of your applications, and adjust the setting if it is not. You might also want to adjust the size of a TMF audit trail used by WebSphere MQ. An audit trail used by WebSphere MQ does not need to be configured for dumping to tape.
**Configuring TMF**

**Monitoring**

Use the TMFCOM interface to monitor the status of TMF, with WebSphere MQ running. Use the status tmf and status datavols commands to investigate the general status of TMF, and the status of individual data volumes.

The System event log (EMS) should also be monitored for critical TMF events that indicate potential future problems within TMF that could affect WebSphere MQ or the applications that use it. TMF is a critical resource for WebSphere MQ and must operate continuously for WebSphere MQ to function properly.

**Audit trail size**

Approximate TMF audit trail sizings can be calculated using the following guidelines:

- Audit trail space is required for persistent message operations (put and destructive get) only.
- The audit trail space should be approximately the total message data size plus 1500 bytes.
- Provided message overflow files are not audited, messaging operations involving messages that are above the message overflow threshold require only 4 KB of audit trail per put or get, irrespective of their size.

**Resource manager configuration**

The internal interface of TMF needs to be configured appropriately for the volume of transactions that are expected to be processed using WebSphere MQ. The WebSphere MQ queue servers take the role of resource manager as far as the TMF subsystem is concerned, and there are various thresholds and limits in the TMF subsystem that apply to resource managers. The required configuration depends on the number of queue servers you use, the distribution of queue servers across the CPUs and how many concurrent syncpoint operations are in progress at any one time. The ALTER BEGINTRANS command of TMFCOM is used to change the values, as described below:

**RMOPENPERCPU**

Should be at least twice the maximum number of queue servers that will run in any CPU. The default value of 128 is usually sufficient.

**BRANCHESPERRM**

Should be at least the maximum number of concurrent syncpoint operations that can be handled by any single queue server. The default value of 128 is usually sufficient, but if not then this parameter may be increased to the maximum value of 1024, or Queues may be assigned to other queue servers to reduce the maximum number of concurrent syncpoint operations handled by a queue server.

For new values of these parameters to take effect, the TMF subsystem must be stopped and restarted.

**Troubleshooting**

EMS events or FFST reports indicating that BEGINTRANSACTION commands have been disabled by TMF usually mean that the audit trail is filled. This can occur because the audit trail is too small, or because a badly behaved application has held a long-running transaction and TMF has not terminated it in time.

In this instance:
Configuring TMF

- Increase the size of the audit trail, or
- Identify the cause of the long-running transaction and correct it, or
- Re-configure TMF to terminate long-running transactions after a shorter period.

EMS events and FFST reports indicating that TMF is not running indicate a configuration problem with TMF that must be corrected before running the queue manager again. In general, the WebSphere MQ queue manager requires TMF to be running correctly to operate in any capacity. Although messages are not lost or corrupted, the queue manager is not able to operate without TMF.
Chapter 13. 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” on page 187
- “The DLQ handler rules table” on page 188
- “How the rules table is processed” on page 194
- “An example DLQ handler rules table” on page 195

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.rul
  ```
- In the rules table. For example:
  ```
  INPUTQ(ABC1.DEAD.LETTER.QUEUE) INPUTQM(ABC1.QUEUE.MANAGER)
  ```

The above examples apply to the DLQ called ABC1.DEAD.LETTER.QUEUE, owned by the queue manager ABC1.QUEUE.MANAGER.

If you do not specify the DLQ or the queue manager as shown above, the default queue manager for the installation is used along with the DLQ belonging to that default 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 278.

The sample DLQ handler, amqsd1q

In addition to the DLQ handler invoked using the runmqdlq command, WebSphere MQ provides the source of a sample DLQ handler, amqsd1q. The function of amqsd1q is similar to that provided by runmqdlq. You can customize amqsd1q 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, amqsd1q, process only those messages on the DLQ that begin with a dead-letter header, MQDLH. Messages that do not begin with an MQDLH are identified as being in error, and remain on the DLQ indefinitely.)

The source of amqsd1q is supplied in the directory:

```
/opt_installation_path/opt/mqm/samp/dlq
```

and the compiled version is supplied in the directory:

```
/opt_installation_path/opt/mqm/samp/bin
```

The DLQ handler rules table

The DLQ handler rules table defines how the DLQ handler processes 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 reprocesses 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.
DLQ handler

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.

APPLDAT (ApplIdentityData1*)
The ApplIdentityData value specified in the message descriptor, MQMD, of the message on the DLQ.

APPLNAME (PutAppName1*)
The name of the application that issued the MQPUT or MQPUT1 call, as specified in the PutAppName field of the message descriptor MQMD of the message on the DLQ.

APPLTYPE (PutApplType1*)
The PutApplType value, specified in the message descriptor MQMD, of the message on the DLQ.

DESTQ (QueueName1*)
The name of the message queue for which the message is destined.

DESTQM (QueueManagerName1*)
The name of the queue manager of the message queue for which the message is destined.

FEEDBACK (Feedback1*)
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 (Format1*)
The name that the sender of the message uses to describe the format of the message data.

MSGTYPE (MsgType1*)
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 (Persistence1*)
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 (ReasonCode1*)

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 (QueueName1*)

The name of the reply-to queue specified in the message descriptor, MQMD, of the message on the DLQ.

### REPLYQM (QueueManagerName1*)

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 (UserIdentifier1*)

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 (QueueName1 &DESTQ &REPLY)

The name of the message queue to which to forward the message when ACTION (FWD) is requested.

- **QueueName**
  - The name of a message queue. FWDQ(' ') is not valid.

- **&DESTQ**
  - Take the queue name from the DestQName field in the MQDLH structure.

- **&REPLYQ**
  - Take the queue name from the ReplyToQ field in the message descriptor, MQMD.

  To avoid error messages when a rule specifying FWDQ (&REPLYQ) matches a message with a blank ReplyToQ field, specify REPLYQ (?)* in the message pattern.

### FWDQM (QueueManagerName1 &DESTQM &REPLYQM1 ' ')

The queue manager of the queue to which to forward a message.
DLQ handler

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 principal 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 that 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:
    
    | Parameter    | Description               |
    |--------------|---------------------------|
    | 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:
    
    | Parameter    |
    |--------------|
    | FORMAT('')   |
    | FORMAT()     |
    | FORMAT       |

  - Wildcard characters are supported. You can use the question mark (?) instead of any single character, except a trailing blank. You can use the asterisk (*) instead of zero or more adjacent characters. The asterisk (*) and the question mark (?) are always interpreted as wildcard characters in parameter values.
  - Wildcard characters cannot be included in the parameters of these keywords: ACTION, HEADER, RETRY, FWDQ, FWDQM, and PUTAUT.
  - Trailing blanks in parameter values, and in the corresponding fields in the message on the DLQ, are not significant when performing wildcard matches. However, leading and embedded blanks within strings in quotation marks are significant to wildcard matches.
  - Numeric parameters cannot include the question mark (?) wildcard character. You can use the asterisk (*) instead of an entire numeric parameter, but not as part of a numeric parameter. For example, these are valid numeric parameters:
    
    | Parameter    | Description               |
    |--------------|---------------------------|
    | 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.
DLQ handler

- 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. However, 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 rescanned 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. 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
```
DLQ handler

* 60-second intervals (the default value for RETRYINT).

REASON(MQRC_Q_FULL) ACTION(RETRY) RETRY(5)

* If a message is placed on the DLQ because of a put inhibited
  * condition, attempt to forward the message to its
  * destination queue. Make 5 attempts at approximately
  * 60-second intervals (the default value for RETRYINT).

REASON(MQRC_PUT_INHIBITED) ACTION(RETRY) RETRY(5)

* The AAAA corporation are always sending messages with incorrect
  * addresses. When we find a request from the AAAA corporation,
  * we return it to the DLQ (DEADQ) of the reply-to queue manager
  * (&REPLYQM).
  * The AAAA DLQ handler attempts to redirect the message.

MSGTYPE(MQMT_REQUEST) REPLYQM(AAAA.*) +
  ACTION(FWD) FWDQ(DEADQ) FWDQM(&REPLYQM)

* The BBBB corporation never do things by half measures. If
  * the queue manager BBBB.1 is unavailable, try to
  * send the message to BBBB.2

DESTQM(bbbb.1) +
  action(fwd) fwdq(&DESTQ) fwdqm(bbbb.2) header(no)

* The CCCC corporation considers itself very security
  * conscious, and believes that none of its messages
  * will ever end up on one of our DLQs.
  * Whenever we see a message from a CCCC queue manager on our
  * DLQ, we send it to a special destination in the CCCC organization
  * where the problem is investigated.

REPLYQM(CCCC.*) +
  ACTION(FWD) FWDQ(ALARM) FWDQM(CCCC.SYSTEM)

* Messages that are not persistent run the risk of being
  * lost when a queue manager terminates. If an application
  * is sending nonpersistent messages, it should be able
  * to cope with the message being lost, so we can afford to
  * discard the message. PERSIST(MQPER_NOT_PERSISTENT) ACTION(DISCARD)
  * For performance and efficiency reasons, we like to keep
  * the number of messages on the DLQ small.
  * If we receive a message that has not been processed by
  * an earlier rule in the table, we assume that it
  * requires manual intervention to resolve the problem.
  * Some problems are best solved at the node where the
  * problem was detected, and others are best solved where
  * the message originated. We don't have the message origin,
  * but we can use the REPLYQM to identify a node that has
  * some interest in this message.
  * Attempt to put the message onto a manual intervention
  * queue at the appropriate node. If this fails,
  * put the message on the manual intervention queue at
  * this node.

REPLYQM(?!.*) +
  ACTION(FWD) FWDQ(DEADQ.MANUAL.INTERVENTION) FWDQM(&REPLYQM)

ACTION(FWD) FWDQ(DEADQ.MANUAL.INTERVENTION)
Chapter 14. Process management

Queue manager processes are created and managed in two ways:
- Processes that are created and managed automatically by the execution controller
- Processes that are Pathway server classes

This chapter is concerned with the first method and describes the configuration and management features of the execution controller. It is important to understand how the execution controller performs this function so that you can configure the queue manager in a way that supports your business. This is a powerful tool that can be used to balance workload throughout the system. For example, you can control execution controller managed processes, priorities, names, and CPU usage. Process-naming capabilities allow you to quickly identify process types. For example, you can name all message channel agents (MCAs) so that the first four characters are always $MCA.

The execution controller directly manages the following processes:
- Local queue manager agent (LQMA) processes that service local application connections
- Message channel agent (MCA) processes that service channels
- Repository manager (REPMAN) processes that manage the cluster repository information for the queue manager

The execution controller uses the process management rules configuration file, qmproc.ini, to provide the configuration options and rules for its process management functions. An initial version of the file is created by the crtmqm command when you create the queue manager. You can customize the file to provide installation-specific conventions; for example, for process naming, CPU usage, and priorities. The contents of the initial qmproc.ini file derive from an installation-wide template called proc.ini. You can customize this template to provide system-specific process management defaults to be used by all queue managers created in the installation.

The execution controller reads and processes the configuration file at the following times:
- At startup
- Whenever it receives a command to reread the configuration file (because the information has been updated while the queue manager is running).

The execution controller always validates the contents of the file before acting on it. If an error or conflict is found, the execution controller issues an appropriate error message and takes action based on the current queue manager state as follows:
1. If the queue manager detects a validation failure during startup, the queue manager continues startup using a default set of values.
2. If a validation failure is detected while the queue manager is running (because the MQSC command RESET QMGR TYPE(NSPROC) or equivalent PCF command has been issued), the execution controller logs the appropriate error messages. The execution controller does not implement the attributes and rules defined in the updated file.
Process management

The new rules do not affect processes that already exist when the file is modified. However, take care not to delete rules that affect running channels or applications. In this situation, stop or end the application or channel before you delete the rule that initiated the process.

Any process started by the execution controller that is not assigned a name by one of the rules is given a name by the operating system.

Any process started by the execution controller that is not assigned to run in a specific CPU is run in a CPU chosen by the operating system.

Attributes and rules

This section describes the categories of information that you can describe in the qmproc.ini configuration file. The actual keywords and stanza names are detailed in “Keyword definitions” on page 203.

Default process attributes

These provide default attribute values for all execution controller managed processes. These attribute values are used unless overridden by an agent-specific attribute or rule.

Table 15. Default process attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU list</td>
<td>A list of one or more CPUs in which the execution controller can start agent processes. The execution controller does not create agents in CPUs that are not on this list, unless other process management rules override the list, or all the CPUs in the list are unavailable. The default is all configured CPUs.</td>
</tr>
<tr>
<td>Environment variables</td>
<td>Up to 10 environment variables that the execution controller passes to processes when the processes are created. The installation root environment variable is passed by default.</td>
</tr>
<tr>
<td>Executables path</td>
<td>An OSS path string to be used for processes started by the execution controller. The default is the OSS executables directory that was originally installed.</td>
</tr>
<tr>
<td>Fail if CPU unavailable</td>
<td>If a rule results in an attempt to create a process in a CPU that is not available or not configured, this Boolean controls whether the execution controller either rejects the connection or channel startup, or chooses a different CPU instead. In both cases, the execution controller logs a message indicating the problem. The default is No.</td>
</tr>
<tr>
<td>Fail if process name unavailable</td>
<td>If a rule results in an attempt to create a process with a process name that is either unavailable (because it is already in use) or invalid, this Boolean controls whether the execution controller either rejects the connection or channel startup, or chooses a different process name instead. In both cases, the execution controller logs a message indicating the problem. The default is No.</td>
</tr>
</tbody>
</table>
### Agent attributes

These provide values for LQMA and MCA processes, known collectively as agent processes.

You can specify these attributes separately for each of the two types of agent process. The attributes specified for a type of agent apply to all processes of that type of agent, unless overridden by rules from the application or channel categories. The attributes specified in this category override any that apply from the AllProcesses stanza.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agent CPU list</td>
<td>A list of one or more CPUs in which the execution controller can create specific types of agent processes, overriding the default for all agents. The execution controller does not create this type of agent in CPUs that are not on this list, unless other process management rules override the list, or all the CPUs in the list are unavailable.</td>
</tr>
<tr>
<td>Agent environment variables</td>
<td>Up to 10 environment variables that the execution controller passes to agent processes of a specific type when the processes are created. These environment variables completely replace any environment variables configured for all processes.</td>
</tr>
<tr>
<td>Agent executable name</td>
<td>The executable file name that the execution controller uses for agent processes of a specific type. This name overrides the built-in default.</td>
</tr>
<tr>
<td>Agent executables path</td>
<td>An OSS path string used by the execution controller for agent processes of a specific type.</td>
</tr>
<tr>
<td>Agent priority</td>
<td>The execution priority used by the execution controller for agent processes of a specific type.</td>
</tr>
<tr>
<td>Agent process name root</td>
<td>Specifies a template for the beginning of a process name that the execution controller uses for all agent processes of a specific type. The template is of the form $ANxxx or $ANNxx, where A is an alphabetic character, N is an alphanumeric character and x represents a character chosen by the execution controller. The execution controller always uses the maximum number of characters in a process name ($ plus five characters) unless overridden by a more specific application or channel rule.</td>
</tr>
<tr>
<td>Maximum number of idle unthreaded agents</td>
<td>The execution controller limits the maximum number of unthreaded agents of a specific type to this number. The default value is 10.</td>
</tr>
</tbody>
</table>
## Agent attributes

Table 16. Process management: agent attributes (continued)

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum number of threaded agents</td>
<td>The execution controller limits the number of threaded agents of a specific type that are not idle. When an agent completes its work, if the idle pool maximum is exceeded the agent does not return to the pool, but exits.</td>
</tr>
<tr>
<td></td>
<td>The default value is 10.</td>
</tr>
<tr>
<td>Maximum number of threads for each threaded agent</td>
<td>The execution controller does not create more than this number of threads in a threaded agent of a specific type.</td>
</tr>
<tr>
<td></td>
<td>The default value is 50.</td>
</tr>
<tr>
<td>Maximum number of unthreaded agents</td>
<td>The execution controller limits the number of unthreaded agents of a specific type that are not idle. When an agent completes its work, if the idle pool maximum is exceeded, the agent does not return to the pool, but ends.</td>
</tr>
<tr>
<td></td>
<td>The default value is 20.</td>
</tr>
<tr>
<td>Maximum reuse count for threaded agents</td>
<td>The execution controller reuses threaded agents of a specific type up to this maximum before replacing the threaded agent with a new process.</td>
</tr>
<tr>
<td></td>
<td>The default value is 10.</td>
</tr>
<tr>
<td>Maximum reuse count for unthreaded agents</td>
<td>The execution controller reuses unthreaded agents of a specific type up to this maximum before replacing the unthreaded agent with a new process.</td>
</tr>
<tr>
<td></td>
<td>The default value is 10.</td>
</tr>
<tr>
<td>Minimum number of idle unthreaded agents</td>
<td>The execution controller attempts to maintain this minimum number of idle unthreaded agents of a specific type.</td>
</tr>
<tr>
<td></td>
<td>The default value is 1. A value of zero has a special meaning to the execution controller; it means that the execution controller should not maintain a unthreaded agent process pool, so agents are started dynamically when needed and stopped when finished.</td>
</tr>
<tr>
<td>Minimum number of threaded agents</td>
<td>The execution controller maintains at least this number of threaded agent processes of a specific type running.</td>
</tr>
<tr>
<td></td>
<td>The default value is 1. A value of zero has a special meaning to the execution controller; it means that the execution controller should not maintain a threaded agent process pool, so agents are started dynamically when needed and stopped when finished.</td>
</tr>
<tr>
<td>Minimum number of threads for each threaded agent</td>
<td>The execution controller attempts to distribute work for threaded agents of a specific type with this minimum number of threads for each process.</td>
</tr>
<tr>
<td></td>
<td>The default value is 2.</td>
</tr>
<tr>
<td>Preferred number of threaded agents</td>
<td>The execution controller attempts to assign work to threaded agents to maintain this number of agents. This value is exceeded only when all the threads of the all the preferred number of agents are in use.</td>
</tr>
<tr>
<td></td>
<td>The default value is 5.</td>
</tr>
<tr>
<td>Preferred number of threads for each threaded agent</td>
<td>The execution controller attempts to distribute work for threaded agents of a specific type with this target number of threads for each process.</td>
</tr>
<tr>
<td></td>
<td>The default value is 10.</td>
</tr>
</tbody>
</table>
Table 16. Process management: agent attributes (continued)

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Threaded agents</td>
<td>Whether LQMA agents are threaded or unthreaded. This attribute applies to LQMAs only.</td>
</tr>
</tbody>
</table>

**Application rules**

The rules in this category provide process management attributes for the LQMA processes that service connections from specific application processes, programs, or sets of application processes. These application rules override any attributes derived from the agent attributes category. Each rule supports the following attributes:

Table 17. Process management: application rules

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application agent CPU list</td>
<td>A list of CPUs in which the execution controller can start the LQMA that services the application identified by the rule.</td>
</tr>
<tr>
<td>Application agent environment variables</td>
<td>Up to 10 environment variables that the execution controller passes to the LQMA that services the application defined by the rule. These environment variables completely replace any environment variables configured for LQMAs.</td>
</tr>
<tr>
<td>Application agent process name</td>
<td>A unique process name that the execution controller uses for the LQMA that services the application identified by the rule, or a process name template of the form $ANxxx or $ANNxx. If this attribute is specified, the execution controller always uses a process with the specified name for the LQMA, even if the threaded attribute is set to Yes and there are threads available in other LQMAs. If no threads are available in the named process, the connection attempt is rejected.</td>
</tr>
<tr>
<td>Application agent threaded</td>
<td>A Boolean indicating whether the LQMA that services the application identified by the rule is threaded.</td>
</tr>
</tbody>
</table>

Application rules are processed in priority order, according to the rule number described below. Rule 1 has highest priority. Within each rule number, the execution controller scans the rules in sequence. The first rule that applies determines the process attributes.

**Rule 1: Application process name match**

If the application process name matches exactly, this rule is applied. This rule is designed for handling specific process instances of applications.

**Rule 2: Application program name full path match**

If the application program name matches exactly, this rule is applied. You can specify program names as fully-qualified OSS file names or NonStop OS fully-qualified local file names. OSS program names are case-sensitive. NonStop OS program names are case-insensitive. This rule is designed for handling connections from a specific application, possibly multiple process instances.

**Rule 3: Application executable file name match**

If the application program file name part matches exactly, this rule is applied. This rule is designed for handling connections from multiple executable programs of the same name in different locations.

**Rule 4: Application directory and subvolume match**

If the fully-qualified path name of the directory or fully-qualified local name of the subvolume (on NonStop OS) containing the application
Application rules

executable matches, this rule is applied. This rule is designed for handling sets of different applications in a single location.

Rule 5: Application CPU match
If the application runs in one of the list of specified CPUs, this rule applies.

Channel rules

The rules in this category provide attributes for MCA processes. The channel rules override any attributes derived from the agent attributes category. Each rule supports the following attributes:

Table 18. Process management: channel rules

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MCA CPU list</td>
<td>A list of CPUs in which the execution controller can start an MCA that implements the channel identified by the rule.</td>
</tr>
<tr>
<td>MCA environment variables</td>
<td>Up to 10 environment variables that the execution controller passes to the MCA that services the channel defined by the rule. These environment variables completely replace any environment variables configured for MCAs.</td>
</tr>
<tr>
<td>MCA process name</td>
<td>A unique process name that the execution controller uses for the MCA that implements the channel identified by the rule. Do not use this attribute with a rule that might apply to more than one process.</td>
</tr>
<tr>
<td>MCA process name root</td>
<td>A process name template of the form $ANxxx or $ANNxx. If this attribute is specified, the execution controller always uses a process with the specified name for the MCA, even if the threaded attribute is set to Yes and there are threads available in other MCAs.</td>
</tr>
<tr>
<td>MCA threaded</td>
<td>A Boolean indicating whether the MCA that services the channel identified by the rule is threaded.</td>
</tr>
<tr>
<td>SNALocalLU</td>
<td>The Local LU for SNA channels. This parameter is used by SNA channels only.</td>
</tr>
<tr>
<td>SNALocalTP</td>
<td>The Local TP Name for SNA channels. This parameter is used by SNA channels only.</td>
</tr>
<tr>
<td>Transport Provider Name</td>
<td>The process name of the TCP/IP or SNA transport provider that corresponds to the transport protocol of the channel.</td>
</tr>
</tbody>
</table>

Channel rules are processed in priority order, according to the rule number described below. Rule 1 has highest priority. Within each rule number, the execution controller scans the rules in sequence. The first rule that applies determines the process attributes.

Rule 1: Outbound channel name match
If the channel name matches, this rule is applied. This rule can apply only to outbound channels because the agents for inbound channels are started before the channel name is known.

Rule 2: Channel type match
If the channel type matches, this rule is applied. This rule applies to caller channels only. The following types of channel are supported:
- Sender
- Server
- Requester
- Cluster sender

Rule 3: Channel protocol match
If the channel protocol type matches, this rule is applied.
Repository manager

The execution controller manages a repository manager process automatically as needed. The REPMAN processes inherit the attributes of the AllProcesses stanza, unless overridden by the REPMAN attributes.

Table 19. Process management: repository manager

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prestart CPU list</td>
<td>A list of CPUs where the execution controller prestarts REPMAN processes. To avoid REPMAN servers dynamically starting up, carefully consider the distribution of processes across CPUs when you prepare this list.</td>
</tr>
<tr>
<td>Process name template</td>
<td>A process name template of the form $ANNxx that the execution controller uses for all REPMAN processes. xx indicates the CPU number that the REPMAN runs in.</td>
</tr>
</tbody>
</table>

Keyword definitions

The following is a list of all valid stanza names:
- AllProcesses
- AppRule1-ProcNameMatch
- AppRule2-ProgNameMatch
- AppRule3-ExeNameMatch
- AppRule4-SubvolMatch
- AppRule5-CpuMatch
- ChlRule1-ChannelNameMatch
- ChlRule2-ChannelTypeMatch
- ChlRule3-ChannelProtocolMatch
- LQMA
- MCA
- Pathway

Attention: The Pathway stanza is for the queue manager’s internal use only and you must not edit it. If this stanza is corrupted, you might not be able to start the queue manager. Pathway is listed here only to present this warning.

The following table summarizes:
- The keywords associated with the available attributes or rules
- The stanza where the keyword is valid
- The default value, if any

Table 20. Process management: keyword definition summary

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Keyword</th>
<th>Valid attribute values</th>
<th>Stanza</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agent Process name</td>
<td>ProcessName</td>
<td>Unique, valid process name</td>
<td>Application rules, Channel rules</td>
<td>None</td>
</tr>
<tr>
<td>Channel name to match</td>
<td>ChannelNameMatch</td>
<td>Channel name</td>
<td>ChlRule1</td>
<td>None</td>
</tr>
<tr>
<td>Channel protocol to match</td>
<td>ChannelProtocolMatch</td>
<td>TCP/IP</td>
<td>SNA</td>
<td>ChlRule3</td>
</tr>
</tbody>
</table>
### Keyword definitions

#### Table 20. Process management: keyword definition summary (continued)

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Keyword</th>
<th>Valid attribute values</th>
<th>Stanza</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Channel type to match</td>
<td>ChannelTypeMatch</td>
<td>Channel type</td>
<td>ChlRule2</td>
<td>None</td>
</tr>
<tr>
<td>CPU list</td>
<td>CPUs</td>
<td>CPU number separated by commas</td>
<td>All stanzas</td>
<td>All configured CPUs</td>
</tr>
<tr>
<td>CPU to match</td>
<td>CpuMatch</td>
<td>CPU list</td>
<td>AppRule5</td>
<td>None</td>
</tr>
<tr>
<td>Environment variables</td>
<td>ENV(xx) (xx=1-10)</td>
<td>Environment variable</td>
<td>All stanzas</td>
<td>Installation root environment variable</td>
</tr>
<tr>
<td>Executable name</td>
<td>ExecutableName</td>
<td>Complete absolute OSS path of the LQMA or MCA application executable program name</td>
<td>LQMA, MCA</td>
<td>Default system names</td>
</tr>
<tr>
<td>Executable name to match</td>
<td>ExecNameMatch</td>
<td>File name part only of the absolute path of the application executable program</td>
<td>AppRule3</td>
<td>None</td>
</tr>
<tr>
<td>Executables path</td>
<td>ExecutablePath</td>
<td>Absolute OSS path to the directory containing the application executable programs</td>
<td>AllProcesses, LQMA, MCA</td>
<td>Installed OSS path</td>
</tr>
<tr>
<td>Fail if CPU unavailable</td>
<td>FailOnCPUunavail</td>
<td>Yes or No</td>
<td>AllProcesses</td>
<td>No</td>
</tr>
<tr>
<td>Fail if process name unavailable</td>
<td>FailOnProcNameUnavail</td>
<td>Yes or No</td>
<td>AllProcesses</td>
<td>No</td>
</tr>
<tr>
<td>Maximum number of idle unthreaded agents</td>
<td>MaxIdleAgents</td>
<td>Number</td>
<td>LQMA, MCA</td>
<td>10</td>
</tr>
<tr>
<td>Maximum number of threaded agents</td>
<td>MaxThreadedAgents</td>
<td>Number</td>
<td>LQMA, MCA</td>
<td>10</td>
</tr>
<tr>
<td>Maximum number of threads for each threaded agent</td>
<td>MaximumThreads</td>
<td>Number</td>
<td>LQMA, MCA</td>
<td>50</td>
</tr>
<tr>
<td>Maximum number of unthreaded agents</td>
<td>MaxUnthreadedAgents</td>
<td>Number</td>
<td>LQMA, MCA</td>
<td>20</td>
</tr>
<tr>
<td>Maximum reuse count for threaded agents</td>
<td>MaxThreadedAgentUse</td>
<td>Number</td>
<td>LQMA, MCA</td>
<td>10</td>
</tr>
<tr>
<td>Maximum reuse count for unthreaded agents</td>
<td>MaxAgentUse</td>
<td>Number</td>
<td>LQMA, MCA</td>
<td>10</td>
</tr>
<tr>
<td>Minimum number of idle unthreaded agents</td>
<td>MinIdleAgents</td>
<td>Number</td>
<td>LQMA, MCA</td>
<td>1</td>
</tr>
<tr>
<td>Minimum number of threaded agents</td>
<td>MinThreadedAgents</td>
<td>Number</td>
<td>LQMA, MCA</td>
<td>1</td>
</tr>
</tbody>
</table>
Table 20. Process management: keyword definition summary (continued)

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Keyword</th>
<th>Valid attribute values</th>
<th>Stanza</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum number of threads for each threaded agent</td>
<td>MinimumThreads</td>
<td>Number</td>
<td>LQMA, MCA</td>
<td>2</td>
</tr>
<tr>
<td>Priority</td>
<td>Priority</td>
<td>Number</td>
<td>AllProcess, LQMA, MCA</td>
<td>165</td>
</tr>
<tr>
<td>Process name root</td>
<td>ProcessNameRoot</td>
<td>$AN or $ANN</td>
<td>LQMA, MCA, application rules, channel rules</td>
<td>None. Process names defined by system</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$ANN</td>
<td>RepositoryManager</td>
<td></td>
</tr>
<tr>
<td>Preferred number of threaded agents</td>
<td>PreferredThreadedAgents</td>
<td>Number</td>
<td>LQMA, MCA</td>
<td>5</td>
</tr>
<tr>
<td>Preferred number of threads for each threaded agent</td>
<td>PreferredThreads</td>
<td>Number</td>
<td>LQMA, MCA</td>
<td>10</td>
</tr>
<tr>
<td>Process name to match</td>
<td>ProcNameMatch</td>
<td>Valid process name</td>
<td>AppRule1</td>
<td>None</td>
</tr>
<tr>
<td>Program name to match</td>
<td>ProgNameMatch</td>
<td>Complete absolute OSS path of the application program name.</td>
<td>AppRule2</td>
<td>None</td>
</tr>
<tr>
<td>SNA Local LU</td>
<td>SNALocalLU</td>
<td>LU name</td>
<td>ChlRules</td>
<td>None</td>
</tr>
<tr>
<td>SNA Local TP</td>
<td>SNALocalTP</td>
<td>TP name</td>
<td>ChlRules</td>
<td>None</td>
</tr>
<tr>
<td>SubVol name to match</td>
<td>SubVolMatch</td>
<td>Subvolume part of the application program file name in the NonStop OS file system</td>
<td>AppRule4</td>
<td>None</td>
</tr>
<tr>
<td>Threaded agents</td>
<td>ThreadedAgents</td>
<td>Yes or No</td>
<td>AllProcesses, LQMA, application rules</td>
<td>Threaded</td>
</tr>
<tr>
<td>Transport Provider Name (TCP or SNA)</td>
<td>TransportName</td>
<td>Process name</td>
<td>ChlRules</td>
<td>TCP- $ZTCO SNA - none</td>
</tr>
</tbody>
</table>

Process management examples

Example 1: Configuring attributes for all agents

This example tells the execution controller that both MCAs and LQMAs should be started in CPUs 1 and 2 with a priority of 170. The execution controller always starts the default minimum number of agents.

```
#*******************************************************************************#
#                                                                             #
# WMQ V5.3 for HP NonStop Server                                             #
#                                                                             #
# Queue Manager process management configuration file                        #
#                                                                             #
#*******************************************************************************#
#                                                                             #
# All Processes stanza
```
Process management examples

```
# ------------------
AllProcesses:
  CPUs=1,2
  Priority=170

Example 2: Configuring attributes for types of agent

In this example LQMA are assigned some specific attributes, MCAs are created
using the AllProcesses stanza and all other attributes are default values.

This example tells the execution controller that only unthreaded LQMA should be
used to service applications, and they must be started in CPUs 2 and 3 with a
priority of 165. The unthreaded LQMA pool contains a minimum of two idle
agents, a maximum of four idle agents, and LQMA are recycled after being
allocated to an application five times. The execution controller is also told to use
$LQM as the template for LQMA process names.

Based on this information, after startup, there is an LQMA named $LQM00
running in CPU 2 with a priority of 165 and an LQMA named $LQM01 running in
CPU 3 also with a priority of 165. The execution controller tracks the template
process numbers in use to avoid creating duplicate process names, but they are
reset when the agent is stopped or recycled. Therefore, for example, when $LQM00
is recycled, the process name $LQM00 is again available for use.

AllProcesses:
  CPUs=1,2
  Priority=170

# LQMA
# ------------------
LQMA:
  ThreadedAgents=No
  ProcessNameRoot=$LQM
  CPUs=2,3
  Priority=165
  MinIdleAgents=2
  MaxIdleAgents=4
  MaxAgentUse=5

Example 3: Using threaded agent attributes

This example looks at the attributes that are associated with threaded agents and
explains how the execution controller reacts to this set of values.

The attributes shown illustrate that the threaded LQMA process pool contains a
minimum of two agents and a maximum of 10 agents. The execution controller
attempts to distribute work amongst the agents to maintain a working set of five
agents.

Next, the execution controller is told that, for each agent, it must attempt to use a
minimum of five threads, a maximum of 15 threads, and optimally the workload
should be limited to 10 threads for each agent.

The execution controller reacts to this set of threaded attributes in the following
manner. Assume for the purposes of this example that work assigned to an agent
does not end. Initially the pool consists of two threaded agents and work is
distributed between them until each has reached the preferred thread limit. At this
point there are two agents each running 10 threads. Then, when new work is
required, the execution controller starts a new agent to add to the thread pool.
The execution controller distributes work to this agent until it too has reached the
preferred thread limit. This process continues until the preferred threaded agents limit is reached, at which point there are five agents running, each using 10 threads.

As more work is required, the execution controller assigns the work to the pool agents until the maximum threads limit is reached. At this point there are five agents, each using 15 threads. Further work causes the execution controller to add agents to the pool until there are the maximum 10 agents in the pool. When the point is reached where there are maximum threads in maximum agents running, the execution controller refuses to accept any further work.

AllProcesses:
  CPUs=1,2
  Priority=170

# LQMA
# ------------------
LQMA:
  ThreadedAgents=Yes
  ProcessNameRoot=$LQM
  CPUs=2,3
  Priority=165
  MinThreadedAgents=2
  MaxThreadedAgents=10
  MaxThreadedAgentUse=50
  PreferredThreadedAgents=5
  MinimumThreads=5
  MaximumThreads=15
  PreferredThreads=10

Example 4: Using the Repository stanza
This example demonstrates the basics in the use of the Repository stanza. The key point here is that if no repository manager (REPMAN) is running in a CPU where a queue manager process, agent, or fast path application is attempting to run, the execution controller dynamically starts a REPMAN in that CPU. This delays the start of the application until the REPMAN is running. The execution controller is told to use the template $REP to name the REPMAN and is told to prestart a REPMAN in CPU 1 and 2.

AllProcesses:
  CPUs=1,2
  Priority=170

# Repository Manager stanza
# -----------------------------
RepositoryManager:
  ProcessNameRoot=$REP
  CPUs=1,2

Example 5: Using channel rules
This example demonstrates the use channel rules and the hierarchical structure of the rules. The execution controller processes the rules in priority order. Rule 1 is the highest therefore rule 1 always supersedes rules 2 and 3. For this rule set, the execution controller uses a transport provider name of $ZTC0 for all TCP/IP channels except the channel named Channel1. The MCA allocated to that channel is named $MCA01 and uses a transport provider name of $ZTC03.

If there is a pool process named $MCA01 and it is idle, it is allocated to this channel. However, if there is already a $MCA01 pool process running that is not idle, the execution controller attempts to start a new MCA process with that name.
Process management examples

What happens next depends on the value of the FailOnProcNameUnavail attribute. If it is set to Yes, the execution controller cancels the MCA allocation request with an error. If it is set to No (the default), the execution controller allows the system to choose the process name. Rule 2 states that all MCAs implementing sender channels must be named $SDRnn. However if Channel1 is a sender channel, the MCA servicing that channel is named $MCA01.

ChlRule1-ChannelNameMatch:
   ChannelNameMatch=Channel1
   ProcessName=$MCA01
   CPUs=3
   TransportName=$ZTC03

ChlRule2-ChannelTypeMatch:
   ChannelTypeMatch=Sender
   ProcessNameRoot=$SDR
   CPUs=3

ChlRule3-ChannelProtocolMatch:
   ChannelProtocolMatch=TCPIP
   TransportName=$ZTC0
   CPUs=3

Example 6: Using application rules

Application rules operate in a similar way to the channel rules. The same hierarchical structure is used by the execution controller when processing application rules. For all rules, an attribute of the application that requests service is used to set attributes of the LQMA allocated to provide service.

Rule 1 states that if the process name of the application is $NO11, the LQMA servicing the application is named $NOX5 and run in CPU3.

AppRule1-ProcNameMatch:
   ProcNameMatch=$NO11
   ProcessName=$NOX5
   CPUs=3

Rule 2 states that a NonStop OS application running with the executable program file name $DATA03.TESTAPP.FRED uses an LQMA process named $FREDA.

AppRule2-ProgNameMatch:
   ProgNameMatch=$DATA03.TESTAPP.FRED
   ProcessName=$FREDA

Rule 3 states that a NonStop OS or OSS application running with the file name part prog03 of the full path of the application program is assigned an LQMA running in CPU 2 or 3.

AppRule3-ExeNameMatch:
   ExeNameMatch=prog03
   CPUs=2,3

Rule 4 states that a NonStop OS application running as an executable program from a file in the subvolume $DATA05.TEST uses an LQMA process that is named using the template process name $TST.

AppRule4-SubvolMatch:
   SubvolMatch=$DATA05.TEST
   ProcessNameRoot=$TST
Rule 5 states that an application process running in CPU 0 or 1 is assigned an LQMA process running in either CPU 2 or 3.

\[
\text{AppRule5-CpuMatch:} \\
\text{CpuMatch}=0,1 \\
\text{CPUs}=2,3
\]

**Example 7: Using environment variables**

Environment variables can be passed to the execution controller’s managed processes as shown in the following examples. Example a. demonstrates passing DEBUG=On to all MCAs:

a. MCA:
   
   Env01=DEBUG=On

Example b. only passes DEBUG=On to MCAs that service TCPIP channels:

b. Ch1Rule3-ChannelProtocolMatch:
   
   ChannelProtocolMatch=TCPIP \\
   Env01=DEBUG=On

Up to 10 environment variables can be passed and they must be sequentially numbered starting at 1.

Note that these are examples and do not represent a real environment variable. There are also independent and would not be used together.
Process management examples
Part 5. Recovery and problem determination

Chapter 15. Recovery and restart ........................................ 213
  Fault tolerance and recovery ........................................ 213
  Backing up and restoring WebSphere MQ ................................ 214
    Backing up WebSphere MQ ........................................ 214
    Restoring WebSphere MQ ........................................ 214
  Recovery and restart of channel server, execution controller, and queue servers ........................................ 215
    Disaster recovery ................................................ 216
      Configuring WebSphere MQ, NonStop RDF, and AutoSYNC to support disaster recovery ................................ 216
      Restarting operations on the backup system after a disaster ................................................ 218

Chapter 16. Problem determination ........................................ 221
  Preliminary checks ................................................ 221
    Has WebSphere MQ run successfully before? .......................... 221
    Are there any error messages? .................................. 222
    Are there any return codes explaining the problem? ................. 222
    Can you reproduce the problem? ................................ 222
    Have any changes been made since the last successful run? .......... 222
    Has the application run successfully before? ....................... 222
      If the application has not run successfully before .......... 223
      Common programming errors .................................. 223
    Problems with commands ........................................ 224
    Does the problem affect specific parts of the network? .............. 224
    Does the problem occur at specific times of the day? .............. 224
    Is the problem intermittent? .................................. 224
    Have you applied any service updates? ........................... 225
  Looking at problems in more detail ................................... 225
    Have you obtained incorrect output? ................................ 225
      Messages that do not appear on the queue ....................... 225
      Messages that contain unexpected or corrupted information .... 226
      Problems with incorrect output when using distributed queues .......... 227
    Have you failed to receive a response from a PCF command? ........ 228
    Are some of your queues failing? ................................ 229
    Does the problem affect only remote queues? ..................... 229
    Is your application or system running slowly? ..................... 230
  Application design considerations .................................. 230
    Effect of message length ....................................... 230
    Effect of message persistence ................................ 230
    Searching for a particular message .................................. 231
    Queues that contain messages of different lengths ............. 231
    Frequency of syncpoints ....................................... 231
    Use of the MQPUT1 call ........................................ 231
    Number of threads in use ....................................... 231
  Error log files .................................................. 231
  Dead letter queues ................................................. 232

Configuration files and problem determination ................................ 233
  Tracing ........................................................................ 233
  Selecting components to trace ...................................... 233
  A sample trace ....................................................... 233
  Trace files ........................................................... 235
  First failure support technology (FFST) ............................. 235
Chapter 15. Recovery and restart

A messaging system ensures that messages entered into the system are delivered to their destination. A messaging system must also provide a method of tracking the messages in the system and of recovering messages if the system fails for any reason.

WebSphere MQ for HP NonStop Server ensures that persistent messages are not lost by using the NonStop OS transaction manager (TMF). TMF provides transaction protection, queue-file consistency, and queue-file recovery.

WebSphere MQ for HP NonStop Server also uses NonStop process-pair technology to ensure that even nonpersistent messages are resilient to failures. The queue servers that are responsible for the storage of messages take a checkpoint of nonpersistent messages to their backup process running in a different CPU.

Taking a checkpoint of nonpersistent message is an option that you can configure for each queue using the altmqfls command. Taking a checkpoint of nonpersistent messages is enabled by default.

The TMF subsystem manages the complex operations for current transactions and database consistency (for both user operations and WebSphere MQ operations). The TMF subsystem makes these operations transparent to both users and application programs.

A recovery restores the queue manager to the state it was in when the queue manager stopped. Any transactions that are in process are rolled back and any messages that were not committed at the time the queue manager stopped are removed from the queues. Recovery restores all persistent messages. Nonpersistent messages are lost during the process.

The remainder of this chapter introduces the concepts of recovery and restart in more detail and explains how to recover if you experience any problems. It covers the following topics:

- “Fault tolerance and recovery”
- “Backing up and restoring WebSphere MQ” on page 214
- “Recovery and restart of channel server, execution controller, and queue servers” on page 215

Fault tolerance and recovery

WebSphere MQ for NonStop Server can recover from a single point of failure while maintaining data integrity. If any single hardware or software component fails, this failure does not result in loss, duplication, corruption of data, or the permanent loss (that is, requiring outside intervention to restore) of any function of the system if you properly configure WebSphere MQ, NonStop OS, and the NonStop Server hardware. For example, a configuration where all components are configured as redundant or mirrored devices or process-pairs as prescribed by HP.

Repeated consecutive failure (for example, fail-recovery looping) of the same software component is trapped when a configured maximum number of failures is exceeded. In these instances, or in the case of multiple-point failure, WebSphere MQ cannot preserve queue integrity.
Fault tolerance

For more on setting up a queue manager for data integrity and availability, see Chapter 8, “Managing scalability, performance, availability, and data integrity,” on page 85.

Backing up and restoring WebSphere MQ

Periodically, you might want to make a backup of your queue manager’s data to provide protection against possible corruption caused by hardware failures.

Backing up WebSphere MQ

To back up a queue manager’s data, you must:

1. Ensure that the queue manager is not running.

   Note: If you try to make a backup of a running queue manager, the backup might not be consistent because of updates that were in progress when the files were copied.

   If your queue manager is running, stop it using the `endmqm` command.

2. Locate the NonStop OS volumes, subvolumes, and the OSS directories that the queue manager stores its data under.

   You can use the information in the configuration files to determine these directories. For more information, see “Chapter 9, “Configuring WebSphere MQ,” on page 115.

   Note: If you have difficulty understanding the names that appear in the directory, it is because the names are transformed to ensure that they are compatible with the platform on which you are using WebSphere MQ. For more information about name transformations, see “WebSphere MQ file names” on page 18.

3. Make copies of all the queue manager’s data in these OSS directories and NonStop OS subvolumes.

   Ensure that you do not omit any of the files. When you take copies of the data in the NonStop OS file system, ensure that you preserve the structure of the files, the file ownership, and the security settings. Use the HP BACKUP or PAX utilities to make copies of NonStop OS filesystem files and PAX to make copies of OSS filesystem files from the NonStop OS environment. Use tar or pax in the OSS environment to make copies of OSS files.

Restoring WebSphere MQ

To restore a backup of a queue manager’s data, you must:

1. Ensure that the queue manager is not running.

   Note: If you try to restore a running queue manager, the restore might not be consistent because of updates that were in progress when the files were copied.

2. Locate the OSS directories and NonStop OS subvolumes that the queue manager stores its data under. This information is located in the configuration file. See step 2 of “Backing up WebSphere MQ.”

3. Empty the OSS directories and NonStop OS subvolumes that you are going to place the backed up data into.

4. Copy the backed-up queue manager data into the correct places. Use the appropriate command according to which method you used to make copies of the files:
• Use the RESTORE utility if you used the BACKUP utility.
• Use UNPAK if you used PAK.
• Use PAX if you used PAX.

From the OSS environment, use the tar or pax utility as appropriate.

5. Check the resulting directory structure to ensure that you have all of the required directories and that you set the file ownership and access permissions correctly.

6. 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.

Note: You can only restore a queue manager to the same location that it was backed up from.

If you have backed up and restored the data correctly, the queue manager starts.

---

**Recovery and restart of channel server, execution controller, and queue servers**

The channel server, execution controller, and queue server processes are NonStop OS process-pairs. This means that they are designed to continue to provide their services in the event of a failure of a single CPU, or of the process itself. In the case of a single failure, the backup channel server, execution controller, or queue server process takes over as the new primary without interruption of queue manager processing. An example of a single failure is the CPU that contains the primary channel server, execution controller, or queue server process fails, or the primary channel server, execution controller, or queue server process itself fails.

Therefore in single-point-of-failure situations no recovery actions specific to the channel server, execution controller, or queue server are required. A message is logged to the message log file by the channel server, execution controller, or queue server whenever the backup has to be restarted by the primary, or the backup takes over as primary.

In the case of a more serious failure (for example, an environmental failure that prevents initialization of the primary or backup of a NonStop OS process-pair), the Pathway PATHMON process attempts to restart the failed process-pair up to 10 times.

Critical databases accessed by the NonStop OS process-pairs are protected by TMF, so that in the event of failures affecting access to the disks, the protection provided by TMF and the DP2 disk subsystem can be relied on.

Never individually stop and restart the channel server, execution controller, or default queue server using PATHCOM commands. If required, you can stop and restart non-default queue servers provided the objects the queue servers are responsible for are not in use by applications. Do not use the TACL STOP command with the process name parameter to stop both the primary and backup processes of any queue server, channel server, or execution controller.

The `strmqm` command automatically starts all queue server classes that have names that begin with the character string MQS-QUEUE. At queue manager
Recovery and restart of servers and execution controller

shutdown, all active queue server classes coordinate their shutdown. When all active queue manager connections are closed, any queue server involved with those connections shuts down.

If a primary process of a NonStop OS process-pair fails, NonStop OS aborts transactions under certain circumstances. This can cause the failure of MQPUT or MQGET operations in progress at the time of a failure of the primary process. For further information, see Chapter 8, “Managing scalability, performance, availability, and data integrity,” on page 85.

Disaster recovery

HP NonStop Remote Database Facility (NonStop RDF) duplicates all changes to audited databases to a geographically-remote backup system. AutoSYNC handles the duplication and synchronization of all non-audited types of file. Together NonStop RDF and AutoSYNC provide close to real-time database backups. You can use WebSphere MQ with NonStop RDF and AutoSYNC to create a duplicate WebSphere MQ environment on a geographically-remote HP NonStop Server system. This type of configuration allows your infrastructure and applications to be quickly restarted on the remote disaster backup system in the event of a total outage on the primary system, for example, a natural disaster.

NonStop RDF provides near real-time duplication of data records stored in TMF audited files as they are updated, created, or deleted. NonStop RDF uses the TMF audit trail log files to automatically duplicate changes to a queue manager’s audited files to an identical set of queue manager files on the backup system. The queue files, queue overflow files, and other critical database files for a queue manager can therefore be kept up-to-date on the backup site. NonStop RDF cannot duplicate data that is not audited by TMF. You can configure NonStop RDF to provide a variety of levels of protection, including Zero Lost Transactions (ZLT) operation. More typically, a configuration is used that allows a maximum of a few seconds of data loss in the event of a disaster to provide an acceptable level of performance. For detailed information about NonStop RDF refer to the HP NonStop RDF System Management Manual.

AutoSYNC provides scheduled or on-demand replication of modified files, subvolumes, or OSS directory trees to the backup system. AutoSYNC is used to complement NonStop RDF and provide the duplication of changing non-audited data in both the NonStop OS and OSS file systems to the backup system. AutoSYNC recognizes changes to complete files using the file system modification timestamp. Run AutoSYNC in scheduled mode to ensure that the queue manager’s non-audited data is kept as up-to-date as possible.

The following topics are covered in this section:

• “Configuring WebSphere MQ, NonStop RDF, and AutoSYNC to support disaster recovery” on page 218
• “Restarting operations on the backup system after a disaster” on page 218

Configuring WebSphere MQ, NonStop RDF, and AutoSYNC to support disaster recovery

Generally you can use WebSphere MQ with NonStop RDF and AutoSYNC with little or no special configuration. Review the following considerations to see if they apply to your environment:
Disaster recovery

- You must use identical NonStop OS volume names on the backup and primary systems for the WebSphere MQ installation and its queue managers. NonStop RDF does have the capability to duplicate data to different backup volume names, but this is not supported by WebSphere MQ.
- If your queue manager uses message overflow files for large persistent messages, configure the queues that store these messages to use a method that is compatible with the use of NonStop RDF using the setting:
  `altmqfs --qsoptions R`
- You must configure the NonStop RDF global option REPLICATEPURGE as ON so that NonStop RDF correctly duplicates the deletion (purge) of audited files on the backup system. WebSphere MQ does not support NonStop RDF usage with this option set to OFF.
- Ensure that NonStop RDF updater processes are running for all volumes that are used by WebSphere MQ.
- Ensure that there are no extraneous files in the subvolumes used by WebSphere MQ before beginning NonStop RDF operation.
- Create queue managers on the primary site only. NonStop RDF and AutoSYNC then ensure that the correct updates are performed on the backup site.
- Configure AutoSYNC to run in scheduled mode so that it automatically updates the backup site. The default frequency for updating is 5 minutes, which in most cases provides sufficient protection for maintaining the backup data.
- Create different AutoSYNC filesets for each queue manager. Configure multiple filesets for AutoSYNC to include the following collections of files:
  1. The queue manager Pathway configuration file, PATHCTL in the queue manager’s subvolume.
     Although there are other non-audited files in this subvolume that are used at runtime by the queue manager, they are not required for a new instance of the queue manager.
  2. The entire queue manager OSS directory structure `var_installation_path/var/mqm/qmgrs/qmname`.
     You must specify the absolute path name of the queue manager’s directory. Specify the NO ALLSYMFLINKS option for this fileset to prevent AutoSYNC from attempting to synchronize the symbolic link (G directory) in the queue manager’s directory to the NonStop OS queue manager’s subvolume.

There are only two specific files in the `var_installation_path/var/mqm` directory that require synchronization:

1. mqs.ini
2. proc.ini

Like all OSS filesets in AutoSYNC, you must specify the absolute pathname of both files.

There is no need for synchronization of any files in the OSS `opt_installation_path/opt/mqm` tree and in any of the NonStop OS subvolumes for the installation (executables, headers, and samples subvolumes). Even though some non-audited files are created during runtime operation in some of these locations, the files are not needed by a new instance of a queue manager. Do not run a queue manager on the backup system at the same time as its corresponding queue manager is running on the primary system. Results on the backup queue manager can be unpredictable when modifications are performed by RDF on databases that the backup queue manager is using.
Disaster recovery

Restarting operations on the backup system after a disaster

After the NonStop RDF TAKEOVER operation has been completed, you should perform the following operations to ensure that WebSphere MQ is ready to be started and support applications for each queue manager in your backup system configuration:

1. Delete the qmstatus.ini file, if it exists, in the queue manager’s OSS directory.
2. Review the contents of the qmproc.ini file in the queue manager’s OSS directory and ensure that any CPU and process name settings are correct for your backup system. Make a note of the PATHMON process name in the Pathway stanza.
3. In TACL, set your default subvolume to the queue manager’s subvolume and run PATHMON manually using the process name specified in the qmproc.ini file for the queue manager. Then use PATHCOM to load the queue manager’s Pathway configuration data. For example:

   ```
   $SYSTEM.SYSTEM> VOLUME $DATA01.QMTEST1
   $DATA01.QMTEST1> PATHMON /NAME $Z2SC, NOWAIT/
   $DATA01.QMTEST1> PATHCOM $Z2SC
   $Y3RT: PATHMON - T8344D44 - (27SEP04)
   (C)1980 Tandem (C)2003 Hewlett Packard Development Company, L.P.
   19DEC05,17:11 $Z2SC: PATHMON - T8344D44 - (27SEP04)
   19DEC05,17:11 $Z2SC: (C)1980 Tandem (C)2003 Hewlett Packard Development Company, L.P.
   =start pathway cool
   PATHWAY CONTROL FILE DATED: 13 DEC 2005, 22:58:56
   ```

4. Perform a PATHCOM INFO command on all server classes. Then:
   a. Ensure that all instances of the system name appear either as \* or have the correct node name for the backup system.
   b. Verify that the CPUS attribute configured for the server classes are valid for this backup site.
   c. Verify that the HOMETERM and OUT attributes are valid for the backup site.
5. Verify and correct the PATHWAY configuration (if necessary) then save it back to disk using the PATHCOM SHUTDOWN2 command. Exit from PATHCOM.
6. Use the `strmqm` command to start the queue manager.
7. Use the `runmqsc` command to verify the channel configuration and adjust if necessary. Unless you can reconfigure the backup system to use the same network address, you need to change the channel configuration.

Be prepared for channel synchronization or message sequence number errors, particularly if the primary site channels were running at the time of the takeover. NonStop RDF does not ensure that the databases on the backup site are up-to-date (in lockstep with the primary) so some amount of data can be lost as a result of a takeover. To minimize the chances of this, ensure that your RDF configuration and the network between the primary and backup systems can handle the volume of database updates associated with your processing.

If a message sequence number error occurs for a channel, use the MQSC command RESET CHANNEL to set the message sequence number to the value required by the remote queue manager. If a synchronization error occurs for a channel and it is marked as in doubt, you cannot use the MQSC command RESOLVE CHANNEL because there is no synchronization data duplicated to the backup system’s queue managers. Attempt to use the RESOLVE CHANNEL command (if appropriate) on the remote queue manager and the RESET CHANNEL command to bring the
message sequence numbers into alignment. If this method fails, you might need to delete and recreate the channels on one or both queue managers.
Chapter 16. 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 225
- “Application design considerations” on page 230
- “Error log files” on page 231
- “Dead letter queues” on page 232
- “Configuration files and problem determination” on page 233
- “Tracing” on page 233
- “First failure support technology (FFST)” on page 235

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 the WebSphere MQ for HP NonStop Server, V5.3 Quick Beginnings to check that you have installed the product correctly, and run the verification procedure.

Also look at WebSphere MQ Intercommunication for information about configuring WebSphere MQ after installation.
Preliminary checks

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 log files” on page 231 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 installations 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 configuration of TMF?

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.

- 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.
- 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.
Preliminary checks

- 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.
- Failing to remember or realize that, on NonStop OS, by default, the operations of putting a message in a queue and getting a message from a queue are within syncpoint control.
- Failing to specify correctly which TMF transaction is current when implementing a global unit of work, or failing to designate that no TMF transaction is current when implementing a local unit of work.

Problems with commands

Be careful when including special characters, for example, backslash (\) 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.

Be careful when entering a control command at an OSS shell command prompt. The values of the parameters in the command might contain certain characters, such as a dollar sign ($), that are intercepted by the OSS shell and interpreted in a different way. For information about how to enter a control command at an OSS shell command prompt, see Chapter 17, “How to use WebSphere MQ control commands,” on page 239.

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
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 PTF error.
- If a PTF 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 228
- “Are some of your queues failing?” on page 229
- “Does the problem affect only remote queues?” on page 229
- “Is your application or system running slowly?” on page 230

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?
What next

- 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 work has been committed.
  - Is your wait interval long enough?
    You can set the wait interval as an option for the MQGET call. Ensure that you are waiting long enough for a response.
  - Are you waiting for a specific message that is identified by a message or correlation identifier (MsgId or CorrelId)?
    Check that you are waiting for a message with the correct MsgId or CorrelId. A successful MQGET call sets both these values to that of the message retrieved, so you might need to reset these values in order to get another message successfully.
    Also, check whether you can get other messages from the queue.
  - Can other applications get messages from the queue?
    - Was the message you are expecting defined as persistent?
      If not, and WebSphere MQ has been restarted, the message has been lost, unless the NonPersistentMessageClass attribute of the queue is set to MQNPM_CLASS_HIGH.
    - 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.
What next

If the format of a message is not one of the built-in formats, you must use a data conversion exit to convert the message. For more information about data conversion and data conversion exits, see the WebSphere MQ Application Programming Guide.

Have you failed to receive a response from a PCF command?

If you have issued a command but have not received a response, consider the following:

- Is the command server running?
  Work with the dspmqcsv command to check the status of the command server.
  - If the response to this command indicates that the command server is not running, use the strmqcsv command to start it.
  - If the response to the command indicates that the SYSTEM.ADMIN.COMMAND.QUEUE is not enabled for MQGET requests, enable the queue for MQGET requests.

- Has a reply been sent to the dead letter queue?
  The dead letter queue header structure contains a reason or feedback code describing the problem. See the WebSphere MQ Application Programming Reference 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 log files” on page 231 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?
       - Does the **ProcessName** attribute of the queue have the correct value?
         Does a process object with the specified name actually exist? Is the application to service the application queue correctly identified by the process object, and can the application be run?
     - 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. For more detailed information about the handles that are currently open, such as the process and user ID associated with a handle, use the **DISPLAY QSTATUS TYPE(HANDLE)** command to request handle status information.

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.
What next

Is your application or system running slowly?
If your application is running slowly, it might be in a loop or waiting for a resource that is not available.

This might also indicate a performance problem. Perhaps your system is operating near the limits of its capacity. This type of problem is probably worst at peak system load times, typically at mid-morning and mid-afternoon. (If your network extends across more than one time zone, peak system load might seem to occur at some other time.)

A performance problem might be caused by a limitation of your hardware.

If you find that performance degradation is not dependent on system loading, but happens sometimes when the system is lightly loaded, a poorly-designed application program is probably to blame. This could appear to be a problem that only occurs when certain queues are accessed.

The following symptoms might indicate that WebSphere MQ is running slowly:
• Your system is slow to respond to MQSC commands.
• Repeated displays of the queue depth indicate that the queue is being processed slowly for an application with which you would expect a large amount of queue activity.

If the performance of your system is still degraded after reviewing the above possible causes, the problem might lie with WebSphere MQ itself. If you suspect this, contact your IBM Support Center for help.

Application design considerations
There are a number of ways in which poor program design can affect performance. These can be difficult to detect because the program can appear to perform well itself, but affect the performance of other tasks. Several problems specific to programs making WebSphere MQ calls are discussed in the following sections.

For more information about application design, see the WebSphere MQ Application Programming Guide.

Effect of message length
The amount of data in a message can affect the performance of the application that processes the message. To achieve the best performance from your application, send only the essential data in a message. For example, in a request to debit a bank account, the only information that might need to be passed from the client to the server application is the account number and the amount of the debit.

Effect of message persistence
Persistent messages are more reliable than nonpersistent messages but, as a general rule, they use more system resources than nonpersistent messages and do not perform as well. Consider carefully the messages that you design. Use nonpersistent messages wherever they can satisfy the requirements of an application.

For more information about persistent and nonpersistent messages, see “Message persistence” on page 86 and “Persistent and nonpersistent data” on page 100.
Searching for a particular message

The MQGET call usually retrieves the first message from a queue. If you use the message and correlation identifiers (MsgId and CorrelId) in the message descriptor to specify a particular message, the queue manager has to search the queue until it finds that message. Using the MQGET call in this way affects the performance of your application.

Queues that contain messages of different lengths

If the messages on a queue are of different lengths, to determine the size of a message, your application could use the MQGET call with the BufferLength field set to zero so that, even though the call fails, it returns the size of the message data. The application can then repeat the call, specifying the identifier of the message it measured in its first call and a buffer of the correct size. However, if there are other applications serving the same queue, you might find that the performance of your application is reduced because its second MQGET call spends time searching for a message that another application has retrieved in the time between your two calls.

If your application cannot use messages of a fixed length, grow and shrink the buffers dynamically to suit the typical message size. Add code so that, when the application issues an MQGET that fails because the buffer is too small for the message size, you resize the buffer and reissue the MQGET.

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

An application might require a large number of threads. Each queue manager process is allocated a maximum allowable number of 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 log files

WebSphere MQ uses error logs to capture information about errors and significant events. The information in the error logs is national language enabled.

WebSphere MQ writes each error message to an error log file in one of the following directories. The directory selected by WebSphere MQ depends on whether the queue manager name is known when the error or event occurs.
Error log files

- If the queue manager name is known, the error message is written to an error log file in the queue manager's errors directory. Normally, this the directory \var\installation_path\var\mqm\qmgrs\qmname\errors.
- Otherwise, the error message is written to an error log file in the errors subdirectory of the directory identified by the DefaultPrefix entry in the AllQueueManagers stanza in the WebSphere MQ configuration file, mqs.ini. For example, if the default prefix is /usr/ibm/wmq/GA/var/mqm, the error message is written to an error log file in the directory /usr/ibm/wmq/GA/var/mqm/errors.

If WebSphere MQ cannot read the DefaultPrefix entry for any reason, the error message is written to an error log file in the directory \var\installation_path\var\mqm\errors.

Each errors directory can contain up to three error log files with the following names:
- AMQERR01.LOG
- AMQERR02.LOG
- AMQERR03.LOG

When an error message is generated, it is always written to AMQERR01.LOG. When AMQERR01.LOG becomes larger than 256 KB, AMQERR03.LOG is deleted, AMQERR02.LOG is renamed AMQERR03.LOG, AMQERR01.LOG is renamed AMQERR02.LOG, and subsequent error messages are written to a new version of AMQERR01.LOG. Therefore, the latest error messages are always found in AMQERR01.LOG. The other error log files are used to maintain a history of error messages.

Use your usual text editor to examine the contents of an error log file.

Provided you have enabled the facility, WebSphere MQ also writes information about selected errors and events to the Event Management Subsystem (EMS). For more information about EMS event messages, see “Event Management Service (EMS) events” on page 435.

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 (amqscbg) 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 43 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 13, “The WebSphere MQ dead-letter queue handler,” on page 187.
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 directories.

Tracing

This section describes how to produce a trace for WebSphere MQ.

You enable or modify tracing using the strmqtrc control command, which is described in “strmqtrc (start trace)” on page 294. To stop tracing, you use the endmqtrc control command, which is described in “endmqtrc (end trace)” on page 274. You can display formatted trace output using the dspmqtrc control command, which is described in “dspmqtrc (display formatted trace)” on page 267.

The control commands strmqtrc and endmqtrc affect tracing only for those server processes of a queue manager that are running in one specific CPU. By default, this is the same CPU as the one in which your OSS shell or TACL session is running. To enable or end tracing for those server processes of a queue manager that are running in another CPU, you must precede the strmqtrc or endmqtrc command by run -cpu=n at an OSS shell command prompt, or run /CPU n/ at a TACL command prompt, where n is the CPU number. Here is an example of how to enter the strmqtrc command at an OSS shell command prompt:

```
run -cpu=2 strmqtrc -m QM1 -t all
```

This command enables tracing for all components of queue manager QM1 that are running in CPU 2.

Selecting components to trace

Use the -t and -x parameters on the strmqtrc command to control the amount of trace detail to record. By default, all trace points are enabled. You can use the -x parameter to specify the points that you do not want to trace. For example, to output data only for trace points associated with data flowing over communications networks, enter the following command at an OSS shell command prompt:

```
run -cpu=2 strmqtrc -m QM1 -t comms
```

Note, however, that this command enables tracing only for those server processes of queue manager QM1 that are running in CPU 2.

For a full description of the strmqtrc command, see “strmqtrc (start trace)” on page 294.

A sample trace

Figure 25 on page 234 shows an extract from a WebSphere MQ trace:
Figure 25. Sample WebSphere MQ trace
**Trace files**

All trace files are created in the directory `var_installation_path/var/mqm/trace`.

**Note:** You can accommodate the production of large trace files by mounting a temporary file system over this directory.

The names of trace files have the following format:

```
AMQccppppp.TRC
```

where `cc` is the number of the CPU for which the trace has been produced, and `pppppp` is the process identification number (PIN) of the process that produced the trace.

**Notes:**

1. The PIN in the name of a trace file always has five digits and contains leading zeros if necessary, as in the following example: AMQ0300331.TRC.
2. There is one trace file for each process running as part of the components being traced.

---

**First failure support technology (FFST)**

This section describes the role of first failure support technology (FFST) for WebSphere MQ.

FFST information is recorded in files in the directory `var_installation_path/var/mqm/errors`.

These errors are normally severe, unrecoverable errors, and indicate either a configuration problem with the system or a WebSphere MQ internal error.

The files are named `AMQccppppp.nn.FDC`, where:

- `cc` is the number of the CPU in which the process that reported the error was running.
- `pppppp` is the process identification number (PIN) of the process that reported the error.
- `nn` is a sequence number, normally 0.

When a process creates an FFST record, it also sends an event message to EMS using the syslog emulation provided by OSS. The event message contains the name of the FFST file to assist in automatic problem tracking.

Some typical FFST data is shown in [Figure 26 on page 236](#).
The function stack is used by IBM to assist in problem determination. In most cases there is little that the system administrator can do when an FFST report is generated, apart from raising problems through the IBM Support Center.

However, there are some problems that the system administrator might be able to solve. If the FFST report contains descriptions like out of resource or unable to obtain memory when a memory function, such as malloc, calloc, or realloc has been called, it is likely that the virtual memory limit for a CPU has been exceeded. If the FFST report indicates that a NonStop OS file system error has occurred, you can often associate the error with a file name or device name using the description in the FFST report and correct the problem.
Part 6. WebSphere MQ control commands

Chapter 17. How to use WebSphere MQ control commands
- Names of WebSphere MQ objects
- How to read syntax diagrams
- Example syntax diagram
- Syntax help
- Examples

Chapter 18. The control commands
- altmqfls (alter WebSphere MQ object attributes)
- altmqusr (alter WebSphere MQ user information)
- crtmqcvx (create code for data conversion exit)
- crtmqm (create queue manager)
- dltmqm (delete queue manager)
- dmpmqaut (dump authority)
- dspmq (display queue managers)
- dspmqaut (display authority)
- dspmqcsv (display command server)
- dspmqfls (display WebSphere MQ object attributes)
- dspmqtrc (display formatted trace)
- dspmque (display WebSphere MQ user information)
- endmqcsv (end command server)
- endmlsr (end listener)
- endmqm (end queue manager)
- endmqtrc (end trace)
- runmqchi (run channel initiator)
- runmqchl (run channel)
- runmqdlq (run dead letter queue handler)
- runmlsr (run listener)
- runmqsc (run MQSC commands)
- runmqrm (start trigger monitor)
- setmqaut (grant or revoke authority)
- strmqcsv (start command server)
- strmqm (start queue manager)
- strmqtrc (start trace)
Chapter 17. How to use WebSphere MQ control commands

This chapter describes how to use 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 143.

On NonStop OS, you can enter a control command at an OSS shell command prompt or a TACL command prompt. At an OSS shell command prompt, the name of the command is case sensitive but, at a TACL command prompt, the name of the command is not case sensitive. At either command prompt, the parameters of the command are case sensitive.

If you enter a control command at an OSS shell command prompt, the values of the parameters in the command might contain certain characters that are intercepted by the OSS shell and interpreted in a different way. These characters include the dollar sign ($), the asterisk (*), the question mark (?), the left parenthesis, and the right parenthesis. To prevent these characters from being intercepted by the OSS shell, precede each occurrence of these characters by a backslash (\), which acts as an escape character, or enclose any value containing these characters within single quotes.

Here are some examples of control commands entered at an OSS shell command prompt:

```
altmqfls --qmgr QMGR --type QLOCAL --qsize "(100,200,500,200,200,100)" TEST.QUEUE
altmqfls --qmgr QMGR --type QLOCAL --qsize "'(100,200,500,200,200,100)'" TEST.QUEUE
crtmqm -ns "$DATA01.ACCOUNTS JUPITER"
dspmqfls -m SAMPLE_QMGR -t prcs \
dspmqfls -m SAMPLE_QMGR 'SYSTEM.ADMIN*' runmqlsr -t tcp -p 1822 -g "$ZTC4 -m GANYMEDE"
setmqaut -m qmgr1 -n 'a.b.*' -t q -p user1 +all
```

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
- Authentication information objects

The maximum length of 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 (_) (see note 1)
- Percent sign (%) (see note 1)

© Copyright IBM Corp. 2006 239
Names

Notes:
1. Forward slash and percent are special characters. If you use either of these characters in a name, the name must be enclosed in double quotation marks whenever it is used.
2. Leading or embedded blanks are not allowed.
3. National language characters are not allowed.
4. Names 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 21. How to read syntax diagrams

<table>
<thead>
<tr>
<th>Convention</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Diagram" /></td>
<td>You must specify the values A, B, and C. Required values are shown on the main line in a syntax diagram.</td>
</tr>
<tr>
<td><img src="image2" alt="Diagram" /></td>
<td>You can specify the value A. Optional values are shown below the main line in a syntax diagram.</td>
</tr>
<tr>
<td><img src="image3" alt="Diagram" /></td>
<td>Values A, B, and C are alternatives, one of which you must specify.</td>
</tr>
<tr>
<td><img src="image4" alt="Diagram" /></td>
<td>Values A, B, and C are alternatives, one of which you can optionally specify.</td>
</tr>
<tr>
<td><img src="image5" alt="Diagram" /></td>
<td>You must specify one or more of the values A, B, and C. Any required separator for more than one value (in this example, the comma (,)) is shown on the arrow.</td>
</tr>
<tr>
<td><img src="image6" alt="Diagram" /></td>
<td>You can specify zero or more of the values A, B, and C. Any required separator for more than one value (in this example, the comma (,)) is shown on the arrow.</td>
</tr>
</tbody>
</table>
Table 21. How to read syntax diagrams (continued)

<table>
<thead>
<tr>
<th>Convention</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Diagram" /></td>
<td>You can specify the value A multiple times. The separator in this example is optional.</td>
</tr>
<tr>
<td><img src="image2" alt="Diagram" /></td>
<td>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.</td>
</tr>
<tr>
<td><img src="image3" alt="Diagram" /></td>
<td>The syntax fragment Name is shown separately from the main syntax diagram.</td>
</tr>
<tr>
<td><img src="image4" alt="Diagram" /></td>
<td>Punctuation and uppercase values Specify exactly as shown.</td>
</tr>
</tbody>
</table>

Example syntax diagram

Here is an example syntax diagram that describes the `hello` command:

**Hello Command**

```
Hello

- hello
  - Name
  - Greeting
```

**Name**

```
- name
  - (1)
```

**Greeting**

```
- how are you?
```

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
```
Syntax diagrams

hello name, name
hello name, name, name
hello, how are you?
hello name, how are you?
hello name, name, how are you?
hello name, name, name, how are you?

The space before the name value is significant, 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
Chapter 18. The control commands

This chapter provides reference information for each of the following WebSphere MQ control commands:

<table>
<thead>
<tr>
<th>Command name</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>altmqfls</td>
<td>Alter WebSphere MQ object attributes</td>
</tr>
<tr>
<td>altmqsur</td>
<td>Alter WebSphere MQ user information</td>
</tr>
<tr>
<td>crtmqcvx</td>
<td>Create a code fragment for a data conversion exit</td>
</tr>
<tr>
<td>crtmqm</td>
<td>Create a queue manager</td>
</tr>
<tr>
<td>dltmqm</td>
<td>Delete a queue manager</td>
</tr>
<tr>
<td>dmpmqaut</td>
<td>Dump authorizations to an object</td>
</tr>
<tr>
<td>dspmq</td>
<td>Display queue managers</td>
</tr>
<tr>
<td>dspmqaut</td>
<td>Display authorizations to an object</td>
</tr>
<tr>
<td>dspmqcsv</td>
<td>Display the status of the command server</td>
</tr>
<tr>
<td>dspmqfls</td>
<td>Display WebSphere MQ object attributes</td>
</tr>
<tr>
<td>dspmqtrc</td>
<td>Display formatted trace output</td>
</tr>
<tr>
<td>dmpmqusr</td>
<td>Display WebSphere MQ user information</td>
</tr>
<tr>
<td>endmqcsv</td>
<td>Stop the command server</td>
</tr>
<tr>
<td>endmqm</td>
<td>Stop all listeners for a queue manager</td>
</tr>
<tr>
<td>endmqm</td>
<td>Stop a queue manager</td>
</tr>
<tr>
<td>endmqtrc</td>
<td>Stop tracing for an entity</td>
</tr>
<tr>
<td>runmqchi</td>
<td>Start a channel initiator</td>
</tr>
<tr>
<td>runmqchl</td>
<td>Start a channel</td>
</tr>
<tr>
<td>runmqdlq</td>
<td>Start the dead letter queue handler</td>
</tr>
<tr>
<td>runmqlsr</td>
<td>Start a listener</td>
</tr>
<tr>
<td>runmqsc</td>
<td>Issue MQSC commands</td>
</tr>
<tr>
<td>runmqtrm</td>
<td>Start a trigger monitor</td>
</tr>
<tr>
<td>setmqaut</td>
<td>Grant or revoke authority</td>
</tr>
<tr>
<td>strmqcsv</td>
<td>Start the command server</td>
</tr>
<tr>
<td>strmqm</td>
<td>Start a queue manager</td>
</tr>
<tr>
<td>strmqtrc</td>
<td>Enable tracing</td>
</tr>
</tbody>
</table>
altmqfls

altmqfls (alter WebSphere MQ object attributes)

Purpose

You use the altmqfls command to alter those attributes of a WebSphere MQ object that are specific to WebSphere MQ for HP NonStop Server and control features that are implemented only on the NonStop OS platform.

If you use the altmqfls command to alter the attributes of a local queue, the command can perform only one of the following three groups of operations:

- Move the files that belong to the local queue to a different volume.
- Change the size of the queue file and queue overflow file associated with the local queue.
- Change the queue server options for the local queue. Using these options, you can optimize the way the queue server handles the storage for the queue, and control the checkpointing of nonpersistent messages. Within this group, you can also change the queue server that manages the queue, and associate a Measure counter with the queue.

You must not use the altmqfls command to alter the attributes of a WebSphere MQ object if an application has the object open. In addition, if the object is a local queue, the queue must not contain uncommitted messages.

Syntax

```
altmqfls --type ObjectType
   --qmgr QMgrName

   --server ServerName
      --qsoptions C L R S none

   --browse MemBrowsePM
      --oflowsize MsgOverflowThresh

   --volume VolumeName
      --msgofsubvol MsgOvflSubvol

   --meascount MeasureCounter
      --resetmeasure

   --qsize (QPriExt, QSecExt, QMaxExt, OPriExt, OSecExt, OMaxExt)

   --maintain ON OFF ObjectName
```
Required parameters

ObjectName
The name of the WebSphere MQ object.

--type ObjectType
The type of the WebSphere MQ object, which can be one of the following types:
- ql or qlocal  A local queue
- qa or qalias  An alias queue
- qr or qremote  A remote queue
- qm or qmodel  A model queue
- proc or process  A process
- nl or namelist  A namelist

Optional parameters

You can use any of the following optional parameters if the WebSphere MQ object is a local queue. If the WebSphere MQ object is not a local queue, you can use only the --qmgr and --server parameters.

--qmgr QMgrName
The name of the queue manager to which the WebSphere MQ object belongs.
The queue manager must have been started. If no queue manager name is specified, the default queue manager is used.

--server ServerName
The symbolic name of a queue server that is to be responsible for the WebSphere MQ object.

When an object is created, it is assigned to the default queue server, which has the symbolic name DEFAULTTQS. When the responsibility for a queue is changed, all nonpersistent messages in the queue are discarded during the change.

--volume VolumeName
The name of the volume to which the files that are associated with the local queue are to be moved.

This parameter can be specified only with the ObjectName, --type ObjectType, and --qmgr QMgrName parameters. It is not allowed in combination with any other parameters.

--qoptions CLRS | none
The queue server options, C, L, R, and S, which you can use to fine tune the performance and data integrity characteristics of the local queue.

Every time you use the altmqfls command with this parameter, each of the options C, L, R, and S is either set or left not set. For example, using the parameter --qoptions S sets the S option, but leaves the C, L, and R options not set. You can use the --qoptions parameter once and once only in a command. You must specify at least one of the options C, L, R, and S, or you can specify none, which means that none of the options C, L, R, and S are to be left set.

The queue server options have the following meanings:

C  Nonpersistent messages are checkpointed to the backup queue server. This provides fault tolerance at the expense of the additional CPU cycles required to handle the checkpointing, the additional IPC
messages, and the additional memory required to store the messages. Use this option if you want a high degree of recoverability for nonpersistent messages.

When a queue is created, this option is set by default (that is, nonpersistent messages are checkpointed).

L The queue server locks in memory the data structures and chains associated with the queue. Normally, the storage associated with a queue is a candidate for removal from the queue server's address space when it is no longer being accessed. Use this option for faster access to the queue's memory data structures at the possible expense of other queues.

When a queue is created, this option is not set by default (that is, the data structures and chains are not locked in memory).

R Message overflow files for the queue are audited by TMF, which means that operations on their contents can be replicated to a backup system using RDF.

When a queue is created, this option is not set by default (that is, message overflow files are not audited).

S The queue server loads the local queue from disk into cache when the queue manager is started up. Normally the messages for a queue are loaded when first referenced by an application. If this option is set, the queue is loaded when the queue manager starts. Use this option to reduce CPU and disk I/O activity when a queue is first opened, at the cost of an increase in queue server activity (CPU and disk I/O) during queue manager startup.

When a queue is created, this option is not set by default (that is, the queue is not loaded on startup).

none None of the options C, L, R, and S are to be left set.

--browse MemBrowsePM
The maximum number of bytes of data of each persistent message to keep in the queue server's cache (as well as on disk). During a browse operation on a persistent message, the queue manager normally reads the data for a message from disk storage and returns it to the application. If this parameter is set to a value other than zero, the specified number of bytes of data will also be kept in memory and the browse operation will return this data to the application without having to access the disk. By using this parameter, you can increase the memory resources in use by the queue server. The minimum value of this parameter is 0 bytes, and the maximum value is 25000 bytes. The default for this parameter when a queue is created is 0 bytes.

--oflowsize MsgOverflowThresh
The minimum message size for the use of a message overflow file to store the message data. Persistent messages that are smaller than this threshold are stored in the queue overflow file. Persistent messages of the threshold size or larger will have their bulk data stored in a dedicated message overflow file. The default for this parameter when a queue is created is 200000 bytes.

--msgosubvol MsgOvlSubvol
The subvolume where the queue server creates new message overflow files. All queues initially use the queue manager's subvolume by default.

The value of this parameter must be the fully qualified local name of the subvolume with the format volume_name.subvolume_name.
--meascount MeasureCounter
The name of a Measure counter, which, if part of an active measurement, is initialized to the current depth of the local queue, and then incremented and decremented by the queue server responsible for the queue when messages are added and removed.

The counter defined in Measure must be an ACCUM type. To view the value of the counter as an absolute number, as opposed to a percentage, use the MEASCOM command SET REPORT RATE OFF.

Start the measurement before any application opens the queue. The counter is initialized when the first application opens the queue, or whenever you enter the altmqfls command with the --resetmeasure parameter.

--resetmeasure
Resets the Measure counter to the current depth of the queue.

--qsize (QPriExt,QSecExt,QMaxExt,OPriExt,OSecExt,OMaxExt)
Properties of the file extents of the queue file and queue overflow file associated with the local queue. You must specify all six properties whenever you use this parameter. The size of an extent is expressed in pages.

QPriExt The size of the primary extent of the queue file
QSecExt The size of a secondary extent for the queue file
QMaxExt The maximum number of extents for the queue file
OPriExt The size of the primary extent of the queue overflow file
OSecExt The size of a secondary extent for the queue overflow file
OMaxExt The maximum number of extents for the queue overflow file

--maintain ON|OFF
Use this parameter to prevent applications from opening the local queue while you perform certain maintenance operations on the queue, such as changing the queue server responsible for the queue, or partitioning the queue file and queue overflow file associated with the queue.

ON Puts the queue into maintenance mode and prevents applications from opening the queue. If an application tries to open the queue while it is in maintenance mode, the MQOPEN call fails with reason code MQRC_UNKNOWN_OBJECT_NAME (2085).

When you put a queue into maintenance mode, all nonpersistent messages in the queue are discarded. You cannot put a queue into maintenance mode while an application still has the queue open.

OFF Takes the queue out of maintenance mode so that applications can open the queue again.

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, entered at an OSS shell command prompt, moves the files associated with the local queue flint.queue to the volume $DATA03. The local queue is owned by queue manager target.queue.mgr.
    altmqfls --qmgr target.queue.mgr --type ql --volume $DATA03 flint.queue
altmqfls

The following command, entered at a TACL command prompt, performs the same function:

```bash
altmqfls --qmgr target.queue.mgr --type ql --volume $DATA03 flint.queue
```

2. The following command changes the queue server for the local queue flint.queue, which is owned by queue manager target.queue.mgr. The symbolic name of the new queue server is ALTERNATE.

```bash
altmqfls --qmgr target.queue.mgr --type ql --server ALTERNATE flint.queue
```

**Related commands**

- `dspmqfls`  
  Display file names
**altmqusr** (alter WebSphere MQ user information)

**Purpose**

Use the **altmqusr** command to create, alter, or delete an entry in the principal database of a queue manager. Each entry maps a WebSphere MQ principal to a NonStop OS user ID.

**Syntax**

```
altmqusr -m QMgrName -p Principal -u NonStopOSUserlD 
```

**Required parameters**

- `-m QMgrName`
  The name of the queue manager whose principal database is to be updated.

- `-p Principal`
  The WebSphere MQ principal whose entry in the principal database is to be created, altered, or deleted.

**Optional parameters**

- `-u NonStopOSUserlD`
  A NonStop OS user ID or Safeguard alias. If the WebSphere MQ principal does not already have an entry in the principal database, a new entry is created to map the principal to the specified NonStop OS user ID, or to the NonStop OS user ID corresponding to the specified Safeguard alias. If the principal already has an entry, the entry is altered so that the principal maps to the new NonStop OS user ID.

  Note that Safeguard aliases are not stored in the principal database. If you specify a Safeguard alias for this parameter, only the corresponding NonStop OS user ID is stored in the principal database.

- `-r`
  The entry for the WebSphere MQ principal is deleted from the principal database.

**Return codes**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Successful operation</td>
</tr>
<tr>
<td>36</td>
<td>Invalid arguments supplied</td>
</tr>
<tr>
<td>69</td>
<td>Storage not available</td>
</tr>
<tr>
<td>71</td>
<td>Unexpected error</td>
</tr>
</tbody>
</table>

**Examples**

1. The following command maps the WebSphere MQ principal mquser1 to the NonStop OS user ID MQTEST.FRED:

   ```
   altmqusr -m MT02 -p mquser1 -u MQTEST.FRED
   ```

2. The following command maps the WebSphere MQ principal mquser2 to the NonStop OS user ID for which user01 is a Safeguard alias:

   ```
   altmqusr -m MT02 -p mquser2 -u user01
   ```

3. The following command deletes the entry for the WebSphere MQ principal mquser1 from the principal database:
Related commands

dspmqusr  Display WebSphere MQ user information
crtmqcvx (create code for data conversion exit)

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

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Command completed normally</td>
</tr>
<tr>
<td>10</td>
<td>Command completed with unexpected results</td>
</tr>
<tr>
<td>20</td>
<td>An error occurred during processing</td>
</tr>
</tbody>
</table>

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:

```c
/* 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`.

Chapter 18. The control commands 251
MQLONG Convertmy_structure(
    PMQBYTE *in_cursor,
    PMQBYTE *out_cursor,
    PMQBYTE in_lastbyte,
    PMQBYTE out_lastbyte,
    MQCONN 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 queue manager and define the system and default objects. The objects created by `crtmqm` are listed in Appendix A, “System and default objects,” on page 427. When a queue manager has been created, use the `strmqm` command to start it.

Syntax

```
crtmqm [-c Text] [-d DefaultTransmissionQueue] [-q]
     [-t IntervalValue] [-u DeadLetterQueue] [-z]
     [-ns QueueManagerSubvolume] [-np PathmonProcessName]
     [-ne ECProcessName] [-nq QueueServerProcessName]
     [-nc ChannelServerProcessName] [-nh HomeTerminal] [-nu CPUString]
```

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.

- **-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 27.

- **-t IntervalValue**
  The trigger time interval in milliseconds for all queues controlled by this queue
manager. This value specifies the time after receiving a trigger-generating message when triggering is suspended. That is, if the arrival of a message on a queue causes a trigger message to be put on the initiation queue, any message arriving on the same queue within the specified interval does not generate another trigger message.

You can use the trigger time interval to ensure that your application is allowed sufficient time to deal with a trigger condition before it is alerted to deal with another on the same queue. You might choose to see all trigger events that happen; if so, set a low or zero value in this field.

Specify a value in the range 0 through 999 999 999. The default is 999 999 999 milliseconds, a time of more than 11 days. Allowing the default to be used effectively means that triggering is disabled after the first trigger message. However, an application can enable triggering again by servicing the queue using a command to alter the queue to reset the trigger attribute.

-u DeadLetterQueue

The name of the local queue that is to be used as the dead-letter (undelivered-message) queue. Messages are put on this queue if they cannot be routed to their correct destination.

The default is no dead-letter queue.

-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.

-ns QueueManagerSubvolume

The subvolume to be used for storing the NonStop OS files of the queue manager. This parameter can have one of the following values:

- The name of a volume. In this case, the command generates a suitable subvolume name.
- The fully qualified local name of a subvolume with the format volume_name.subvolume_name. If the subvolume identified in this way already contains files, the command fails.

During installation, you are asked to select a default volume for all queue managers. The name of the default volume is recorded in the HPNSSQMDefaultGuardianVol entry in the AllQueueManagers stanza in the WebSphere MQ configuration file (mqs.ini) for the installation. If you omit this parameter when creating a queue manager, the queue manager uses the default volume, and the command generates a suitable subvolume name. If you include this parameter, the queue manager uses the specified volume instead.

-np PathmonProcessName

The process name to be used for the PATHMON process in the queue manager’s Pathway environment. This must be a valid NonStop OS process name. If you do not specify this parameter, the operating system assigns a process name.

-ne ECPProcessName

The process name to be used for the execution controller. This must be a valid process name. If you do not specify this parameter, the operating system assigns a process name.
-ng QueueServerProcessName
   The process name to be used for the default queue server. This must be a valid NonStop OS process name. If you do not specify this parameter, the operating system assigns a process name.

-nc ChannelServerProcessName
   The process name to be used for the channel server. This must be a valid NonStop OS process name. If you do not specify this parameter, the operating system assigns a process name.

-nh HomeTerminal
   The home terminal to be used by the server processes of the queue manager. You cannot start the queue manager unless the home terminal is available, which means that it must exist and the server processes of the queue manager must be able to write to it. The default value is $ZHOME.

-nu CPUSString
   A list of the CPUs in which the server processes of the queue manager can run. The permissible values of the parameter are the same as those of the CPUS attribute of a server class in Pathway. The default value is (0:1).

Return codes

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Queue manager created</td>
</tr>
<tr>
<td>8</td>
<td>Queue manager already exists</td>
</tr>
<tr>
<td>49</td>
<td>Queue manager stopping</td>
</tr>
<tr>
<td>69</td>
<td>Storage not available</td>
</tr>
<tr>
<td>70</td>
<td>Queue space not available</td>
</tr>
<tr>
<td>71</td>
<td>Unexpected error</td>
</tr>
<tr>
<td>72</td>
<td>Queue manager name error</td>
</tr>
<tr>
<td>111</td>
<td>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.</td>
</tr>
</tbody>
</table>

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:
  .crtmqm -c "Paint shop" -q Paint.queue.manager

2. 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

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dltmqm</td>
<td>Delete queue manager</td>
</tr>
<tr>
<td>endmqm</td>
<td>End queue manager</td>
</tr>
<tr>
<td>strmqm</td>
<td>Start queue manager</td>
</tr>
</tbody>
</table>
dltmqm

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.

Syntax

```
  dltmqm -z QMgrName
```

Required parameters

- **QMgrName**
  The name of the queue manager to delete.

Optional parameters

- **-z** Suppresses error messages.

Return codes

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Queue manager deleted</td>
</tr>
<tr>
<td>3</td>
<td>Queue manager being created</td>
</tr>
<tr>
<td>5</td>
<td>Queue manager running</td>
</tr>
<tr>
<td>16</td>
<td>Queue manager does not exist</td>
</tr>
<tr>
<td>49</td>
<td>Queue manager stopping</td>
</tr>
<tr>
<td>69</td>
<td>Storage not available</td>
</tr>
<tr>
<td>71</td>
<td>Unexpected error</td>
</tr>
<tr>
<td>72</td>
<td>Queue manager name error</td>
</tr>
<tr>
<td>112</td>
<td>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.</td>
</tr>
</tbody>
</table>

Examples

1. The following command deletes the queue manager saturn.queue.manager.
   
   ```
   dltmqm saturn.queue.manager
   ```

2. The following command deletes the queue manager called travel and also suppresses any messages caused by the command.
   
   ```
   dltmqm -z travel
   ```

Related commands

- **crtmqm** Create queue manager
- **endmqm** End queue manager
- **strmqm** Start queue manager
dmpmqaut (dump authority)

Purpose

Use the dmpmqaut command to dump the current authorizations that match the specified criteria.

Syntax

```
  dmpmqaut -m QMgrName -n Profile -t ObjectType
             -l -s ServiceComponent -g GroupName
```

Optional parameters

- **-m QMgrName**
  
  Dump authority records for the specified queue manager. If you omit this parameter, the 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 151.

- **-l**
  
  Dump only a list of all defined profile names and their associated object types.

- **-t ObjectType**
  
  The type of object for which to dump authorizations. Possible values are:

  - `queue` or `q` A queue
  - `qmgr` The queue manager object
  - `process` or `prcs` A process
  - `namelist` or `nl` A namelist

- **-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.

  For more information about authorization service components, see "Installable services" on page 138, Chapter 19, "Installable services and components," on page 303, and Chapter 20, "Authorization service," on page 311.

- **-g GroupName**
  
  The name of the user group for which to dump authorizations. You can specify only one user group.

Examples

1. This example dumps all authority records with a profile that matches queue a.b.c for the group called groupa:

   ```
   dmpmqaut -m qm1 -n a.b.c -t q -g groupa
   ```
The resulting dump looks something like this:

profile: a.b.*
object type: queue
entity: groupa
type: group
authority: get, browse, put, inq

Related commands

dspmqa
Display authority

setmqaut
Set or reset authority
**Purpose**

Use the `dspmq` command to display the names and operational status of all the queue managers in an installation. The installation is determined by the environment variables MQNSKOPTPATH and MQNSKVARPATH.

**Syntax**

```
dsmpmq -m QMgrName -s
```

**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`
  
  Requests 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
Purpose

Use the dspmqaut command to display the current authorizations for a specified WebSphere MQ object.

Syntax

```
   dspmqaut -n ObjectName -t ObjectType
   -p PrincipalName
   -g GroupName
   -s ServiceComponent
```

Required parameters

- **-n ObjectName**
  The name of a queue, process, or namelist on which to make the inquiry.
  You must include this parameter unless the inquiry relates to the queue manager object, in which case you must omit it.

- **-t ObjectType**
  The type of object on which to make the inquiry. Possible values are:
  - `queue` or `q`
    A queue
  - `qmgr`
    The queue manager object
  - `process` or `prcs`
    A process
  - `namelist` or `nl`
    A namelist

Optional parameters

- **-m QMgrName**
  The name of the queue manager on which to make the inquiry. This parameter is optional if you are inquiring on the authorizations of your default queue manager.

- **-p Principal**
  The WebSphere MQ principal for which to display authorizations to the specified object. If the NonStop OS user ID to which the principal maps is member of more than one user group, the command displays the combined authorizations of all the user groups. You can specify only one principal.

- **-g GroupName**
  The name of the user group for which to display authorizations to the specified object. You can specify only one user group.

- **-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.

For more information about authorization service components, see "Installable services" on page 138, Chapter 19, "Installable services and components," on page 303, and Chapter 20, "Authorization service," on page 311.
Results

The command displays all the authorities that the specified principal or user group has against the specified object.

Table 22 lists all the authorities and shows which authorities are applicable to each type of object.

<table>
<thead>
<tr>
<th>Authority</th>
<th>Queue</th>
<th>Process</th>
<th>Queue manager</th>
<th>Namelist</th>
</tr>
</thead>
<tbody>
<tr>
<td>all</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>alladm</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>allmqi</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>altusr</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>browse</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>chg</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>clr</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>connect</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>crt</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>dlt</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>dsp</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>get</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>inq</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>passall</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>passid</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>put</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>set</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>setall</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>setid</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

The meaning of each authority is explained in the following list:

all       Use all operations relevant to the object.
alladm    Perform all administration operations relevant to the object.
allmqi    Use all MQI calls relevant to the object.
altusr    Specify an alternate user ID on an MQI call.
browse    Retrieve a message from a queue by issuing an MQGET call with the BROWSE option.
chg       Change the attributes of the specified object, using the appropriate command set.
clr       Clear a queue (PCF command Clear queue only).
connect   Connect the application to the specified queue manager by issuing an MQCONN call.
crt       Create objects of the specified type using the appropriate command set.
dlt       Delete the specified object using the appropriate command set.
dsp       Display the attributes of the specified object using the appropriate command set.
get       Retrieve a message from a queue by issuing an MQGET call.
inq      Make an inquiry on a specific queue by issuing an MQINQ call.
passall  Pass all context.
passid   Pass the identity context.
put      Put a message on a specific queue by issuing an MQPUT call.
set      Set attributes on a queue from the MQI by issuing an MQSET call.
setall   Set all context on a queue.
setid    Set the identity context on a queue.

The authorities for administration operations apply to these sets of commands:
• Control commands
• MQSC commands
• PCF commands

Return codes

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Successful operation</td>
</tr>
<tr>
<td>36</td>
<td>Invalid arguments supplied</td>
</tr>
<tr>
<td>40</td>
<td>Queue manager not available</td>
</tr>
<tr>
<td>49</td>
<td>Queue manager stopping</td>
</tr>
<tr>
<td>69</td>
<td>Storage not available</td>
</tr>
<tr>
<td>71</td>
<td>Unexpected error</td>
</tr>
<tr>
<td>72</td>
<td>Queue manager name error</td>
</tr>
<tr>
<td>133</td>
<td>Unknown object name</td>
</tr>
<tr>
<td>145</td>
<td>Unexpected object name</td>
</tr>
<tr>
<td>146</td>
<td>Object name missing</td>
</tr>
<tr>
<td>147</td>
<td>Object type missing</td>
</tr>
<tr>
<td>148</td>
<td>Invalid object type</td>
</tr>
<tr>
<td>149</td>
<td>Entity name missing</td>
</tr>
</tbody>
</table>

Examples

1. The following command displays the authorities that user group MQM has
against the queue manager object of queue manager saturn.queue.manager:

dspmqaut -m saturn.queue.manager -t qmgr -g staff

The results from this command are:
Entity staff has the following authorizations for object:
inq
     set
     connect
     altusr
     crt
     dlt
     chg
     dsp
     setid
     setall

2. The following command displays the authorities that principal user1 has
against the queue a.b.c:

dspmqaut -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
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

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Command completed normally</td>
</tr>
<tr>
<td>10</td>
<td>Command completed with unexpected results</td>
</tr>
<tr>
<td>20</td>
<td>An error occurred during processing</td>
</tr>
</tbody>
</table>

Examples

The following command displays the status of the command server associated with `venus.q.mgr`:
```
dspmqcsv venus.q.mgr
```

Related commands

- `endmqcsv` — End command server
- `strmqcsv` — Start command server
**dspmqfls**

---

**dspmqfls (display WebSphere MQ object attributes)**

**Purpose**

Use the `dspmqfls` command to display the real file system names 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 18 for information about name transformation.

The `dspmqfls` command also displays those attributes of the WebSphere MQ objects that are specific to WebSphere MQ for HP NonStop Server and control features that are implemented only on the NonStop OS platform. These are the attributes set by the `altmqfls` command. For a local queue, `dspmqfls` also displays the names of the queue file and the queue overflow file, but it does not display the names of any message overflow files.

**Syntax**

```
   -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, which is the default
- `q` or `queue` A queue
- `ql` or `qlocal` A local queue
- `qa` or `qalias` An alias queue
- `qr` or `qremote` A remote queue
- `qm` or `qmodel` A model queue
- `qmgr` A queue manager object
- `prcs` or `process` A process
- `ctlg` or `catalog` An object catalog
- `nl` or `namelist` A namelist
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, entered at an OSS shell command prompt, displays the information for all WebSphere MQ objects whose names begin with SYSTEM.ADMIN and are owned by the queue manager SAMPLE_QMGR:

   dspmqfls -m SAMPLE_QMGR 'SYSTEM.ADMIN*'

   The following command, entered at a TACL command prompt, performs the same function:

   dspmqfls -m SAMPLE_QMGR SYSTEM.ADMIN*

   Each command generates the following output:

   WebSphere MQ Display MQ Files

   CATALOGUE Object Catalogue
   $DATA01.SAMPLEXQ.AMQCAT

   QMGR Queue Manager Object
   $DATA01.SAMPLEXQ.TSAMPLEX

   Queue/Status Server : DEFAULTQS

   QLOCAL SYSTEM.ADMIN.CHANNEL.EVENT
   $DATA01.SAMPLEXQ.QHMY02E2
   $DATA01.SAMPLEXQ.QHMY02E2
   $DATA01.SAMPLEXQ.THMY02E2

   Queue/Status Server : DEFAULTQS
   Queue Server Options : C
   Message overflow threshold : 200000 bytes

   QLOCAL SYSTEM.ADMIN.COMMAND.QUEUE
   $DATA01.SAMPLEXQ.QIK3VVY1
   $DATA01.SAMPLEXQ.QIK3VVY1
   $DATA01.SAMPLEXQ.TIK3VVY1

   Queue/Status Server : DEFAULTQS
   Queue Server Options : C
   Message overflow threshold : 200000 bytes

   QLOCAL SYSTEM.ADMIN.PERFM.EVENT
   $DATA01.SAMPLEXQ.QWDJPV01
   $DATA01.SAMPLEXQ.QWDJPV01
   $DATA01.SAMPLEXQ.TWDJPV01

   Queue/Status Server : DEFAULTQS
   Queue Server Options : C
   Message overflow threshold : 200000 bytes

   QLOCAL SYSTEMADMIN.QMGR.EVENT
   $DATA01.SAMPLEXQ.QSIB1KO
   $DATA01.SAMPLEXQ.QSIB1KO
   $DATA01.SAMPLEXQ.TSIB1KO

   Queue/Status Server : DEFAULTQS
   Queue Server Options : C
   Message overflow threshold : 200000 bytes
2. The following command, entered at an OSS shell command prompt, displays the information for all processes owned by the queue manager SAMPLE_QMGR:

```
dspmqfls -m SAMPLE_QMGR -t prcs *
```

The following command, entered at a TACL command prompt, performs the same function:

```
dspmqfls -m SAMPLE_QMGR -t prcs *
```

Each command generates the following output:

```
WebSphere MQ Display MQ Files

PROCESS     SYSTEM.DEFAULT.PROCESS
            $DATA01.SAMPLEXQ.TKF5FQU

Queue/Status Server : DEFAULTQS
```
dspmqtrc (display formatted trace)

Purpose
Use the dspmqtrc command to display WebSphere MQ formatted trace output.

Syntax
```
-dspmqtrc -t FormatTemplate -h -s -o OutputFilename
```

Required parameters

- `InputFileName`
  The name of the file containing the unformatted trace; for example, `var_installation_path/var/mqm(trace)/AMQ1200345.TRC`. If you provide one input file, dspmqtrc formats it either to stdout or to the output file you name. If you provide more than one input file, any output file name you specify is ignored, and each output file has the same name as the corresponding input file, but with an extension of .FMT instead of .TRC.

Optional parameters

- `-t FormatTemplate`
  The name of the template file containing details of how to display the trace. The default value is `opt_installation_path/opt/mqm/lib/amqtrc.fmt`.

- `-h`
  Omit header information from the report.

- `-s`
  Extract trace header and put to stdout.

- `-o output_filename`
  The name of the file into which to write formatted data.

Related commands
```
endmqtrc    End trace
strmqtrc    Start trace
```
dsmpqusr (display WebSphere MQ user information)

**Purpose**
Use the `dsmpqusr` command to display information about a specified WebSphere MQ principal that has an entry in the principal database of a queue manager. Alternatively, use the command to display information about all the principals that have entries in the principal database.

**Syntax**
```
dsmpqusr -m QMgrName -p Principal
```

**Description**
For each WebSphere MQ principal, the command displays the following information:
- The principal itself
- The NonStop OS user ID to which the principal maps
- The user groups to which the NonStop OS user ID belongs

**Required parameters**
- `-m QMgrName`
The name of the queue manager whose principal database is to be queried.

**Optional parameters**
- `-p Principal`
The WebSphere MQ principal whose information is to be displayed. If you omit this parameter, the command displays information about all the principals that have entries in the principal database of the queue manager.

**Return codes**
- 0  Successful operation
- 36 Invalid arguments supplied
- 69 Storage not available
- 71 Unexpected error

**Examples**
1. This example shows what `dsmpqusr` displays for a newly created queue manager:

```
dsmpqusr -m MT02
```

<table>
<thead>
<tr>
<th>Principal</th>
<th>Userid</th>
<th>Username</th>
<th>Alias</th>
<th>GroupName</th>
<th>GroupType</th>
</tr>
</thead>
<tbody>
<tr>
<td>mqm</td>
<td>20.255</td>
<td>MQM.MANAGER</td>
<td>n</td>
<td>MQM</td>
<td>a</td>
</tr>
<tr>
<td>nobody</td>
<td>0.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The principal database contains the principal mqm, which maps to the user ID, MQM.MANAGER, of the user who created the queue manager.
2. This example shows what `dspmqusr` displays after additional principals have been added to the principal database using `altmqusr`:

```
dspmqusr -m MT02

<table>
<thead>
<tr>
<th>Principal</th>
<th>Userid</th>
<th>Username</th>
<th>Alias</th>
<th>GroupName</th>
<th>GroupType</th>
</tr>
</thead>
<tbody>
<tr>
<td>mqm</td>
<td>0.1</td>
<td></td>
<td>n</td>
<td>MQM</td>
<td>a</td>
</tr>
<tr>
<td>nobody</td>
<td>0.0</td>
<td></td>
<td>n</td>
<td>MQM</td>
<td>a</td>
</tr>
<tr>
<td>mquser1</td>
<td>50.3</td>
<td>MQTEST.FRED</td>
<td>n</td>
<td>MQTEST</td>
<td>a</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>MQM</td>
<td>s</td>
</tr>
<tr>
<td>mquser2</td>
<td>1.1</td>
<td>GROUP.USER01</td>
<td>n</td>
<td>GROUP</td>
<td>a</td>
</tr>
</tbody>
</table>
```

Principal `mquser1`, which maps to user ID `MQTEST.FRED`, has been added. `MQTEST.FRED` is a member of two user groups, `MQTEST` and `MQM`. Principal `mquser2`, which maps to user ID `GROUP.USER01`, has also been added. `GROUP.USER01` is a member of only one user group called `GROUP`.

**Related commands**

`altmqusr` Alter WebSphere MQ user information
endmqcsv

endmqcsv (end command server)

Purpose
Use the endmqcsv command to stop the command server on the specified queue manager.

Syntax

```markdown
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

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Command completed normally</td>
</tr>
<tr>
<td>10</td>
<td>Command completed with unexpected results</td>
</tr>
<tr>
<td>20</td>
<td>An error occurred during processing</td>
</tr>
</tbody>
</table>

Examples

1. The following command stops the command server on queue manager saturn.queue.manager:
   ```shell
   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:
   ```shell
   endmqcsv -i pluto
   ```

Related commands

dspmqcsv  Display command server
strmqcsv  Start command server
endmqlsr (end listener)

Purpose

The `endmqlsr` command ends all listener processes for the specified queue manager.

Stop the queue manager before issuing the `endmqlsr` command.

Syntax

```
endmqlsr -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

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Command completed normally</td>
</tr>
<tr>
<td>10</td>
<td>Command completed with unexpected results</td>
</tr>
<tr>
<td>20</td>
<td>An error occurred during processing</td>
</tr>
</tbody>
</table>
endmqm

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.

Syntax

```
endmqm [-c] [-w] [-i] [-p] QMgrName
```

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.
  
  You receive the message `Waiting for queue manager QMgrName to end` while the shutdown is in progress.

- `-w` Wait shutdown.
  
  This type of shutdown is the same as a controlled shutdown.

- `-i` Immediate shutdown.
  
  The queue manager stops after it has completed all the MQI calls currently being processed. Any MQI requests issued after the command has been issued fail. Any incomplete units of work are rolled back when the queue manager is next started.

  Control is returned after the queue manager has ended.

- `-p` Preemptive shutdown.
  
  *Use this type of shutdown only in exceptional circumstances.* For example, when a queue manager does not stop as a result of a normal `endmqm` command.
The queue manager 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.

-\texttt{-z} Suppresses error messages on the command.

### Return codes

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Queue manager ended</td>
</tr>
<tr>
<td>3</td>
<td>Queue manager being created</td>
</tr>
<tr>
<td>16</td>
<td>Queue manager does not exist</td>
</tr>
<tr>
<td>40</td>
<td>Queue manager not available</td>
</tr>
<tr>
<td>49</td>
<td>Queue manager stopping</td>
</tr>
<tr>
<td>69</td>
<td>Storage not available</td>
</tr>
<tr>
<td>71</td>
<td>Unexpected error</td>
</tr>
<tr>
<td>72</td>
<td>Queue manager name error</td>
</tr>
</tbody>
</table>

### Examples

The following examples show commands that stop the specified queue managers.

1. This command ends the queue manager named \texttt{mercury.queue.manager} in a controlled way. All applications currently connected are allowed to disconnect.

   \begin{verbatim}
   endmqm mercury.queue.manager
   \end{verbatim}

2. This command ends the queue manager named \texttt{saturn.queue.manager} immediately. All current MQI calls complete, but no new ones are allowed.

   \begin{verbatim}
   endmqm -i saturn.queue.manager
   \end{verbatim}

### Related commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{crtmqm}</td>
<td>Create queue manager</td>
</tr>
<tr>
<td>\texttt{dltmqm}</td>
<td>Delete queue manager</td>
</tr>
<tr>
<td>\texttt{strmqm}</td>
<td>Start queue manager</td>
</tr>
</tbody>
</table>
endmqtrc

endmqtrc (end trace)

Purpose

Use the endmqtrc command to end tracing for the specified queue manager, or for all queue managers in the installation.

Syntax

```
endmqtrc [-m QMgrName] [-e] [-a]
```

Description

This command ends tracing only for those server processes of a queue manager that are running in one specific CPU. By default, this is the same CPU as the one in which your OSS shell or TACL session is running. To end tracing for those server processes of a queue manager that are running in another CPU, you must precede the endmqtrc command by run -cpu=n at an OSS shell command prompt, or run /CPU n/ at a TACL command prompt, where n is the CPU number. For an example, see “Examples.”

Optional parameters

- **-m QMgrName**
  The name of the queue manager for which to end tracing.

  You can include both this parameter and the -e parameter in the same command. But, unless you use the -a parameter, you must include at least this parameter or the -e parameter in the command. It is an error to enter the command without any parameters.

- **-e**
  Ends early tracing for all queue managers in the installation.

- **-a**
  Ends tracing for all queue managers in the installation.

Return codes

**AMQ5611** The command has parameters that are not valid.

Examples

1. This command ends tracing for those server processes of queue manager QM1 that are running in the same CPU as the OSS shell or TACL session in which the command is entered. The tracing of the server processes of queue manager QM1 that are running in other CPUs is unaffected.

   `endmqtrc -m QM1`

2. This command ends tracing for those server processes of queue manager QM2 that are running in CPU 3. The tracing of the server processes of queue manager QM2 that are running in other CPUs is unaffected. This is how to enter the command at an OSS shell command prompt:

   `run -cpu=3 endmqtrc -m QM2`

   This is how to enter the command at a TACL command prompt:

   `run /CPU 3/ endmqtrc -m QM2`
3. This command ends tracing for all queue managers in an installation, but only for those server processes that are running in the same CPU as the OSS shell or TACL session in which the command is entered. The tracing of the server processes that are running in other CPUs is unaffected.
endmqtrc -a

Related commands

dspmqtrc       Display formatted trace
strmqtrc       Start trace
runmqchi

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".

Syntax

```bash
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

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Command completed normally</td>
</tr>
<tr>
<td>10</td>
<td>Command completed with unexpected results</td>
</tr>
<tr>
<td>20</td>
<td>An error occurred during processing</td>
</tr>
</tbody>
</table>

If return codes 10 or 20 are generated, review the error log of the associated queue manager for the error messages, and review the installation wide error log for records of problems that occur before a channel is associated with the queue manager. For more information about error logs, see “Error log files” on page 231.
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

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Command completed normally</td>
</tr>
<tr>
<td>10</td>
<td>Command completed with unexpected results</td>
</tr>
<tr>
<td>20</td>
<td>An error occurred during processing</td>
</tr>
</tbody>
</table>

If return codes 10 or 20 are generated, review the error log of the associated queue manager for the error messages, and review the installation wide 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 log files" on page 231.
runmqdlq

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
```
runmqdlq QName QMgrName
```

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 DLQ handler reads its input from the standard IN file, or stdin within an OSS shell, and writes its results and a summary to a report that is sent to the standard OUT file, or stdout within an OSS shell.

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 input from a file containing a rules table, the DLQ handler reads its input from the keyboard. The DLQ handler does not start to process the named queue until it reads an end of file character, Ctrl+Y.

For more information about rules tables and how to construct them, see "The DLQ handler rules table" on page 188.

Optional parameters

QName
  The name of the queue to be processed.

  If you omit the name, the dead letter queue defined for the local queue manager is used. If you enter one or more blanks (" "), the dead letter queue of the local queue manager is explicitly assigned.

QMgrName
  The name of the queue manager that owns the queue to be processed.

  If you omit the name, the default queue manager for the installation is used. If you enter one or more blanks (" "), the default queue manager for this installation is explicitly assigned.
runmqlsr (run listener)

Purpose

Use the runmqlsr command to start a listener process.

Syntax

```
runmqlsr -t tcp [-u] [-m QMgrName]
```

TCP/IP parameters:

```
-lu62 [-p Port] [-i IPAddr] [-b Backlog] [-g TCP/IPProcessName]
```

Description

If you are using SNA LU6.2, you cannot enter this command at an OSS shell command prompt or a TACL command prompt. An LU6.2 listener can run only as a server process within a SNAX or ICE Pathway environment. However, when you configure a server class for an LU6.2 listener, you must set one of the attributes of the server class to a string containing the relevant parameters of this command.

For more information about configuring and starting an LU6.2 listener, see Appendix K, “Setting up communications,” on page 479.

If you are using TCP/IP and enter this command at an OSS shell command prompt or a TACL command prompt, the command does not return the control to the prompt until the listener ends.

Required parameters

- `-t` The communications protocol to be used:
  - `tcp` TCP/IP
  - `lu62` SNA LU 6.2

Optional parameters

- `-p Port`
  The port number for TCP/IP. If you omit the parameter, the listener listens on the port specified by the Port entry in the TCP stanza in the queue manager configuration file, qm.ini. If there is no Port entry in the queue manager configuration file, the listener listens on port number 1414 by default.

- `-i IPAddr`
  The IP address for the listener, specified in dotted decimal or alphanumeric format. If you omit this parameter, the listener listens on all IP addresses available to the TCP/IP stack.

- `-b Backlog`
  The maximum number of connection requests that can be waiting to be accepted by the listener. For more information about using this parameter, see “TCP” on page 140.
runmqslr

-g TCP/IPProcessName
   The name of the TCP/IP process to be used by the listener and all the
   responder MCAs that implement channels started by the listener. The process
   name must be a valid NonStop OS process name. If you omit this parameter,
   the listener and responder MCAs uses the default TCP/IP process, $ZTC0.

-u   The listener starts each channel using an MCA that runs as a process, not as a
      thread. If you omit this parameter, the listener starts each channel using an
      MCA that runs as a thread.

-m QMgrName
   The name of the queue manager. By default the command operates on the
   default queue manager.

Return codes
   0   Command completed normally
   10  Command completed with unexpected results
   20  An error occurred during processing

Examples
   The following command starts a TCP/IP listener for the default queue manager,
   The listener listens on port number 1415.
       runmqslr -t tcp -p 1415
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

```
runmqsc [-e] [-v] [-w WaitTime] [-x] QMgrName
```

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 reads its input from the standard IN file, or stdin within an OSS shell. When the MQSC commands are processed, the results and a summary are written to a report that is sent to the standard OUT file, or stdout within an OSS shell.

By taking input from the keyboard, you can enter MQSC commands interactively.

By redirecting the input from a file, you can run a sequence of frequently used MQSC commands that are 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 MQSC commands without performing the actions. This mode is available only locally.

- `-w WaitTime`
  
  Run the MQSC commands on another queue manager in indirect mode. You
must have the required channels and transmission queues set up for this. See "Preparing channels and transmission queues for remote administration" on page 64 for more information.

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.

**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.

-x The target queue manager is running on z/OS. 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

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>MQSC command file processed successfully</td>
</tr>
<tr>
<td>10</td>
<td>MQSC command file processed with errors; report contains reasons for failing commands</td>
</tr>
<tr>
<td>20</td>
<td>Error; MQSC command file not run</td>
</tr>
</tbody>
</table>

### Examples

1. Enter this command at a 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 by the default queue manager.

2. Enter the following command at an OSS shell command prompt to verify the MQSC commands in the file /home/username/mqscin:

   ```
   runmqsc -v saturn.queue.manager < /home/username/mqscin
   ```

   Alternatively, enter the following command at a TACL command prompt to verify the MQSC commands in the file called mqscin:

   ```
   runmqsc /IN mqscin/ -v saturn.queue.manager
   ```

   In each case, the MQSC commands are processed by the queue manager saturn.queue.manager, and the output is displayed on the screen.

3. Enter the following command at an OSS shell command prompt to run the MQSC commands in the file /home/username/mqscin:

   ```
   runmqsc < /home/username/mqscin > /home/username/mqscout
   ```

   The output is directed to the file /home/username/mqscout.
   Alternatively, enter the following command at a TACL command prompt to run the MQSC commands in the file called mqscin:

   ```
   runmqsc /IN mqscin, OUT mqscout/
   ```

   The output is directed to the file called mqscout.
In each case, the MQSC commands are processed by the default queue manager.
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.

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 not end, and so a value of 0 is never returned. The value is reserved.
- `10` Trigger monitor interrupted by an error.
- `20` Error; trigger monitor not run.
Purpose

Use the `setmqaut` command to change the authorizations for a profile or an individual WebSphere MQ object. Authorizations can be granted to, or revoked from, any number of user groups.

Syntax

```
setmqaut -m QMgrName -n Profile -t ObjectType
```

MQI authorizations:

```
+altusr -altusr +browse -browse +connect -connect +get -get +inq -inq +put -put +set -set
```
setmqaut

**Context authorizations:**

```
+passall
−passall
+passid
−passid
+setall
−setall
+setid
−setid
```

**Administration authorizations:**

```
+chg
−chg
+clr
−clr
+crt
−crt
+dlt
−dlt
+dsp
−dsp
```

**Generic authorizations:**

```
+all
−all
+alladm
−alladm
+allmqi
−allmqi
+none
```

**Description**

Use `setmqaut` both to *grant* an authorization, that is, give a WebSphere MQ principal or user group permission to perform an operation, and to *revoke* an authorization, that is, remove the permission to perform an operation. You must specify the principals and user groups to which the authorizations apply, the queue manager, object type, and the profile name identifying the object or objects.

The authorizations that can be granted are categorized as follows:

- Authorizations for issuing MQI calls
- Authorizations for MQI context
- Authorizations for issuing commands for administration tasks
- Generic authorizations

Each authorization to be changed is specified in an authorization list as part of the command. Each item in the list is a string prefixed by a plus sign (+) or a minus
sign (-). For example, if you include +put in the authorization list, you grant
authority to issue MQPUT calls against a queue. Alternatively, if you include −put
in the authorization list, you revoke the authority to issue MQPUT calls.

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 any number of principals, user groups, and authorizations in a
single command, but you must specify at least one principal or user group.

Internally, all authorities are held by user groups, not by principals. This has the
following implications:

- If you use the setmqaut command to grant an authority to a principal, the
  authority is actually granted to the primary user group of the NonStop OS user
  ID to which the principal maps. This means that the authority is effectively
  granted to all members of that user group.
- If the NonStop OS user ID to which a principal maps is member of more than
  one user group, the principal effectively has the combined authorities of all
  those user groups.
- If you use the setmqaut command to revoke an authority from a principal, the
  authority is actually revoked from the primary user group of the NonStop OS
  user ID to which the principal maps. This means that the authority is effectively
  revoked from all members of that user group.

**Required parameters**

- `-t ObjectType`
  The type of object for which to change authorizations.

  Possible values are:

  - `queue` or `q` A queue
  - `qmgr` The queue manager object
  - `process` or `prcs` A process
  - `namelist` or `nl` A namelist

- `-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 151.

  If you give an explicit profile name (without any wildcard characters), the
  object identified must exist.

  You must include this parameter unless you are changing the authorities
  against the queue manager object, in which case you must omit it.

**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 Principal
The WebSphere MQ principal for which to change authorizations. If you specify more than one principal, each principal must be prefixed by the -p flag.

-g GroupName
The name of the user group for which to change authorizations. If you specify more than one user group, the name of each user group must be prefixed by the -g flag.

-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.

For more information about authorization service components, see “Installable services” on page 138, Chapter 19, “Installable services and components,” on page 303, and Chapter 20, “Authorization service,” on page 311.

-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 plus sign (+), indicating that authority is to be given, or a minus sign (-), indicating that authority is to be removed.

For example, to grant authority to issue MQPUT calls, specify +put in the list. To revoke the authority to issue MQPUT calls, specify -put.

Table 23 shows the authorities that can be given to the different object types.

Table 23. Specifying authorities for different object types

<table>
<thead>
<tr>
<th>Authority</th>
<th>Queue</th>
<th>Process</th>
<th>Queue manager</th>
<th>Namelist</th>
</tr>
</thead>
<tbody>
<tr>
<td>all</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>alladm</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>allmqi</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>none</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>altusr</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>browse</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>chg</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>clr</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>connect</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>crt</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>dlt</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>dsp</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>get</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>put</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>inq</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>passall</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>passid</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>
Table 23. Specifying authorities for different object types (continued)

<table>
<thead>
<tr>
<th>Authority</th>
<th>Queue</th>
<th>Process</th>
<th>Queue manager</th>
<th>Namelist</th>
</tr>
</thead>
<tbody>
<tr>
<td>set</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>setall</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>setid</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

**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.

**Authorizations for commands**

- **chg** Change the attributes of the specified object.
- **clr** Clear the specified queue (PCF Clear queue command only).
- **crt** Create objects of the specified type.
- **dlt** Delete the specified object.
- **dsp** Display the attributes of the specified object.

**Authorizations for generic operations**

- **all** Use all operations applicable to the object.
- **alladm** 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
Examples

1. This example shows a command that specifies that the object on which authorizations are being given is the queue orange.queue on queue manager saturn.queue.manager. If the queue does not exist, the command fails.

   ```
   setmqaut -m saturn.queue.manager -n orange.queue -t queue
   -g tango +inq +alladm
   ```

   The authorizations are given to a user group called tango, and the authorization list specifies that the user group can:
   - Issue MQINQ calls
   - Perform all administration operations on that object

2. In this example, the authorization list specifies that a user group called foxy:
   - Cannot issue any 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 -m saturn.queue.manager -n orange.queue -t queue
   -g foxy -allmqi +alladm
   ```

3. This example gives principal user1 full access to all queues with names beginning a.b. on queue manager qmgr1. The profile is persistent, and applies to any queue with a name that matches the profile name. This is the command as entered at an OSS shell command prompt:

   ```
   setmqaut -m qmgr1 -n 'a.b.*' -t q -p user1 +all
   ```

   The following command, entered at a TACL command prompt, performs the same function:

   ```
   setmqaut -m qmgr1 -n a.b.* -t q -p user1 +all
   ```

4. The following command, entered at an OSS shell command prompt, deletes the specified profile:

   ```
   setmqaut -m qmgr1 -n 'a.b.*' -t q -p user1 -remove
   ```

   The following command, entered at a TACL command prompt, performs the same function:

   ```
   setmqaut -m qmgr1 -n a.b.* -t q -p user1 -remove
   ```

5. The following command, entered at an OSS shell command prompt, creates a profile with no authority:

   ```
   setmqaut -m qmgr1 -n 'a.b.*' -t q -p user1 +none
   ```

   The following command, entered at a TACL command prompt, performs the same function:

   ```
   setmqaut -m qmgr1 -n a.b.* -t q -p user1 +none
   ```

Related commands

dmpmqaut    Dump authority
dspmqaut  Display authority
strmqcsv

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.

**Syntax**
```
strmqcsv [QMgrName]
```

**Required parameters**
None

**Optional parameters**

*QMgrName*
The name of the queue manager for which to start the command server.

**Return codes**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Command completed normally</td>
</tr>
<tr>
<td>10</td>
<td>Command completed with unexpected results</td>
</tr>
<tr>
<td>20</td>
<td>An error occurred during processing</td>
</tr>
</tbody>
</table>

**Examples**
The following command starts a command server for queue manager earth:

```
strmqcsv earth
```

**Related commands**

- `dspmqcsv`  Display command server
- `endmqcsv`  End command server
**strmqm (start queue manager)**

**Purpose**
Use the `strmqm` command to start a queue manager.

**Syntax**
```
strmqm [-c] [-z] QMgrName
```

**Optional parameters**
- `-c` Starts the queue manager, redefines the default and system objects, then stops the queue manager. (Use the `crtmqm` 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.
- `-z` Suppresses error messages.
  This flag is used within WebSphere MQ to suppress unwanted error messages. Because using this flag could result in loss of information, do not use it when entering commands on a command line.

**QMgrName**
The name of a queue manager to start, by default the default queue manager.

**Return codes**
- 0  Queue manager started
- 3  Queue manager being created
- 5  Queue manager running
- 16 Queue manager does not exist
- 49 Queue manager stopping
- 69 Storage not available
- 71 Unexpected error
- 72 Queue manager name error

**Examples**
The following command starts the queue manager account:
```
strmqm account
```

**Related commands**
- `crtmqm` Create queue manager
- `dltmqm` Delete queue manager
- `endmqm` End queue manager
**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**

```
strmqtrc [-m QMgrName] [-e] [-t TraceType] [-x TraceType] [-l MaxSize]
```

**Description**

This command starts tracing only for those server processes of a queue manager that are running in one specific CPU. By default, this is the same CPU as the one in which your OSS shell or TACL session is running. To start tracing for those server processes of a queue manager that are running in another CPU, you must precede the `strmqtrc` command by `run -cpu=n` at an OSS shell command prompt, or `run /CPU n/` at a TACL command prompt, where n is the CPU number. For an example, see “Examples” on page 296.

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.

The output file is created in the directory `var_installation_path/var/mqm/trace`.

For examples of trace data generated by this command see “Tracing” on page 233.

**Optional parameters**

`-m QMgrName`

The name of the queue manager to trace.

You can include both this parameter and the `-e` parameter in the same command, but you must include at least one of them.

`-e`

Requests early tracing, making it possible to trace the creation or startup of a queue manager. If you include this parameter, any process belonging to any component of any queue manager in the installation traces its early processing. The default is not to perform early tracing.
-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.

<table>
<thead>
<tr>
<th>Trace Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>all</td>
<td>Output data for every trace point in the system (the default). This trace type activates tracing at default detail level.</td>
</tr>
<tr>
<td>api</td>
<td>Output data for trace points associated with the MQI and major queue manager components.</td>
</tr>
<tr>
<td>commentary</td>
<td>Output data for trace points associated with comments in the WebSphere MQ components.</td>
</tr>
<tr>
<td>comms</td>
<td>Output data for trace points associated with data flowing over communications networks.</td>
</tr>
<tr>
<td>csdata</td>
<td>Output data for trace points associated with internal data buffers in common services.</td>
</tr>
<tr>
<td>csflows</td>
<td>Output data for trace points associated with processing flow in common services.</td>
</tr>
<tr>
<td>detail</td>
<td>Activate tracing at high detail level for flow processing trace points.</td>
</tr>
<tr>
<td>lqmdata</td>
<td>Output data for trace points associated with internal data buffers in local queue manager agents.</td>
</tr>
<tr>
<td>lqmflows</td>
<td>Output data for trace points associated with processing flow in local queue manager agents.</td>
</tr>
<tr>
<td>otherdata</td>
<td>Output data for trace points associated with internal data buffers in other components.</td>
</tr>
<tr>
<td>otherflows</td>
<td>Output data for trace points associated with processing flow in other components.</td>
</tr>
<tr>
<td>parms</td>
<td>Activate tracing at default detail level for flow processing trace points.</td>
</tr>
<tr>
<td>remotedata</td>
<td>Output data for trace points associated with internal data buffers in the communications component.</td>
</tr>
<tr>
<td>remoteflows</td>
<td>Output data for trace points associated with processing flow in the communications component.</td>
</tr>
<tr>
<td>servicedata</td>
<td>Output data for trace points associated with internal data buffers in the service component.</td>
</tr>
<tr>
<td>serviceflows</td>
<td>Output data for trace points associated with processing flow in the service component.</td>
</tr>
<tr>
<td>versiondata</td>
<td>Output data for trace points associated with the version of WebSphere MQ running.</td>
</tr>
</tbody>
</table>
strmqtrc

-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, AMQccppppp.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 AMQccppppp.TRS and a new AMQccppppp.TRC file is started. All trace files are restarted when the maximum limit is reached. If a previous copy of an AMQccppppp.TRS file exists, it is deleted.

Return codes
AMQ7024 The command has parameters that are not valid.
AMQ8304 Nine concurrent traces (the maximum) already running.

Examples
This command enables tracing of processing flow from common services and local queue manager agents for a queue manager called QM1. Trace data is generated at the default level of detail. The command enables tracing only for those server processes of the queue manager that are running in CPU 3. This is how to enter the command at an OSS shell command prompt:
run -cpu=3 strmqtrc -m QM1 -t csflows -t lqmflows -t parms

This is how to enter the command at a TACL command prompt:
run /CPU 3/ strmqtrc -m QM1 -t csflows -t lqmflows -t parms

Related commands
dspmqtrc Display formatted trace
endmqtrc End trace
Part 7. WebSphere MQ installable services and API exits

Chapter 19. Installable services and components .................................................. 303
Why installable services? .................................................. 303
Functions and components .................................................. 304
  Entry points .................................................. 305
  Return codes .................................................. 305
  Component data .................................................. 305
Initialization .................................................. 306
  Primary initialization .................................................. 306
  Secondary initialization .................................................. 306
  Primary termination .................................................. 306
  Secondary termination .................................................. 306
Configuring services and components .................................................. 306
  Service stanza format .................................................. 307
  Service component stanza format .................................................. 307
Creating your own service component .................................................. 308
Using multiple service components .................................................. 308
  Omitting entry points when using multiple components .................................................. 308
  Example of entry points used with multiple components .................................................. 308

Chapter 20. Authorization service .................................................. 311
Object authority manager (OAM) .................................................. 311
  Defining the service to the queue manager .................................................. 311
  Refreshing the OAM after changing a user’s authorization .................................................. 311
  Migrating from MQSeries .................................................. 312
Authorization service .................................................. 312
  Configuring authorization service stanzas .................................................. 312
Authorization service interface .................................................. 313

Chapter 21. Name service .................................................. 315
How the name service works .................................................. 315
  Name service interface .................................................. 316

Chapter 22. Installable services interface reference information ......................... 319
How the functions are shown .................................................. 320
  Parameters and data types .................................................. 320
MQZEP – Add component entry point .................................................. 321
  Syntax .................................................. 321
  Parameters .................................................. 321
    Hconfig (MQHCONFIG) – input .................................................. 321
    Function (MQFUNCTION) – input .................................................. 321
    EntryPoint (PMQFUNCTION) – input .................................................. 321
    CompCode (MQFUNCTION) – output .................................................. 321
    Reason (MQLONG) – output .................................................. 321
  C invocation .................................................. 322
MQHCONFIG – Configuration handle .................................................. 322
  C declaration .................................................. 322
PMQFUNCTION – Pointer to function .................................................. 322
  C declaration .................................................. 322
MQZ_CHECK_AUTHORITY – Check authority .................................................. 323
  Syntax .................................................. 323
  Parameters .................................................. 323
    QMgrName (MQCHAR48) – input .................................................. 323
    EntityName (MQCHAR12) – input .................................................. 323
    EntityType (MQLONG) – input .................................................. 323
    ObjectName (MQCHAR48) – input .................................................. 323
    ObjectType (MQLONG) – input .................................................. 324
    Authority (MQLONG) – input .................................................. 324
    ComponentData .................................................. 324
      (MQBYTE×ComponentDataLength) – input/output .................................................. 326
      Continuation (MQLONG) – output .................................................. 326
      CompCode (MQLONG) – output .................................................. 326
      Reason (MQLONG) – output .................................................. 326
  C invocation .................................................. 327
MQZ_ENUMERATE_AUTHORITY_DATA – Enumerate authority data .................................................. 328
  Syntax .................................................. 328
  Parameters .................................................. 328
    QMgrName (MQCHAR48) – input .................................................. 328
    RefObjectName (MQCHAR48) – input .................................................. 328
    ObjectName (MQCHAR48) – input .................................................. 328
    ObjectType (MQLONG) – input .................................................. 328
    ComponentData .................................................. 328
      (MQBYTE×ComponentDataLength) – input/output .................................................. 329
      Continuation (MQLONG) – output .................................................. 329
      CompCode (MQLONG) – output .................................................. 329
      Reason (MQLONG) – output .................................................. 329
  C invocation .................................................. 330
MQZ_DELETE_AUTHORITY – Delete authority .................................................. 331
  Syntax .................................................. 331
  Parameters .................................................. 331
    QMgrName (MQCHAR48) – input .................................................. 331
    ObjectName (MQCHAR48) – input .................................................. 331
    ObjectType (MQLONG) – input .................................................. 331
    ComponentData .................................................. 331
      (MQBYTE×ComponentDataLength) – input/output .................................................. 331
      Continuation (MQLONG) – output .................................................. 332
      CompCode (MQLONG) – output .................................................. 332
      Reason (MQLONG) – output .................................................. 332
  C invocation .................................................. 332
MQZ_ENUMERATE_ALL_AUTHORITY – Copy all authority .................................................. 332
  Syntax .................................................. 332
  Parameters .................................................. 332
    QMgrName (MQCHAR48) – input .................................................. 332
    CompCode (MQLONG) – output .................................................. 332
    Reason (MQLONG) – output .................................................. 332
  C invocation .................................................. 335
Microcode parameter ................................. 336
  Parameters .................................................. 336
    QMgrName (MQCHAR48) – input .................................................. 336
    CompCode (MQLONG) – output .................................................. 336
    Reason (MQLONG) – output .................................................. 336
  C invocation .................................................. 336

© Copyright IBM Corp. 2006
### MQZ_GET_AUTHORITY – Get authority

**Parameters**
- `QMgrName` (MQCHAR48) – input
- `EntityName` (MQCHAR12) – input
- `EntityType` (MQLONG) – input
- `ObjectName` (MQCHAR48) – input
- `ObjectType` (MQLONG) – input
- `Authority` (MQLONG) – input
- `ComponentData` (MQLONG) – output
- `Hconfig` (MQHCONFIG) – output
- `ComponentDataLength` (MQLONG) – output
- `Reason` (MQLONG) – output

**C invocation**

- For **invocation**

### MQZ_GET_EXPLICIT_AUTHORITY – Get explicit authority

**Parameters**
- `QMgrName` (MQCHAR48) – input
- `EntityName` (MQCHAR12) – input
- `EntityType` (MQLONG) – input
- `ObjectName` (MQCHAR48) – input
- `ObjectType` (MQLONG) – input
- `Authority` (MQLONG) – input
- `ComponentData` (MQLONG) – output

**Syntax**

- `(MQBYTE*ComponentDataLength)`
  - input/output
- `Continuation` (MQLONG) – output
- `CompCode` (MQLONG) – output
- `Reason` (MQLONG) – output

**C invocation**

- For **invocation**

### MQZ_TERM_AUTHORITY – Terminate authorization service

**Parameters**
- `Hconfig` (MQHCONFIG) – input
- `Options` (MQLONG) – input
- `QMgrName` (MQCHAR48) – input
- `ComponentData` (MQLONG) – output

**Syntax**

- `(MQBYTE*ComponentDataLength)`
  - input/output
- `CompCode` (MQLONG) – output
- `Reason` (MQLONG) – output

**C invocation**

- For **invocation**

---

**Note:** This text seems to be a fragment of a larger document, possibly a system administration guide for WebSphere MQ for HP NonStop Server. It appears to describe various MQ (Message Queue) operations and their parameters, syntax, and C language invocations. The text is not complete and may be missing some parts or be in a preliminary or draft state.
ComponentData
(MQBYTE×ComponentDataLength)  
input/output ..................................... 360
Version (MQLONG)  
input/output ..................................... 361
CompCode (MQLONG)  
output ........................................ 361
Reason (MQLONG)  
output ........................................ 361
C invocation ..................................... 361

MQZ_INSERT_NAME – Insert name ........................................ 363
Syntax ........................................ 363
Parameters ....................................... 363
QMgrName (MQCHAR48)  
input ........................................ 363
QName (MQCHAR48)  
input ........................................ 363
ResolvedQMgrName (MQCHAR48)  
input ........................................ 363
ComponentData
(MQBYTE×ComponentDataLength)  
input/output ..................................... 363
Continuation (MQLONG)  
output ........................................ 364
CompCode (MQLONG)  
output ........................................ 364
Reason (MQLONG)  
output ........................................ 364
C invocation ..................................... 364

MQZ_LOOK_UP_NAME – Look up name .................................... 365
Syntax ........................................ 365
Parameters ....................................... 365
QMgrName (MQCHAR48)  
input ........................................ 365
QName (MQCHAR48)  
input ........................................ 365
ResolvedQMgrName (MQCHAR48)  
output ......................................... 365
ComponentData
(MQBYTE×ComponentDataLength)  
input/output ..................................... 365
Continuation (MQLONG)  
output ........................................ 366
CompCode (MQLONG)  
output ........................................ 366
Reason (MQLONG)  
output ........................................ 366
C invocation ..................................... 366

MQZ_TERM_NAME – Terminate name service ............................. 368
Syntax ........................................ 368
Parameters ....................................... 368
Hconfig (MQHCONFIG)  
input ........................................ 368
Options (MQLONG)  
input ........................................ 368
QMgrName (MQCHAR48)  
input ........................................ 368
ComponentData
(MQBYTE×ComponentDataLength)  
input/output ..................................... 368
CompCode (MQLONG)  
output ........................................ 369
Reason (MQLONG)  
output ........................................ 369
C invocation ..................................... 369

Chapter 23. API exits .................................................. 371
Why use API exits ........................................ 371
How you use API exits ..................................... 371
How to configure WebSphere MQ for API exits ..................... 371
How to write an API exit .................................... 372
What happens when an API exit runs? .............................. 372
Configuring API exits ....................................... 373
Attributes for all stanzas ...................................... 373
Sample stanzas ........................................ 374
Changing the configuration information ......................... 375

Chapter 24. API exit reference information .......................... 377
General usage notes ........................................ 377
MQACH – API exit chain header .................................... 379
Fields ........................................ 379
Strucid (MQCHAR4) .................................... 379
Version (MQLONG) .................................... 380
StrucLength (MQLONG) ................................ 380
ChainAreaLength (MQLONG) ................................ 380
ExitInfoName (MQCHAR48) ................................ 381
NextChainAreaPtr (PMQACH) ................................ 381
C declaration ........................................ 381

MQAXC – API exit context ........................................... 382
Fields ........................................ 382
Strucid (MQCHAR4) .................................... 382
Version (MQLONG) .................................... 382
Environment (MQLONG) .................................. 383
UserId (MQCHAR12) .................................... 383
SecurityId (MQBYTE40) .................................. 383
ConnectionName (MQCHAR264) ................................ 384
LongMCAUserLength (MQLONG) ............................. 384
LongRemoteUserIdLength (MQLONG) .......................... 384
LongMCAUserPtr (MQPTR) ................................ 384
LongRemoteUserIdPtr (MQPTR) ............................. 384
AppName (MQCHAR28) .................................... 384
AppType (MQLONG) ..................................... 385
ProcessId (MQUID) ..................................... 385
ThreadId (MQUITD) ..................................... 385
C declaration ........................................ 385

MQAXP – API exit parameter ......................................... 386
Fields ........................................ 386
Strucid (MQCHAR4) .................................... 386
Version (MQLONG) .................................... 386
ExitId (MQLONG) ...................................... 387
ExitReason (MQLONG) ................................... 387
ExitResponse (MQLONG) ................................ 388
ExitResponse2 (MQLONG) ................................ 389
Feedback (MQLONG) .................................... 390
APICallerType (MQLONG) ................................ 390
ExitUserData (MQBYTE16) ................................ 390
ExitData (MQCHAR32) ................................... 391
ExitInfoName (MQCHAR48) ................................ 391
ExitPDArea (MQBYTE48) .................................. 391
QMgrName (MQCHAR48) .................................. 391
ExitChainAreaPtr (PMQACH) ................................ 391
Hconfig (MQHCONFIG) ...................................... 392
Function (MQLONG) ..................................... 392
C declaration ........................................ 392

MQXP – Register entry point ......................................... 394
Syntax ........................................ 394
Parameters ....................................... 394
Hconfig (MQHCONFIG)  
input ........................................ 394
ExitReason (MQLONG)  
input ........................................ 394
Function (MQLONG)  
input ........................................ 394
EntryPoint (PMQFUNC)  
input ........................................ 395
Reserved (MQPTR)  
input ........................................ 395
pCompCode (PMQLONG)  
output ........................................ 395
pReason (PMQLONG)  
output ........................................ 396
C invocation ..................................... 396

MQ_BACK_EXIT – Back out changes .................................. 397
Syntax ........................................ 397
Parameters ....................................... 397
pExitParms (PMQAXP)  
input/output ..................................... 397
pExitContext (PMQAXC)  
input/output ..................................... 397
pHconn (PMQHCONN)  
input/output ..................................... 397
pCompCode (PMQLONG)  
input/output ..................................... 397

Part 7. WebSphere MQ installable services and API exits 299
MQ_CLOSE_EXIT – Close object
Syntax
Parameters
pExitParms (PMQAXP) – input/output
pExitContext (PMQAXC) – input/output
pHconn (PMQHCONN) – input/output
ppHobj (PPMHOBJ) – input/output
pOptions (PMQLONG) – input/output
pCompCode (PMQLONG) – input/output
pReason (PMQLONG) – input/output
C invocation.

MQ_CONNXX_EXIT – Connect queue manager
(extended)
Syntax
Parameters
pExitParms (PMQAXP) – input/output
pExitContext (PMQAXC) – input/output
pQMgrName (PMQCHAR48) – input/output
ppConnectOpts (PPMQCNO) – input/output
ppHconn (PMQHCONN) – input/output
pCompCode (PMQLONG) – input/output
pReason (PMQLONG) – input/output
Usage notes
C invocation.

MQ_DISC_EXIT – Disconnect queue manager
Syntax
Parameters
pExitParms (PMQAXP) – input/output
pExitContext (PMQAXC) – input/output
ppHconn (PMQHCONN) – input/output
pCompCode (PMQLONG) – input/output
pReason (PMQLONG) – input/output
C invocation.

MQ_GET_EXIT – Get message
Syntax
Parameters
pExitParms (PMQAXP) – input/output
pExitContext (PMQAXC) – input/output
pHconn (PMQHCONN) – input/output
pHobj (PMQHOBJ) – input/output
ppMsgDesc (PPMQMD) – input/output
ppGetMsgOpts (PPMQGM) – input/output
pBufferLength (PMQLONG) – input/output
pBufferSize (PMPQMD) – input/output
pDataLength (PMQLONG) – input/output
pCompCode (PMQLONG) – input/output
pReason (PMQLONG) – input/output
Usage notes
C invocation.

MQ_INIT_EXIT – Initialize exit environment

MQ_CLOSE_EXIT – Close object
Syntax
Parameters
pExitParms (PMQAXP) – input/output
pExitContext (PMQAXC) – input/output
pHconn (PMQHCONN) – input/output
ppOptions (PMQLONG) – input/output
pCompCode (PMQLONG) – input/output
pReason (PMQLONG) – input/output
C invocation.

MQ_CONNXX_EXIT – Connect queue manager
(extended)
Syntax
Parameters
pExitParms (PMQAXP) – input/output
pExitContext (PMQAXC) – input/output
pQMgrName (PMQCHAR48) – input/output
ppConnectOpts (PPMQCNO) – input/output
ppHconn (PMQHCONN) – input/output
pCompCode (PMQLONG) – input/output
pReason (PMQLONG) – input/output
C invocation.

MQ_DISC_EXIT – Disconnect queue manager
Syntax
Parameters
pExitParms (PMQAXP) – input/output
pExitContext (PMQAXC) – input/output
ppHconn (PMQHCONN) – input/output
pCompCode (PMQLONG) – input/output
pReason (PMQLONG) – input/output
C invocation.

MQ_GET_EXIT – Get message
Syntax
Parameters
pExitParms (PMQAXP) – input/output
pExitContext (PMQAXC) – input/output
pHconn (PMQHCONN) – input/output
pHobj (PMQHOBJ) – input/output
ppMsgDesc (PPMQMD) – input/output
ppGetMsgOpts (PPMQGM) – input/output
pBufferLength (PMQLONG) – input/output
pBufferSize (PMPQMD) – input/output
pDataLength (PMQLONG) – input/output
pCompCode (PMQLONG) – input/output
pReason (PMQLONG) – input/output
Usage notes
C invocation.

MQ_PUT_EXIT – Put message
Syntax
Parameters
pExitParms (PMQAXP) – input/output
pExitContext (PMQAXC) – input/output
pHconn (PMQHCONN) – input/output
pHobj (PMQHOBJ) – input/output
ppObjDesc (PPMQOD) – input/output
pOptions (PMQLONG) – input/output
pCompCode (PMQLONG) – input/output
pReason (PMQLONG) – input/output
C invocation.

MQ_PUT1_EXIT – Put one message
Syntax
Parameters
pExitParms (PMQAXP) – input/output
pExitContext (PMQAXC) – input/output
pHconn (PMQHCONN) – input/output
ppMsgDesc (PPMQMD) – input/output
pBufferSize (PMPQMD) – input/output
pBuffer (PMPQMD) – input/output
pDataLength (PMQLONG) – input/output
pCompCode (PMQLONG) – input/output
pReason (PMQLONG) – input/output
Usage notes
C invocation.

MQ_INIT_EXIT – Initialize exit environment
MQ_SET_EXIT – Set object attributes

Syntax

Parameters

pExitParms (PMQAXP) – input/output
pExitContext (PMQAXC) – input/output
pHconn (PMQHCONN) – input/output
pHobj (PMQHOBJ) – input/output
pSelectorCount (PMQLONG) – input/output
ppSelectors (PPMQLONG) – input/output
pIntAttrCount (PMQLONG) – input/output
ppIntAttrs (PPMQCHAR) – input/output
pCharAttrLength (PMQLONG) – input/output
ppCharAttrs (PPMQCHAR) – input/output
pCompCode (PMQLONG) – input/output
pReason (PMQLONG) – input/output
C invocation.

MQ_TERM_EXIT – Terminate exit environment

Syntax

Parameters

pExitParms (PMQAXP) – input/output
pExitContext (PMQAXC) – input/output
pCompCode (PMQLONG) – input/output
pReason (PMQLONG) – input/output
Usage notes

C invocation.

Chapter 25. WebSphere MQ constants

List of constants

MQ_* (Lengths of character string and byte fields).
MQACH_* (API exit chain header length).
MQACH_* (API exit chain header structure identifier).
MQACH_* (API exit chain header version).
MQAXC_* (API exit context structure identifier).
MQAXC_* (API exit context version).
MQAXP_* (API exit parameter structure identifier).
MQAXP_* (API exit parameter version).
MQCC_* (Completion code).
MQFB_* (Feedback).
MQOT_* (Object type).
MQRC_* (Reason code).
MQSID_* (Security identifier).
MQXACT_* (API exit caller type).
MQXCC_* (Exit response).
MQXE_* (API exit environment).
MQXF_* (API exit function identifier).
MQXPD_* (API exit problem determination area).
MQXR_* (Exit reason).
MQXBR_* (Secondary exit response).
MQXT_* (Exit identifier).
MQXUA_* (Exit user area).
MQZAD_* (Authority data structure identifier).
MQZAD_* (Authority data version).
MQZADE_* (Authority service entity type).
MQZAO_* (Authority service authorization type).
MQZAS_* (Authority service version).
MQZCI_* (Continuation indicator).
MQZED_* (Entity descriptor structure identifier).
MQZED_* (Entity descriptor version).
MQZID_* (Function identifier, all services).
MQZID_* (Function identifier, authority service).
MQZID_* (Function identifier, name service).
MQZID_* (Function identifier, userid service).
MQZIO_* (Initialization options).
MQZNS_* (Name service version).
MQZSE_* (Start-enumeration indicator).
MQZTO_* (Termination options).
MQZUS_* (Userid service version).
Chapter 19. Installable services and components

This chapter introduces the installable services and the functions and components associated with them. The interface to these functions is documented so that you, or software vendors, can supply components.

The chapter includes:

- “Why installable services?”
- “Functions and components” on page 304
- “Initialization” on page 306
- “Configuring services and components” on page 306
- “Creating your own service component” on page 308
- “Using multiple service components” on page 308

The installable services interface is described in Chapter 22, “Installable services interface reference information,” on page 319.

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 NonStop OS.

Table 24. Installable service components summary

<table>
<thead>
<tr>
<th>Installable service</th>
<th>Supplied component</th>
<th>Function</th>
<th>Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authorization service</td>
<td>Object Authority Manager (OAM)</td>
<td>Provides authorization checking on commands and MQI calls. Users can write their own component to augment or replace the OAM.</td>
<td>(Appropriate platform authorization facilities are assumed)</td>
</tr>
</tbody>
</table>
Installable services

Table 24. Installable service components summary (continued)

<table>
<thead>
<tr>
<th>Installable service</th>
<th>Supplied component</th>
<th>Function</th>
<th>Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name service</td>
<td>No supplied component</td>
<td>• Allows queue managers to share queues, or • User defined <strong>Note</strong>: Shared queues must have their <em>Scope</em> attribute set to CELL.</td>
<td>A third-party or user-written name manager</td>
</tr>
</tbody>
</table>

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, the supplied authorization service component, the OAM, performs all the available functions. See “Authorization service interface” on page 313 for more information. The component is also responsible for managing any underlying resources or software (for example, user-written name services) 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 27 on page 305 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 can be 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.
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. 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 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.

### 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 component stanza format

The format of the *Service component* stanza is:

```
ServiceComponent:
    Service=<service_name>
    Name=<component_name>
    Module=<module_name>
    ComponentDataSize=<size>
```

where:

- `<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. 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 a service component

Creating your own service component

To create your own service component:

- Ensure that the header file cmqzc.h is included in your program.
- Create the shared library by compiling the program and linking it with the shared libraries libmqm_r and libmqmzf_r.

**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.

- 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.

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 functions 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 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). 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.

<table>
<thead>
<tr>
<th>Function number</th>
<th>ABC name service component</th>
<th>XYZ name service component</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQZID_INIT_NAME (Initialize)</td>
<td>ABC_initialize()</td>
<td>XYZ_initialize()</td>
</tr>
<tr>
<td>MQZID_TERM_NAME (Terminate)</td>
<td>ABC_terminate()</td>
<td>XYZ_terminate()</td>
</tr>
<tr>
<td>MQZID_INSERT_NAME (Insert)</td>
<td>ABC_Insert()</td>
<td>NULL</td>
</tr>
</tbody>
</table>
Table 25. Example of entry-points for an installable service (continued)

<table>
<thead>
<tr>
<th>Function number</th>
<th>ABC name service component</th>
<th>XYZ name service component</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQZID_DELETE_NAME</td>
<td>ABC_Delete()</td>
<td>NULL</td>
</tr>
<tr>
<td>MQZID_LOOKUP_NAME</td>
<td>NULL</td>
<td>XYZ_Lookup()</td>
</tr>
</tbody>
</table>

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 page 312
- “Authorization service interface” on page 313

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 dspmqaut (display authority), dspmqmaut (display object authority), dmpmqaut (dump object authority), and setmqaut (set and reset authority).

The syntax of these commands and how to use them are described in Chapter 18, “The control commands,” on page 243.

The OAM works with the entity of a principal or group. The principal associated with a user ID is determined using the Principal Database on the HP NonStop Server platform. On other platforms the principal is identical to the user ID.

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 queue manager

The authorization service stanzas in the queue manager configuration file, qm.ini, define the authorization service to the queue manager. See “Configuring services and components” on page 306 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.

Migrating from MQSeries®

When you use the upgmqm utility to create a Version 5.3 queue manager from an existing Version 5.1 queue manager, the utility copies all the authorization data from the Version 5.1 queue manager to the newly created Version 5.3 queue manager. For more information about using the upgmqm utility, see the WebSphere MQ for HP NonStop Server, V5.3 Quick Beginnings.

Authorization service

Principal
Is an identifier associated with a NonStop OS user ID using the Principal Database. The altmqusr control command is used to add or modify entries in the Principal Database.

Group
Is a NonStop OS user group (defined in Safeguard).

You can grant or revoke authorizations at the group level only. When a principal’s authority is granted or revoked, the primary group for the user ID associated with that principal is updated.

Configuring authorization service stanzas

Each queue manager has its own queue manager configuration file.

For example, the default path and file name of the queue manager configuration file for queue manager QMNAME is var_installation_path/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 you can override them using mqsnaout. Add any other ServiceComponent stanzas manually.

For example, the following stanzas in the queue manager configuration file define two authorization service components:

Service:
Name=AuthorizationService
EntryPoints=7

ServiceComponent:
Service=AuthorizationService
Name=MQ.UNIX.authorization.service
Module=/usr/mqm/lib/amqzfuma
ComponentDataSize=0

ServiceComponent:
Service=AuthorizationService
Name=user.defined.authorization.service
Module=/usr/bin/udas01
ComponentDataSize=96

Figure 28. Authorization service stanzas in qm.ini

The service component stanza (MQ.UNIX.authorization.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_INIT_AUTHORITY**
Initializes authorization service component.

**MQZ_TERM_AUTHORITY**
Terminates authorization service component.

**MQZ_CHECK_AUTHORITY**
Checks whether an entity has authority to perform one or more operations on a specified object.

**MQZ_SET_AUTHORITY**
Sets the authority that an entity has to a specified object.

**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_COPY_ALL_AUTHORITY**
Copies all the current authorizations that exist for a referenced object to another object.

**MQZ_ENUMERATE_AUTHORITY_DATA**
Retrieves all the authority data that matches the selection criteria specified.

**MQZ_DELETE_AUTHORITY**
Deletes all authorizations associated with a specified object.

**MQZ_REFRESH_CACHE**
Refresh all authorizations.

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 308 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. The remote queues must have their Scope attribute set to CELL. A name service is not invoked for objects other than queues.

When an application opens a queue, it first looks for the name of the queue in the queue manager’s directory. If the application does not find the name of the queue there, the application looks in as many name services as have been configured, until it finds one that recognizes the queue name. If none of the name services 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, a DCE 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.
3. No remote queue definition is required when sending to a queue with a scope of CELL.
Name service

4. The naming service defines the destination queue centrally, 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. For example, this component could be used 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 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 29. Name service stanzas in qm.ini*
Chapter 22. Installable services interface reference information

This chapter provides reference information for the installable services. The chapter contains the following sections:

- “How the functions are shown” on page 320
- “MQZEP – Add component entry point” on page 321
- “MQHCONFIG – Configuration handle” on page 322
- “PMQFUNC – Pointer to function” on page 322
- “MQZ_CHECK_AUTHORITY – Check authority” on page 323
- “MQZ_COPY_ALL_AUTHORITY – Copy all authority” on page 328
- “MQZ_DELETE_AUTHORITY – Delete authority” on page 331
- “MQZ_ENUMERATE_AUTHORITY_DATA – Enumerate authority data” on page 334
- “MQZ_GET_AUTHORITY – Get authority” on page 337
- “MQZ_GET_EXPLICIT_AUTHORITY – Get explicit authority” on page 340
- “MQZ_INIT_AUTHORITY – Initialize authorization service” on page 343
- “MQZ_REFRESH_CACHE – Refresh all authorizations” on page 346
- “MQZ_SET_AUTHORITY – Set authority” on page 348
- “MQZ_TERM_AUTHORITY – Terminate authorization service” on page 351
- “MQZAD – Authority data” on page 353
- “MQZED – Entity descriptor” on page 356
- “MQZ_DELETE_NAME – Delete name” on page 358
- “MQZ_INIT_NAME – Initialize name service” on page 360
- “MQZ_INSERT_NAME – Insert name” on page 363
- “MQZ_LOOKUP_NAME – Lookup name” on page 365
- “MQZ_TERM_NAME – Terminate name service” on page 368

The functions and data types are in alphabetic order within the group for each service type.

Table 26. Installable services functions

<table>
<thead>
<tr>
<th>Installable service</th>
<th>Functions and data types</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>All services</td>
<td>MQZEP</td>
<td>321</td>
</tr>
<tr>
<td></td>
<td>MQHCONFIG</td>
<td>322</td>
</tr>
<tr>
<td></td>
<td>PMQFUNC</td>
<td>322</td>
</tr>
</tbody>
</table>
Installable services

Table 26. Installable services functions (continued)

<table>
<thead>
<tr>
<th>Installable service</th>
<th>Functions and data types</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authorization service</td>
<td>MQZ_CHECK_AUTHORITY</td>
<td>323</td>
</tr>
<tr>
<td></td>
<td>MQZ_COPY_ALL_AUTHORITY</td>
<td>328</td>
</tr>
<tr>
<td></td>
<td>MQZ_DELETE_AUTHORITY</td>
<td>331</td>
</tr>
<tr>
<td></td>
<td>MQZ_ENUMERATE_AUTHORITY_DATA</td>
<td>334</td>
</tr>
<tr>
<td></td>
<td>MQZ_GET_AUTHORITY</td>
<td>337</td>
</tr>
<tr>
<td></td>
<td>MQZ_GET_EXPLICIT_AUTHORITY</td>
<td>340</td>
</tr>
<tr>
<td></td>
<td>MQZ_INIT_AUTHORITY</td>
<td>343</td>
</tr>
<tr>
<td></td>
<td>MQZ_REFRESH_CACHE</td>
<td>346</td>
</tr>
<tr>
<td></td>
<td>MQZ_SET_AUTHORITY</td>
<td>348</td>
</tr>
<tr>
<td></td>
<td>MQZ_TERM_AUTHORITY</td>
<td>351</td>
</tr>
<tr>
<td></td>
<td>MQZAD</td>
<td>353</td>
</tr>
<tr>
<td></td>
<td>MQZED</td>
<td>356</td>
</tr>
<tr>
<td>Name service</td>
<td>MQZ_DELETE_NAME</td>
<td>358</td>
</tr>
<tr>
<td></td>
<td>MQZ_INIT_NAME</td>
<td>360</td>
</tr>
<tr>
<td></td>
<td>MQZ_INSERT_NAME</td>
<td>363</td>
</tr>
<tr>
<td></td>
<td>MQZ_LOOKUP_NAME</td>
<td>365</td>
</tr>
<tr>
<td></td>
<td>MQZ_TERM_NAME</td>
<td>368</td>
</tr>
</tbody>
</table>

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 that 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_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_NAMELIST**
  Namelist.

- **MQOT_PROCESS**
  Process definition.

- **MQOT_Q**
  Queue.

- **MQOT_Q_MGR**
  Queue manager.

**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:

- **MQZAOCONNECT**
  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.

**MQZAO_CLEAR**  
Ability to delete all messages from a specified queue.

**MQZAO_AUTHORIZE**  
Ability to authorize other users for a specified 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:
MQZ_CHECK_AUTHORITY

MQZAO_ALL
All authorizations, other than MQZAO_CREATE.

MQZAO_NONE
No authorizations.

ComponentData (MQBYTExComponentDataLength) – 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:

MQRC_NONE
(0, X’000’) No reason to report.

If CompCode is MQCC_FAILED:

MQRC_NOTAUTHORIZED
(2035, X’7F3’) Not authorized for access.

MQRC_SERVICE_ERROR
(2289, X’8F1’) Unexpected error occurred accessing service.

MQRC_SERVICE_NOTAVAILABLE
(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_COPY_ALL_AUTHORITY

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_NAMELIST
Namelist.

MQOT_PROCESS
Process definition.

MQOT_Q
Queue.
MQOT_Q_MGR
Queue manager.

ComponentData (MQBYTExComponentDataLength) – 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.

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_NAMELIST
Namelist.

MQOT_PROCESS
Process definition.

MQOT_Q
Queue.

MQOT_Q_MGR
Queue manager.

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](https://www.ibm.com/docs/WebSphere-MQ).

**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
MQZ_DELETE_AUTHORITY

MQLONG CompCode; /* Completion code */
MQLONG Reason; /* Reason code qualifying CompCode */
MQZ_ENUMERATE_AUTHORITY_DATA

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 the 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 353 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, because 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. The buffer needs to be bigger than an MQZAD so that it can accommodate the MQZAD, MQZED, as well as 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:

**MQZC1_DEFAULT**
Continuation dependent on queue manager.

For MQZ_ENUMERATE_AUTHORITY_DATA this has the same effect as MQZC1_CONTINUE.

**MQZC1_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**

MQ_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_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_NAMELIST**
  - Namelist.
- **MQOT_PROCESS**
  - Process definition.
- **MQOT_Q**
  - Queue.
- **MQOT_Q_MGR**
  - Queue manager.

**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_EXPLICIT_AUTHORITY

MQZ_GET_EXPLICIT_AUTHORITY – Get explicit authority

This function is provided by a MQZAS_VERSION_1 authorization service component. The function 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 named principal’s primary group 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.
**ObjectType (MQLONG) – input**

Object type.

The type of entity specified by ObjectName. It is one of the following:
- MQOT_AUTH_INFO
  Authentication information.
- MQOT_NAMELIST
  Namelist.
- MQOT_PROCESS
  Process definition.
- MQOT_Q
  Queue.
- MQOT_Q_MGR
  Queue manager.

**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_NOTAUTHORIZED**
(2035, X'7F3') Not authorized for access.

**MQRC_SERVICEERROR**
(2289, X'8F1') Unexpected error occurred accessing service.

**MQRC_SERVICENOTAVAILABLE**
(2285, X'8ED') Underlying service not available.

**MQRC_UNKNOWNENTITY**
(2292, X'8F4') Entity unknown to service.

For more information on these reason codes, see the [WebSphere MQ Application Programming Reference](#).

**C invocation**

```c
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_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
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.

**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](#).
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_REFRESH_CACHE

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](https://www.ibm.com/support/pages/node/98509) book.

**C invocation**

```c
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

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, 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_NAMELIST**
  Namelist.

- **MQOT_PROCESS**
  Process definition.

- **MQOT_Q**
  Queue.

- **MQOT_Q_MGR**
  Queue manager.

**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.

**CompCode (MQLONG) – output**
Completion code.

It is one of the following:

- **MQCC_OK**
  Successful completion.

- **MQCC_FAILED**
  Call failed.
MQZ_SET_AUTHORITY

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_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 */
MQLONG ComponentData[n];  /* Component data */
MQLONG CompCode;  /* Completion code */
MQLONG Reason;  /* Reason code qualifying CompCode */
MQZAD – Authority data

The following table summarizes the fields in the structure.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>StrucId</td>
<td>Structure identifier</td>
<td>353</td>
</tr>
<tr>
<td>Version</td>
<td>Structure version number</td>
<td>353</td>
</tr>
<tr>
<td>ProfileName</td>
<td>Profile name</td>
<td>354</td>
</tr>
<tr>
<td>ObjectType</td>
<td>Object type</td>
<td>354</td>
</tr>
<tr>
<td>Authority</td>
<td>Authority</td>
<td>354</td>
</tr>
<tr>
<td>EntityDataPtr</td>
<td>Address of MQZED structure identifying an entity</td>
<td>354</td>
</tr>
<tr>
<td>EntityType</td>
<td>Type of entity</td>
<td>354</td>
</tr>
</tbody>
</table>

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.
MQZAD – Authority data

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.

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_Q
Queue.
MQOT_NAMELIST
Namelist.
MQOT_PROCESS
Process definition.
MQOT_Q_MGR
Queue manager.
MQOT_AUTH_INFO
Authentication information.

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.

### C declaration

```c
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 entity */
    MQLONG EntityType; /* Entity type */
};
```
MQZED – Entity descriptor

The following table summarizes the fields in the structure.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>StrucId</td>
<td>Structure identifier</td>
<td>356</td>
</tr>
<tr>
<td>Version</td>
<td>Structure version number</td>
<td>356</td>
</tr>
<tr>
<td>EntityNamePtr</td>
<td>Address of entity name</td>
<td>356</td>
</tr>
<tr>
<td>EntityDomainPtr</td>
<td>Address of entity domain name</td>
<td>356</td>
</tr>
<tr>
<td>SecurityId</td>
<td>Security identifier</td>
<td>357</td>
</tr>
</tbody>
</table>

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.
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.

**C declaration**

```c
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 */
};
```
MQZ_DELETE_NAME

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.
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 might 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

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
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 */
MQZ_INIT_NAME

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](https://www-01.ibm.com/support/docview.wss?uid=swg27038377).

**C invocation**

```c
MQZ_INSERT_NAME (QMgrName, QName, ResolvedQMgrName, ComponentData, &Continuation, &CompCode, &Reason);
```

The parameters passed to the service are declared as follows:

```c
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.
MQZ_LOOKUP_NAME

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
MQLONG CompCode; /* Completion code */
MQLONG Reason; /* Reason code qualifying CompCode */
MQZ_TERM_NAME

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.
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](http://www.ibm.com/support/docview.wss?uid=swg27035374).

**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 372
- “Configuring API exits” on page 373

There is a brief introduction to writing API exits in “How to write an API exit” on page 372. 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 avoid having to change the code of each application affected.

Here are a few suggestions of areas where 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 applications 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. For example, you could 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 way: edit the WebSphere MQ configuration files, mqs.ini and qm.ini.
API exits

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 373. For a description of how API exits run, see "What happens when an API exit runs?".

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.

Write your exits using the C programming language. To help you do so, a sample exit, amqsaxe0, is provided that generates trace entries to a named file. When you start writing exits, 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
- MQBACK, to back out a UOW
- MQCMIT, to commit a UOW
- MQOPEN, to open a WebSphere MQ resource for subsequent access
- MQCLOSE, to close a WebSphere MQ 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 a WebSphere MQ 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, which are in mqs.ini and qm.ini. 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" determines the order in which the routines defined in the stanzas run.

**Configuring API exits**

This section tells you how to configure API exits by explaining how to add the stanzas.

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 375.

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.
**API exits**

**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.
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:

**ApixExitTemplate**
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.

**ApixExitCommon**
This stanza defines an exit with the descriptive name MQPoliceman, module
name tmqp, and sequence number 1. The data passed is an instruction
(CheckEverything).

```
ApiExitTemplate:
 Name=OurPayrollQueueAuditor
 Sequence=2
 Function=EntryPoint
 Module=/opt/ABC/auditor
 Data=123

ApiExitCommon:
 Name=MQPoliceman
```
Sequence=1
Function=EntryPoint
Module=/opt/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=/opt/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_installation_path/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_installation_path/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 necessary.

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.)
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 379
  - “MQAXC – API exit context” on page 382
  - “MQAXP – API exit parameter” on page 386

- Calls an API exit function can issue:
  - “MQXEP – Register entry point” on page 394

- Definitions of the API exit functions:
  - “MQ BACK_EXIT – Back out changes” on page 397
  - “MQ CLOSE_EXIT – Close object” on page 398
  - “MQ CMIT_EXIT – Commit changes” on page 399
  - “MQ CONNX_EXIT – Connect queue manager (extended)” on page 400
  - “MQ DISC_EXIT – Disconnect queue manager” on page 402
  - “MQ GET_EXIT – Get message” on page 403
  - “MQ INIT_EXIT – Initialize exit environment” on page 405
  - “MQ INQ_EXIT – Inquire object attributes” on page 406
  - “MQ OPEN_EXIT – Open object” on page 408
  - “MQ PUT_EXIT – Put message” on page 409
  - “MQ_PUT1_EXIT – Put one message” on page 411
  - “MQ SET_EXIT – Set object attributes” on page 413
  - “MQ TERM_EXIT – Terminate exit environment” on page 415

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, 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. 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.

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
API exit – General usage notes

result in the relevant API exit functions being invoked. Exit functions can therefore be invoked recursively. 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 can impair the correct functioning of the application. However, the exit function can 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). 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 might 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 might 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).
The MQACH – API exit chain header

The following table summarizes the fields in the structure.

Table 29. Fields in MQACH

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>StrucId (MQCHAR4)</td>
<td>Structure identifier</td>
<td>379</td>
</tr>
<tr>
<td>Version</td>
<td>Structure version number</td>
<td>380</td>
</tr>
<tr>
<td>StrucLength</td>
<td>Length of MQACH structure</td>
<td>380</td>
</tr>
<tr>
<td>ChainAreaLength</td>
<td>Total length of chain area</td>
<td>380</td>
</tr>
<tr>
<td>ExitInfoName</td>
<td>Exit information name</td>
<td>381</td>
</tr>
<tr>
<td>NextChainAreaPtr</td>
<td>Address of next chain area</td>
<td>381</td>
</tr>
</tbody>
</table>

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 in 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.
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 */
};
The following table summarizes the fields in the structure.

Table 30. Fields in MQAXC

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>StrucId</td>
<td>Structure identifier</td>
<td>382</td>
</tr>
<tr>
<td>Version</td>
<td>Structure version number</td>
<td>382</td>
</tr>
<tr>
<td>Environment</td>
<td>Environment</td>
<td>383</td>
</tr>
<tr>
<td>UserId</td>
<td>User identifier</td>
<td>383</td>
</tr>
<tr>
<td>SecurityId</td>
<td>Security identifier</td>
<td>383</td>
</tr>
<tr>
<td>ConnectionName</td>
<td>Connection name</td>
<td>384</td>
</tr>
<tr>
<td>LongMCAUserIdLength</td>
<td>Length of long MCA user identifier</td>
<td>384</td>
</tr>
<tr>
<td>LongRemoteUserIdLength</td>
<td>Length of long remote user identifier</td>
<td>384</td>
</tr>
<tr>
<td>LongMCAUserIdPtr</td>
<td>Address of long MCA user identifier</td>
<td>384</td>
</tr>
<tr>
<td>LongRemoteUserIdPtr</td>
<td>Address of long remote user identifier</td>
<td>384</td>
</tr>
<tr>
<td>ApplName</td>
<td>Application name</td>
<td>384</td>
</tr>
<tr>
<td>ApplType</td>
<td>Application type</td>
<td>384</td>
</tr>
<tr>
<td>ProcessId</td>
<td>Process identifier</td>
<td>385</td>
</tr>
<tr>
<td>ThreadId</td>
<td>Thread identifier</td>
<td>385</td>
</tr>
</tbody>
</table>

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** 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 that 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.
MQAXC – API exit context

**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**

```c
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 */
};
```
The following table summarizes the fields in the structure.

Table 31. Fields in MQAXP

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>StrucId</td>
<td>Structure identifier</td>
<td>386</td>
</tr>
<tr>
<td>Version</td>
<td>Structure version number</td>
<td>386</td>
</tr>
<tr>
<td>ExitId</td>
<td>Type of exit</td>
<td>387</td>
</tr>
<tr>
<td>ExitReason</td>
<td>Reason for invoking exit</td>
<td>387</td>
</tr>
<tr>
<td>ExitResponse</td>
<td>Response from exit</td>
<td>388</td>
</tr>
<tr>
<td>ExitResponse2</td>
<td>Secondary response from exit</td>
<td>389</td>
</tr>
<tr>
<td>Feedback</td>
<td>Feedback code</td>
<td>390</td>
</tr>
<tr>
<td>APICallerType</td>
<td>API caller type</td>
<td>390</td>
</tr>
<tr>
<td>ExitUserArea</td>
<td>Exit user area</td>
<td>390</td>
</tr>
<tr>
<td>ExitData</td>
<td>Exit data</td>
<td>391</td>
</tr>
<tr>
<td>ExitInfoName</td>
<td>Exit information name</td>
<td>391</td>
</tr>
<tr>
<td>ExitPDArea</td>
<td>Problem determination area</td>
<td>391</td>
</tr>
<tr>
<td>QMgrName</td>
<td>Name of local queue manager</td>
<td>391</td>
</tr>
<tr>
<td>ExitChainAreaPtr</td>
<td>Address of first chain area</td>
<td>391</td>
</tr>
<tr>
<td>Hconfig</td>
<td>Configuration handle</td>
<td>392</td>
</tr>
<tr>
<td>Function</td>
<td>API function identifier</td>
<td>392</td>
</tr>
</tbody>
</table>

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 that 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

For the 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.
MQGET call, this value occurs with the 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:

<table>
<thead>
<tr>
<th>Exit function</th>
<th>Reason code set by queue manager</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQXF_INIT</td>
<td>MQRC_API_EXIT_INIT_ERROR</td>
</tr>
<tr>
<td>MQXF_TERM</td>
<td>MQRC_API_EXIT_TERM_ERROR</td>
</tr>
<tr>
<td>All others</td>
<td>MQRC_API_EXIT_ERROR</td>
</tr>
</tbody>
</table>

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 immediately deregisters all of the exit functions belonging to this exit suite. The only exception is the MQXF_TERM exit function, which is invoked at termination of the connection if MQXF_TERM is registered when MQXCC_SUPPRESS_EXIT is returned. If an MQXR_BEFORE exit function returns this value, the matching MQXR_AFTER exit function is not be invoked after the API call, because that exit function is no longer 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.
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 MQX_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 and 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 MQX_INIT exit function). 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 MQX_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 and 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 and 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.
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, do this 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 that 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 and 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. Specify this handle 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_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

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.
MQXEP call

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_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 that 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_CLOSE_EXIT

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_CMIT_EXIT – Commit changes

Exit providers can supply an MQ_CMIT_EXIT function to intercept the MQCMIT call. If the unit of work is being coordinated by an external unit-of-work manager, MQ_CMIT_EXIT is also invoked in response to the application issuing the unit-of-work manager’s commit call.

Syntax

MQ_CMIT_EXIT (pExitParms, pExitContext, pHconn, pCompCode, pReason)

Parameters

The MQ_CMIT_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_CMIT_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 */
PMLONG  pReason;  /* Reason code qualifying CompCode */
MQ_CONNX_EXIT

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, when 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 might therefore be invalid (including invalid pointers for the addresses of parameters).
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

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


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 (PMQLONG) – 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, you can use a single exit function for both; the
Function field in the MQAXP structure indicates which exit function has been invoked. Alternatively, use the MQXEP call 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 */
PMMQHOBJ   pHobj; /* Object handle */
PMMQMD     ppMsgDesc; /* Message descriptor */
PMMQGMO    ppGetMsgOpts; /* Options that control the action of MQGET */
PMLONG     pBufferLength; /* Length in bytes of the pBuffer area */
PMLONG     pBuffer; /* Area to contain the message data */
PMLONG     ppDataLength; /* Length of the message */
PMLONG     pCompCode; /* Completion code */
PMLONG     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 MQXEP, 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, &pSelectors, &IntAttrCount, 
&ppIntAttrs, &CharAttrLength, &ppCharAttrs, 
&CompCode, &Reason);

The parameters passed to the exit are declared as follows:
PMQAXP  pExitParms;    /* Exit parameter structure */
PMQAXC  pExitContext;  /* Exit context structure */
PMQCONN  pHconn;       /* Connection handle */
PMSOBJ   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 */
PPMQLONG ppCharAttrs;   /* Character attributes */
PMQLONG  pCompCode;     /* Completion code */
PMQLONG  pReason;       /* Reason code qualifying CompCode */
MQ_OPEN_EXIT

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, &pCompCode, &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 ensure you intercept all of the report messages generated by the MCA, use both MQ_PUT_EXIT and MQ_PUT1_EXIT.

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, ppBuffer, 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 ppBuffer.

- **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 */
MQ_PUT1_EXIT

PPMQVOID 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, &pSelectors, &pIntAttrCount, &pIntAttrs, &pCharAttrLength, &ppCharAttrs, &pCompCode, &pReason);

The parameters passed to the exit are declared as follows:
PMQAXP  pExitParms; /* Exit parameter structure */
PMQAXC  pExitContext; /* Exit context structure */
PMQCONN  pHConn; /* Connection handle */
PMQOBJ  pHobj; /* Object handle */
PMQLONG  pSelectorCount; /* Count of selectors */
PMQLONG  ppSelectors; /* Array of attribute selectors */
PMQLONG  pIntAttrCount; /* Count of integer attributes */
PMQLONG  ppIntAttr; /* Array of integer attributes */
PMQLONG  pCharAttrLength; /* Length of character attributes buffer */
PMQLONG  ppCharAttr; /* 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 */
MQ_TERM_EXIT – Usage notes
Chapter 25. WebSphere MQ constants

This chapter specifies the values of the named constants that apply to installable services and the API exit.

The constants are grouped according to the parameter or field to which they relate. All of the names of the constants in a group begin with a common prefix of the form “MQxxxx_”, where xxxx represents a string of 0 through 5 characters that indicates the parameter or field to which the values relate. The constants are ordered alphabetically by this prefix.

Notes:
1. For constants with numeric values, the values are shown in both decimal and hexadecimal forms.
2. Hexadecimal values are represented using the notation X'hhhh', where each “h” denotes a single hexadecimal digit.
3. Character values are shown delimited by single quotation marks; the quotation marks are not part of the value.
4. Blanks in character values are represented by one or more occurrences of the symbol “b”.

List of constants

The following sections list all the named constants that are mentioned in this book, and show their values.

**MQ_** *(Lengths of character string and byte fields)*

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value 1</th>
<th>Value 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQ_APPL_NAME_LENGTH</td>
<td>28</td>
<td>X'0000001C'</td>
</tr>
<tr>
<td>MQ_CONN_NAME_LENGTH</td>
<td>264</td>
<td>X'00000108'</td>
</tr>
<tr>
<td>MQ_EXIT_DATA_LENGTH</td>
<td>32</td>
<td>X'00000020'</td>
</tr>
<tr>
<td>MQ_EXIT_INFO_NAME_LENGTH</td>
<td>48</td>
<td>X'00000030'</td>
</tr>
<tr>
<td>MQ_EXIT_PD_AREA_LENGTH</td>
<td>48</td>
<td>X'00000030'</td>
</tr>
<tr>
<td>MQ_EXIT_USER_AREA_LENGTH</td>
<td>16</td>
<td>X'00000010'</td>
</tr>
<tr>
<td>MQ_Q_MGR_NAME_LENGTH</td>
<td>48</td>
<td>X'00000030'</td>
</tr>
<tr>
<td>MQ_SECURITY_ID_LENGTH</td>
<td>40</td>
<td>X'00000028'</td>
</tr>
<tr>
<td>MQ_USER_ID_LENGTH</td>
<td>12</td>
<td>X'0000000C'</td>
</tr>
</tbody>
</table>

**MQACH_** *(API exit chain header length)*

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQACH_LENGTH_1</td>
<td>(variable)</td>
</tr>
<tr>
<td>MQACH_CURRENT_LENGTH</td>
<td>(variable)</td>
</tr>
</tbody>
</table>

**MQACH_** *(API exit chain header structure identifier)*

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQACH_STRUC_ID</td>
<td>'ACHb'</td>
</tr>
</tbody>
</table>

For the C programming language, the following array version is also defined:
WebSphere MQ constants

MQACH_STRUC_ID_ARRAY  'A','C','H','b'

**MQACH_* (API exit chain header version)**

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQACH_VERSION_1</td>
<td>1</td>
</tr>
<tr>
<td>MQACH_CURRENT_VERSION</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>X'00000001'</td>
</tr>
</tbody>
</table>

**MQAXC_* (API exit context structure identifier)**

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQAXC_STRUC_ID</td>
<td>'AXCb'</td>
</tr>
</tbody>
</table>

For the C programming language, the following array version is also defined:

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQAXC_STRUC_ID_ARRAY</td>
<td>'A','X','C','b'</td>
</tr>
</tbody>
</table>

**MQAXC_* (API exit context version)**

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQAXC_VERSION_1</td>
<td>1</td>
</tr>
<tr>
<td>MQAXC_CURRENT_VERSION</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>X'00000001'</td>
</tr>
</tbody>
</table>

**MQAXP_* (API exit parameter structure identifier)**

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQAXP_STRUC_ID</td>
<td>'AXPb'</td>
</tr>
</tbody>
</table>

For the C programming language, the following array version is also defined:

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQAXP_STRUC_ID_ARRAY</td>
<td>'A','X','P','b'</td>
</tr>
</tbody>
</table>

**MQAXP_* (API exit parameter version)**

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQAXP_VERSION_1</td>
<td>1</td>
</tr>
<tr>
<td>MQAXP_CURRENT_VERSION</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>X'00000001'</td>
</tr>
</tbody>
</table>

**MQCC_* (Completion code)**

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQCC_OK</td>
<td>0</td>
</tr>
<tr>
<td>MQCC_WARNING</td>
<td>1</td>
</tr>
<tr>
<td>MQCC_FAILED</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>X'00000000'</td>
</tr>
<tr>
<td></td>
<td>X'00000001'</td>
</tr>
<tr>
<td></td>
<td>X'00000002'</td>
</tr>
</tbody>
</table>

**MQFB_* (Feedback)**

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQFB_NONE</td>
<td>0</td>
</tr>
<tr>
<td>MQFB_SYSTEM_FIRST</td>
<td>1</td>
</tr>
<tr>
<td>MQFB_SYSTEM_LAST</td>
<td>65535</td>
</tr>
<tr>
<td>MQFB_SYSTEM_LAST</td>
<td>X'0000FFFFFF'</td>
</tr>
<tr>
<td>MQFB_APPL_FIRST</td>
<td>65536</td>
</tr>
<tr>
<td>MQFB_APPL_LAST</td>
<td>X'00010000'</td>
</tr>
<tr>
<td>MQFB_APPL_LAST</td>
<td>X'3B9AC9FF'</td>
</tr>
</tbody>
</table>
MQOT_* (Object type)

MQOT_Q 1 X'00000001'
MQOT_NAMELIST 2 X'00000002'
MQOT_PROCESS 3 X'00000003'
MQOT_Q_MGR 5 X'00000005'
MQOT_AUTH_INFO 7 X'00000007'
MQOT_RESERVED_1 999 X'000003E7'

MQRC_* (Reason code)

MQRC_NONE 0 X'00000000'
MQRC_BUFFER_LENGTH_ERROR 2005 X'000007D5'
MQRC_NOT_AUTHORIZED 2035 X'000007F3'
MQRC_RESOURCE_PROBLEM 2102 X'00000836'
MQRC_SUPPRESSED_BY_EXIT 2109 X'0000083D'
MQRC_UNEXPECTED_ERROR 2195 X'00000893'
MQRC_HCONFIG_ERROR 2280 X'000008E8'
MQRC_FUNCTION_ERROR 2281 X'000008E9'
MQRC_SERVICE_NOT_AVAILABLE 2285 X'000008ED'
MQRC_INITIALIZATION_FAILED 2286 X'000008EE'
MQRC_TERMINATION_FAILED 2287 X'000008EF'
MQRC_UNKNOWN_Q_NAME 2288 X'000008F0'
MQRC_SERVICE_ERROR 2289 X'000008F1'
MQRC_Q_ALREADY_EXISTS 2290 X'000008F2'
MQRC_USER_ID_NOT_AVAILABLE 2291 X'000008F3'
MQRC_UNKNOWN_ENTITY 2292 X'000008F4'
MQRC_UNKNOWN_REF_OBJECT 2294 X'000008F6'
MQRC_WRONG_CF_LEVEL 2366 X'0000093E'
MQRC_API_EXIT_ERROR 2374 X'00000946'
MQRC_API_EXIT_INIT_ERROR 2375 X'00000947'
MQRC_API_EXIT_TERM_ERROR 2376 X'00000948'
MQRC_EXIT_REASON_ERROR 2377 X'00000949'
MQRC_RESERVED_VALUE_ERROR 2378 X'0000094A'
MQRC_NO_DATA_AVAILABLE 2379 X'0000094B'

MQSID_* (Security identifier)

MQSID_NONE X'00...00' (40 nulls)

For the C programming language, the following array version is also defined:

MQSID_NONE_ARRAY '\0','\0',...'\0','\0'

MQXACT_* (API exit caller type)

MQXACT_EXTERNAL 1 X'00000001'
MQXACT_INTERNAL 2 X'00000002'
### WebSphere MQ constants

#### MQXCC_\* (Exit response)

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Hexadecimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQXCC_FAILED</td>
<td>-8</td>
<td>X'FFFFFFFB'</td>
</tr>
<tr>
<td>MQXCC_SUPPRESS_EXIT</td>
<td>-5</td>
<td>X'FFFFFFFB'</td>
</tr>
<tr>
<td>MQXCC_SKIP_FUNCTION</td>
<td>-2</td>
<td>X'FFFFFFFE'</td>
</tr>
<tr>
<td>MQXCC_SUPPRESS_FUNCTION</td>
<td>-1</td>
<td>X'FFFFFEEE'</td>
</tr>
<tr>
<td>MQXCC_OK</td>
<td>0</td>
<td>X'00000000'</td>
</tr>
</tbody>
</table>

#### MQXE_\* (API exit environment)

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Hexadecimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQXE_OTHER</td>
<td>0</td>
<td>X'00000000'</td>
</tr>
<tr>
<td>MQXE_MCA</td>
<td>1</td>
<td>X'00000001'</td>
</tr>
<tr>
<td>MQXE_MCA_SVRCONN</td>
<td>2</td>
<td>X'00000002'</td>
</tr>
<tr>
<td>MQXE_COMMAND_SERVER</td>
<td>3</td>
<td>X'00000003'</td>
</tr>
<tr>
<td>MQXE_MQSC</td>
<td>4</td>
<td>X'00000004'</td>
</tr>
</tbody>
</table>

#### MQXF_\* (API exit function identifier)

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Hexadecimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQXF_INIT</td>
<td>1</td>
<td>X'00000001'</td>
</tr>
<tr>
<td>MQXF_TERM</td>
<td>2</td>
<td>X'00000002'</td>
</tr>
<tr>
<td>MQXF_CONN</td>
<td>3</td>
<td>X'00000003'</td>
</tr>
<tr>
<td>MQXF_CONNX</td>
<td>4</td>
<td>X'00000004'</td>
</tr>
<tr>
<td>MQXF_DISC</td>
<td>5</td>
<td>X'00000005'</td>
</tr>
<tr>
<td>MQXF_OPEN</td>
<td>6</td>
<td>X'00000006'</td>
</tr>
<tr>
<td>MQXF_CLOSE</td>
<td>7</td>
<td>X'00000007'</td>
</tr>
<tr>
<td>MQXF_PUT1</td>
<td>8</td>
<td>X'00000008'</td>
</tr>
<tr>
<td>MQXF_PUT</td>
<td>9</td>
<td>X'00000009'</td>
</tr>
<tr>
<td>MQXF_GET</td>
<td>10</td>
<td>X'0000000A'</td>
</tr>
<tr>
<td>MQXF_DATA_CONV_ON_GET</td>
<td>11</td>
<td>X'0000000B'</td>
</tr>
<tr>
<td>MQXF_INQ</td>
<td>12</td>
<td>X'0000000C'</td>
</tr>
<tr>
<td>MQXF_SET</td>
<td>13</td>
<td>X'0000000D'</td>
</tr>
<tr>
<td>MQXF_BEGIN</td>
<td>14</td>
<td>X'0000000E'</td>
</tr>
<tr>
<td>MQXF_CMIT</td>
<td>15</td>
<td>X'0000000F'</td>
</tr>
<tr>
<td>MQXF_BACK</td>
<td>16</td>
<td>X'00000010'</td>
</tr>
</tbody>
</table>

#### MQXPDA_\* (API exit problem determination area)

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQXPDA_NONE</td>
<td>X'00...00'</td>
</tr>
<tr>
<td></td>
<td>(40 nulls)</td>
</tr>
</tbody>
</table>

For the C programming language, the following array version is also defined:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQXPDA_NONE_ARRAY</td>
<td>'\0','\0',...'\0','\0'</td>
</tr>
</tbody>
</table>

#### MQXR_\* (Exit reason)

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Hexadecimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQXR_BEFORE</td>
<td>1</td>
<td>X'00000001'</td>
</tr>
<tr>
<td>MQXR_AFTER</td>
<td>2</td>
<td>X'00000002'</td>
</tr>
<tr>
<td>MQXR_CONNECTION</td>
<td>3</td>
<td>X'00000003'</td>
</tr>
</tbody>
</table>
### MQXR2_* (Secondary exit response)

- **MQXR2_DEFAULT_CONTINUATION**: 0, `X'00000000'`
- **MQXR2_CONTINUE_CHAIN**: 8, `X'00000008'`
- **MQXR2_SUPPRESS_CHAIN**: 16, `X'00000010'`

### MQXT_* (Exit identifier)

- **MQXT_API_EXIT**: 2, `X'00000002'`

### MQXUA_* (Exit user area)

- **MQXUA_NONE**: `X'00...00' (16 nulls)`

For the C programming language, the following array version is also defined:

- **MQXUA_NONE_ARRAY**: `'\0','\0',...'\0','\0'`

### MQZAD_* (Authority data structure identifier)

- **MQZAD_STRUC_ID**: `'ZADb'`

For the C programming language, the following array version is also defined:

- **MQZAD_STRUC_ID_ARRAY**: `'Z','A','D','b'`

### MQZAD_* (Authority data version)

- **MQZAD_VERSION_1**: 1, `X'00000001'`
- **MQZAD_CURRENT_VERSION**: 1, `X'00000001'`

### MQZAET_* (Authority service entity type)

- **MQZAET_NONE**: 0, `X'00000000'`
- **MQZAET_PRINCIPAL**: 1, `X'00000001'`
- **MQZAET_GROUP**: 2, `X'00000002'`

### MQZAO_* (Authority service authorization type)

- **MQZAO_NONE**: 0, `X'00000000'`
- **MQZAO_CONNECT**: 1, `X'00000001'`
- **MQZAO_BROWSE**: 2, `X'00000002'`
- **MQZAO_INPUT**: 4, `X'00000004'`
- **MQZAO_OUTPUT**: 8, `X'00000008'`
- **MQZAO_INQUIRE**: 16, `X'00000010'`
### MQZAO_\_* (Authority service version)

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQZAS_VERSION_1</td>
<td>1</td>
<td>X'00000001'</td>
</tr>
<tr>
<td>MQZAS_VERSION_2</td>
<td>2</td>
<td>X'00000002'</td>
</tr>
<tr>
<td>MQZAS_VERSION_3</td>
<td>3</td>
<td>X'00000003'</td>
</tr>
<tr>
<td>MQZAS_VERSION_4</td>
<td>4</td>
<td>X'00000004'</td>
</tr>
</tbody>
</table>

### MQZCI_\_* (Continuation indicator)

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQZCI_DEFAULT</td>
<td>0</td>
<td>X'00000000'</td>
</tr>
<tr>
<td>MQZCI_CONTINUE</td>
<td>0</td>
<td>X'00000000'</td>
</tr>
<tr>
<td>MQZCI_STOP</td>
<td>1</td>
<td>X'00000001'</td>
</tr>
</tbody>
</table>

### MQZED_\_* (Entity descriptor structure identifier)

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQZED_STRUC_ID</td>
<td>'ZEDb'</td>
<td></td>
</tr>
</tbody>
</table>

For the C programming language, the following array version is also defined:

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQZED_STRUC_ID_ARRAY</td>
<td>'Z', 'E', 'D', 'b'</td>
<td></td>
</tr>
</tbody>
</table>

### MQZED_\_* (Entity descriptor version)

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQZED_VERSION_1</td>
<td>1</td>
<td>X'00000001'</td>
</tr>
<tr>
<td>MQZED_CURRENT_VERSION</td>
<td>1</td>
<td>X'00000001'</td>
</tr>
</tbody>
</table>

### MQZID_\_* (Function identifier, all services)

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQZID_INIT</td>
<td>0</td>
<td>X'00000000'</td>
</tr>
<tr>
<td>MQZID_TERM</td>
<td>1</td>
<td>X'00000001'</td>
</tr>
</tbody>
</table>
### MQZID_* (Function identifier, authority service)

<table>
<thead>
<tr>
<th>Function Identifier</th>
<th>Code</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQZID_INIT_AUTHORITY</td>
<td>0</td>
<td>X'00000000'</td>
</tr>
<tr>
<td>MQZID_TERM_AUTHORITY</td>
<td>1</td>
<td>X'00000001'</td>
</tr>
<tr>
<td>MQZID_CHECK_AUTHORITY</td>
<td>2</td>
<td>X'00000002'</td>
</tr>
<tr>
<td>MQZID_COPY_ALL_AUTHORITY</td>
<td>3</td>
<td>X'00000003'</td>
</tr>
<tr>
<td>MQZID_DELETE_AUTHORITY</td>
<td>4</td>
<td>X'00000004'</td>
</tr>
<tr>
<td>MQZID_SET_AUTHORITY</td>
<td>5</td>
<td>X'00000005'</td>
</tr>
<tr>
<td>MQZID_GET_AUTHORITY</td>
<td>6</td>
<td>X'00000006'</td>
</tr>
<tr>
<td>MQZID_GET_EXPLICIT_AUTHORITY</td>
<td>7</td>
<td>X'00000007'</td>
</tr>
<tr>
<td>MQZID_REFRESH_CACHE</td>
<td>8</td>
<td>X'00000008'</td>
</tr>
<tr>
<td>MQZID_ENUMERATE_AUTHORITY_DATA</td>
<td>9</td>
<td>X'00000009'</td>
</tr>
</tbody>
</table>

### MQZID_* (Function identifier, name service)

<table>
<thead>
<tr>
<th>Function Identifier</th>
<th>Code</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQZID_INIT_NAME</td>
<td>0</td>
<td>X'00000000'</td>
</tr>
<tr>
<td>MQZID_TERM_NAME</td>
<td>1</td>
<td>X'00000001'</td>
</tr>
<tr>
<td>MQZID_LOOKUP_NAME</td>
<td>2</td>
<td>X'00000002'</td>
</tr>
<tr>
<td>MQZID_INSERT_NAME</td>
<td>3</td>
<td>X'00000003'</td>
</tr>
<tr>
<td>MQZID_DELETE_NAME</td>
<td>4</td>
<td>X'00000004'</td>
</tr>
</tbody>
</table>

### MQZID_* (Function identifier, userid service)

<table>
<thead>
<tr>
<th>Function Identifier</th>
<th>Code</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQZID_INIT_USERID</td>
<td>0</td>
<td>X'00000000'</td>
</tr>
<tr>
<td>MQZID_TERM_USERID</td>
<td>1</td>
<td>X'00000001'</td>
</tr>
<tr>
<td>MQZID_FIND_USERID</td>
<td>2</td>
<td>X'00000002'</td>
</tr>
</tbody>
</table>

### MQZIO_* (Initialization options)

<table>
<thead>
<tr>
<th>Option</th>
<th>Code</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQZIO_PRIMARY</td>
<td>0</td>
<td>X'00000000'</td>
</tr>
<tr>
<td>MQZIO_SECONDARY</td>
<td>1</td>
<td>X'00000001'</td>
</tr>
</tbody>
</table>

### MQZNS_* (Name service version)

<table>
<thead>
<tr>
<th>Version</th>
<th>Code</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQZNS_VERSION_1</td>
<td>1</td>
<td>X'00000001'</td>
</tr>
</tbody>
</table>

### MQZSE_* (Start-enumeration indicator)

<table>
<thead>
<tr>
<th>Indicator</th>
<th>Code</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQZSE_START</td>
<td>1</td>
<td>X'00000001'</td>
</tr>
<tr>
<td>MQZSE_CONTINUE</td>
<td>0</td>
<td>X'00000000'</td>
</tr>
</tbody>
</table>

### MQZTO_* (Termination options)

<table>
<thead>
<tr>
<th>Option</th>
<th>Code</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQZTO_PRIMARY</td>
<td>0</td>
<td>X'00000000'</td>
</tr>
<tr>
<td>MQZTO_SECONDARY</td>
<td>1</td>
<td>X'00000001'</td>
</tr>
</tbody>
</table>

### MQZUS_* (Userid service version)

<table>
<thead>
<tr>
<th>Version</th>
<th>Code</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQZUS_VERSION_1</td>
<td>1</td>
<td>X'00000001'</td>
</tr>
</tbody>
</table>
WebSphere MQ constants
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 32 lists the system and default queue objects.
- Table 33 on page 428 lists the system and default channel objects.
- Table 34 on page 428 lists the system and default namelist objects.
- Table 35 on page 428 lists the system and default process objects.

Table 32. System and default objects: queues

<table>
<thead>
<tr>
<th>Object name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSTEM.ADMIN.CHANNEL.EVENT</td>
<td>Event queue for channels.</td>
</tr>
<tr>
<td>SYSTEM.ADMIN.COMMAND.QUEUE</td>
<td>Administration command queue. Used for remote MQSC commands and PCF commands.</td>
</tr>
<tr>
<td>SYSTEM.ADMIN.PERFM.EVENT</td>
<td>Event queue for performance events.</td>
</tr>
<tr>
<td>SYSTEM.ADMIN.QMGR.EVENT</td>
<td>Event queue for queue manager events.</td>
</tr>
<tr>
<td>SYSTEM.AUTH.DATA.QUEUE</td>
<td>The queue that holds access control lists for the queue manager.</td>
</tr>
<tr>
<td>SYSTEM.CHANNEL.INITQ</td>
<td>Channel initiation queue.</td>
</tr>
<tr>
<td>SYSTEM.CHANNEL.SYNCQ</td>
<td>The queue that holds the synchronization data for channels.</td>
</tr>
<tr>
<td>SYSTEM.CICS.INITIATION.QUEUE</td>
<td>Default CICS® initiation queue.</td>
</tr>
<tr>
<td>SYSTEM.CLUSTER.COMMAND.QUEUE</td>
<td>The queue used to carry messages to the repository queue manager.</td>
</tr>
<tr>
<td>SYSTEM.CLUSTER.REPOSITORY.QUEUE</td>
<td>The queue used to store all repository information.</td>
</tr>
<tr>
<td>SYSTEM.CLUSTER.TRANSMIT.QUEUE</td>
<td>The transmission queue for all messages to all clusters.</td>
</tr>
<tr>
<td>SYSTEM.DEAD.LETTER.QUEUE</td>
<td>Dead letter (undelivered message) queue.</td>
</tr>
<tr>
<td>SYSTEM.DEFAULT.ALIAS.QUEUE</td>
<td>Default alias queue.</td>
</tr>
<tr>
<td>SYSTEM.DEFAULT.INITIATION.QUEUE</td>
<td>Default initiation queue.</td>
</tr>
<tr>
<td>SYSTEM.DEFAULT.LOCAL.QUEUE</td>
<td>Default local queue.</td>
</tr>
<tr>
<td>SYSTEM.DEFAULT.MODEL.QUEUE</td>
<td>Default model queue.</td>
</tr>
<tr>
<td>SYSTEM.DEFAULT.REMOTE.QUEUE</td>
<td>Default remote queue.</td>
</tr>
<tr>
<td>SYSTEM.MQSC.REPLY.QUEUE</td>
<td>MQSC command reply-to queue. This is a model queue that creates a temporary dynamic queue for replies to remote MQSC commands.</td>
</tr>
<tr>
<td>SYSTEM.PENDING.DATA.QUEUE</td>
<td>Support deferred messages in JMS.</td>
</tr>
</tbody>
</table>
**Table 33. System and default objects: channels**

<table>
<thead>
<tr>
<th>Object name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSTEM.AUTO.RECEIVER</td>
<td>Default definition for an automatically defined receiver channel.</td>
</tr>
<tr>
<td>SYSTEM.AUTO.SVRCONN</td>
<td>Default definition for an automatically defined server connection channel.uegosjej🌎hsdhsdsds</td>
</tr>
<tr>
<td>SYSTEM.DEF.CLUSRCVR</td>
<td>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.</td>
</tr>
<tr>
<td>SYSTEM.DEF.CLUSSDR</td>
<td>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.</td>
</tr>
<tr>
<td>SYSTEM.DEF.RECEIVER</td>
<td>Default receiver channel.</td>
</tr>
<tr>
<td>SYSTEM.DEF.REQUESTER</td>
<td>Default requester channel.</td>
</tr>
<tr>
<td>SYSTEM.DEF.SENDER</td>
<td>Default sender channel.</td>
</tr>
<tr>
<td>SYSTEM.DEF.SERVER</td>
<td>Default server channel.</td>
</tr>
<tr>
<td>SYSTEM.DEF.SVRCONN</td>
<td>Default server connection channel.</td>
</tr>
<tr>
<td>SYSTEM.DEF.CLNTCONN</td>
<td>Default client connection channel.</td>
</tr>
</tbody>
</table>

**Table 34. System and default objects: namelists**

<table>
<thead>
<tr>
<th>Object name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSTEM.DEFAULT.NAMELIST</td>
<td>Default namelist.</td>
</tr>
</tbody>
</table>

**Table 35. System and default objects: processes**

<table>
<thead>
<tr>
<th>Object name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSTEM.DEFAULT.PROCESS</td>
<td>Default process definition.</td>
</tr>
</tbody>
</table>
Appendix B. Directory structure

Figure 30 on page 430 shows the general layout of the OSS files and directories for an installation with one queue manager. These files and directories are contained in the var directory for an installation. You select the path to the var directory when you install WebSphere MQ for HP NonStop Server. For more information about the location of the product files and directories, see the WebSphere MQ for HP NonStop Server, V5.3 Quick Beginnings.

The file and directory structure shown in Figure 30 on page 430 is typical of what might exist after the 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.
Table 36 describes the files and directories of a queue manager. By default, these files and directories are in the directory `/var/mqm/qmgrs/qmname/`, where `qmname` is the name of the queue manager that has been transformed into a valid directory name.

Table 36. The files and directories of a queue manager

<table>
<thead>
<tr>
<th>Description</th>
<th>File Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMQCLCHL.TAB</td>
<td>File containing the client channel definition table.</td>
</tr>
<tr>
<td>AMQRFCD.A.DAT</td>
<td>File containing the definitions of channels other than client connection channels.</td>
</tr>
<tr>
<td>auth/</td>
<td>This directory is not used.</td>
</tr>
</tbody>
</table>
Table 36. The files and directories of a queue manager (continued)

<table>
<thead>
<tr>
<th>Directory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dce/</td>
<td>This directory is not used.</td>
</tr>
<tr>
<td>ECprocessName</td>
<td>While the queue manager is running, this file contains the NonStop OS process name of the execution controller of the queue manager.</td>
</tr>
</tbody>
</table>
| errors/   | Directory containing the error log files for the queue manager:  
  - AMQERR01.LOG  
  - AMQERR02.LOG  
  - AMQERR03.LOG |
| esem/     | Directory containing files that are used internally. |
| G         | Link to the queue manager’s subvolume. |
| isem/     | Directory containing files that are used internally. |
| msem/     | Directory containing files that are used internally. |
| plugcomp/ | Empty directory reserved for use by installable services. |
| qm.ini    | The queue manager configuration file. |
| qmproc.ini| The process management rules configuration file. |
| shmem/    | Directory containing files that are used internally. |
| ssem/     | Directory containing files that are used internally. |
| ssl/      | This directory contains three files used by the SSL support:  
  - The certificate store  
  - The pass phrase stash file  
  - The certificate revocation list file (optional) |
| startprm/ | Directory containing temporary files that are used internally. |
| zsocketapp| This directory is not used. |
| zsocketEC | This directory is not used. |
| @ipcc/    |  
  - esem/ Directory containing files that are used internally.  
  - isem/ Directory containing files that are used internally.  
  - msem/ Directory containing files that are used internally.  
  - shmem/ Directory containing files that are used internally.  
  - ssem/ Directory containing files that are used internally. |
| @qmpersist|  
  - esem/ Directory containing files that are used internally.  
  - isem/ Directory containing files that are used internally.  
  - msem/ Directory containing files that are used internally.  
  - shmem/ Directory containing files that are used internally.  
  - ssem/ Directory containing files that are used internally. |
Directory structure
Appendix C. Instrumentation events and event messages

This chapter:

- Provides a brief introduction to WebSphere MQ instrumentation events, which you can use to monitor the operation of queue managers. See "WebSphere MQ instrumentation events." For detailed information about instrumentation events, see WebSphere MQ Event Monitoring.
- Describes the use of Event Management Service (EMS) events by WebSphere MQ for HP NonStop Server. See "Event Management Service (EMS) events" on page 435.

WebSphere MQ instrumentation events

Instrumentation events cause event messages to be generated when a queue manager detects a predefined set of conditions. For example, a Queue Full event results from the following conditions:

- Queue Full events are enabled for a specified queue.
- An application issues an MQPUT call to put a message on that queue, but the call fails because the queue is full.

Other conditions that can cause instrumentation events include:

- A limit on the number of messages on a queue being reached
- A queue not being serviced within a specified time
- A channel instance being started or stopped
- An application attempting to open a queue specifying a user ID that is not authorized

With the exception of channel events, you must enable all instrumentation events before they can be generated.

The event message contains information about the conditions resulting in the event. The event message is put onto an event queue. An application can retrieve the event message from this queue for analysis.

If you define event queues as remote queues, you can put all the event queues on a single queue manager (for those nodes that support instrumentation events). You can then use the events generated to monitor a network of queue managers from a single node.

Types of event

There are three types of instrumentation event:

Queue manager events

Queue manager events are related to the definitions of resources within queue managers. For example, a queue manager event can be generated when an application attempts to put a message to a queue that does not exist.

Performance events

Performance events are notifications that a threshold has been reached by a resource. For example, a performance event can be generated when a
Instrumentation events

queue-depth limit has been reached or, following an MQGET call, if a queue has not been serviced within a predefined time.

Channel events
Channel events are reported by channels as a result of conditions detected during their operation. For example, a channel event can be generated when a channel instance is stopped.

Event notification through event queues
When an event occurs, the queue manager puts an event message on the appropriate event queue, if defined. The event message contains information about the event that you can retrieve by writing a suitable MQI application program that:

- Gets the message from the queue.
- Processes the message to extract the event data. For a description of event message formats, see [WebSphere MQ Event Monitoring](#).

Each category of event has its own event queue. All events in that category result in an event message being put onto the same queue.

<table>
<thead>
<tr>
<th>This event queue...</th>
<th>Contains messages from...</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSTEM.ADMIN.QMGR.EVENT</td>
<td>Queue manager events</td>
</tr>
<tr>
<td>SYSTEM.ADMIN.PERFM.EVENT</td>
<td>Performance events</td>
</tr>
<tr>
<td>SYSTEM.ADMIN.CHANNEL.EVENT</td>
<td>Channel events</td>
</tr>
</tbody>
</table>

Using triggered event queues
You can set up the event queues with triggers so that when an event is generated, the event message put onto the event queue starts a user-written monitoring application. This application can process the event messages and take appropriate action. For example, some events can require that an operator be informed, and others can start an application that performs various administration tasks automatically.

Enabling instrumentation events
How you enable an instrumentation event depends on the event type:

- Queue manager events are enabled by setting attributes on the queue manager.
- Performance events as a whole must be enabled on the queue manager. You must also enable specific performance events by setting the appropriate queue attribute, and identify the conditions, such as a queue-depth-high limit, that result in the event.
- Channel events occur automatically; they do not need to be enabled. If you do not want to monitor channel events, you can put-inhibit the channel event queue.

You enable and disable the generation of instrumentation events using any of the following:

- MQSC commands. For more information, see the [WebSphere MQ Script (MQSC) Command Reference](#).
- PCF commands for queue managers. For more information, see [WebSphere MQ Programmable Command Formats and Administration Interface](#).
- MQAI commands. For more information, see [WebSphere MQ Programmable Command Formats and Administration Interface](#).
Event messages

Event messages contain information relating to the origin of an event, including the type of event, the name of the application that caused the event, and, for performance events, a short statistics summary for the queue.

The format of event messages is similar to that of PCF response messages. The message data can be retrieved from event messages by user-written administration programs using the data structures described in WebSphere MQ Programmable Command Formats and Administration Interface.

Event Management Service (EMS) events

WebSphere MQ for HP NonStop Server provides a centralized operator message logging subsystem called the Event Management Service (EMS). EMS is a required subsystem and runs continuously. EMS collects and stores event messages from running system and application software and provides tools and services for system administrators to format filter, view, and respond to the events as appropriate. EMS event messages use a standard format and NonStop OS provides services to receive and process EMS event messages in a user-written monitoring and control application.

WebSphere MQ generates EMS event messages that correspond to the WebSphere MQ queue manager events, channel events, and performance events. EMS messages might also be generated that correspond to the message entries in the WebSphere MQ logs and to FFSTs. These event messages can alert system operators and administrators to software conditions that can have an adverse effect on the WebSphere MQ operating environment.

EMS template files supplied with WebSphere MQ for HP NonStop Server

In order for NonStop OS to format and display EMS event messages generated by WebSphere MQ, you must integrate the EMS template files supplied with WebSphere MQ with the EMS templates for other subsystems. For detailed information about the procedures for doing this, refer to the HP EMS Manual.

The following files are supplied in the NonStop OS headers and include files subvolume:

ZMQSTMPL (file code 839)
An EMS template object file containing the formatting templates for the EMS events generated by WebSphere MQ.

ZMQSDDL (file code 101)
The Data Definition Language schema for the EMS events generated by WebSphere MQ.

ZMQSC (file code 101)
Compiled output (C) from the DDL compiler of definitions of the EMS events generated by the product.

ZMQSCOB (file code 101)
Compiled output (COBOL) from the DDL compiler of definitions of the EMS events generated by the product.

ZMQSTACL (file code 101)
Compiled output (TACL) from the DDL compiler of definitions of the EMS events generated by the product.
This subvolume also contains the EMS template source file AMQSTMPL from which the template object file ZMQSTMPL is generated. The file ZMQSTMPL is ready for integration with your system’s event templates using the template installer program (TEMPLI) and SYSGEN. The source of the event templates is supplied, so that you can modify the formatting of the events when they are used in your environment.

For example, you might not be interested in displaying all of the information that is contained in an event, or you might want to add or change text that is displayed along with the information in the event. See the HP EMS Manual for a description of the EMS event template source language, and for the procedures used to compile the definitions to produce a customized ZMQSTMPL file.

### Integrating the WebSphere MQ EMS event templates

The template object file must be integrated into your system’s resident and nonresident EMS template files, so that programs such as VIEWPOINT and EMSDIST can format and display WebSphere MQ EMS events.

A procedure for integrating the WebSphere MQ EMS templates into the system templates is described in the remainder of this section. Note that different procedures might be preferred in your installation.

1. Determine the name of the current system template using the SCF command
   ```
   SCF;ASSUME SUBSYS $ZZKR;INFO
   ``
   NonStop Kernel - Info SUBSYS \HAWK.$ZZKR
   Current Settings
   *DAYLIGHT_SAVING_TIME ................ USA66
   *NONRESIDENT_TEMPLATES................ $SYSTEM.SYS01.TEMPLATE
   *POWERFAIL_DELAY_TIME.................. 30
   *RESIDENT_TEMPLATES................... $SYSTEM.SYS01.RTMPLATE
   SUPER_SUPER_IS_UNDENIABLE............. OFF
   *SYSTEM_NAME.......................... \HAWK
   *SYSTEM_NUMBER......................... 2
   SYSTEM_PROCESSOR_TYPE ................ NSR-W
   *TIME_ZONE_OFFSET..................... -05:00
   Pending Changes (will take effect at next system load)
   None Total Errors = 0 Total Warnings = 0

2. Run the TEMPLI compiler to create new system template files combining the current system templates with the new WebSphere MQ templates. This is a two-step process:
   a. Create a text file containing the following commands:

   ```
   FILE <current NONRESIDENT system template file>
   FILE <New WebSphere MQ template file>
   EXIT
   ```
Defining the PARAM MQEMSEVENTS

To complete the enablement of WebSphere MQ EMS events, you must ensure that the environment variable MQEMSEVENTS is defined in the context of WebSphere MQ processes. The value is a three-character string, which is a decimal value interpreted as a bit map, as follows:

<table>
<thead>
<tr>
<th>EMS message</th>
<th>Bit-map entry</th>
<th>MQEMSEVENT value</th>
</tr>
</thead>
<tbody>
<tr>
<td>FFST</td>
<td>0x00000001</td>
<td>1</td>
</tr>
<tr>
<td>START / STOP</td>
<td>0x00000002</td>
<td>2</td>
</tr>
<tr>
<td>PERFORMANCE</td>
<td>0x00000004</td>
<td>4</td>
</tr>
<tr>
<td>CHANNEL</td>
<td>0x00000008</td>
<td>8</td>
</tr>
<tr>
<td>QUEUE MANAGER</td>
<td>0x00000010</td>
<td>16</td>
</tr>
<tr>
<td>MESSAGE</td>
<td>0x00000020</td>
<td>32</td>
</tr>
<tr>
<td>ERROR</td>
<td>0x00000040</td>
<td>64</td>
</tr>
<tr>
<td>ALL</td>
<td>0x0000007F</td>
<td>127</td>
</tr>
</tbody>
</table>

Note: To make this change permanent (so that the change survives a cold load of your system), you must update the system using SYSGEN.

For further information about EMS templates, see the HP DSM Template Services Manual. This book also describes how to use SYSGEN to perform this task.
EMS events

Thus, to switch on all EMS events for WebSphere MQ, you must define the following environment variable or TACL PARAM in the environment from which any administration commands are issued.

For TACL:

```
PARAM MQEMSEVENTS 127
```

For OSS:

```
export MQEMSEVENTS 127
```

This definition is also required in server class definitions of all server classes for WebSphere MQ. You can configure each server class with different options. See “Changing configuration information in Pathway” on page 115 for more information.

By default no EMS events are generated, that is, the environment variable or PARAM MQEMSEVENTS are not defined.

Using an alternative collector

On a NonStop OS system, the default EMS event collector is called $0 and is always present. All EMS events generated by a WebSphere MQ queue manager are sent to the default collector. If you want a different collector to collect EMS events for a queue manager, define the environment variable MQEMSCOLLECTOR in the environment of all processes of the queue manager. The value of the MQEMSCOLLECTOR environment variable is the name of the alternate collector.

Writing programs to process WebSphere MQ EMS events

You can write an application to monitor a WebSphere MQ queue manager by processing EMS event messages. This type of application also affect the operation of the queue manager by issuing PCF commands in response to the EMS event messages generated.

The ZMQSC, ZMQSTAL, ZMQSCOB, and ZMQSTACL files supplied with the WebSphere MQ in the NonStop OS headers and include files subvolume define the tokens contained in the WebSphere MQ EMS event messages in C, TAL, COBOL, and TACL. These definitions can be used by an administration program to understand the format of the messages.
Appendix D. Stopping and removing queue managers manually

If the normal methods for stopping and removing queue managers fail, try the methods described here.

Stopping a queue manager manually

The normal method of stopping queue managers, using the `endmqm` command, should work even in the event of failures within the queue manager. In exceptional circumstances, if this method of stopping a queue manager fails, use the following procedure to stop it manually:

1. Find the process IDs of the queue manager programs that are still running. Use the FUP LISTOPENS command on the AMQSHUTD file in the queue manager’s data subvolume to find out the process names that belong to the queue manager.
2. End the queue manager processes that are still running. Use the STOP command together with the process IDs found in the previous step.
   a. MQECSVR
   b. Any other processes that are still running
3. After you have ended all processes, delete the ECprocessName file located in the queue manager’s OSS directory. Determine this directory from the entry for the queue manager in the mqs.ini file located in `var_installation_path/var/mqm`. The Prefix entry for the queue manager gives the location of the installation’s qmgrs directory. The Directory prefix specifies the directory containing the queue manager files on OSS.
4. Modify the qmstatus.ini file, which is also located in this directory as follows:
   
   ```
   QueueManagerStatus:
   CurrentStatus=Ended
   ```

Note: If you stop the queue manager manually, FFSTs might be taken and FD files produced. This is not a defect in the queue manager. The queue manager should restart normally even if it was ended using the preceding method.

If you want to delete the queue manager after stopping it manually, use the `dltmqm` command as normal. Then use the following command to ensure that any listeners associated with the queue manager are also ended:

```
endmqlsr -m queue_manager_name
```

If, for some reason, the `dltmqm` command fails to delete the queue manager, you can use the manual process detailed in the following section.

Removing a queue manager manually

To remove a queue manager manually:

1. Ensure that there are no queue manager processes running for the queue manager you want to remove.
2. Locate the entry for the queue manager in the mqs.ini file in `var_installation_path/var/mqm`. 
Removing a queue manager manually

3. Determine the queue manager’s NonStop OS subvolume (identified by the HPNSSGuardianSubvol stanza), then delete all files in that subvolume using the FUP PURGE command. For example:

   FUP PURGE $VOL.QMSVOL*.*

4. Determine the queue manager’s OSS directory (identified by the Directory stanza), then delete that directory tree from var_installation_path/var/mqm/qmgs. For example, for a queue manager TEST with Directory set to TEST:

   rm -rf var_installation_path/var/mqm/qmgs/TEST

5. Delete the QueueManager stanza from the mqs.ini file.
Appendix E. Comparing command sets

The tables in this appendix compare the facilities available from the different administration command sets.

Commands for queue manager administration

Table 37. Commands for queue manager administration

<table>
<thead>
<tr>
<th>Description</th>
<th>PCF command</th>
<th>MQSC command</th>
<th>Control command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Change Queue Manager</td>
<td>ALTER QMGR</td>
<td>No equivalent</td>
<td>No equivalent</td>
</tr>
<tr>
<td>(Create queue manager)</td>
<td></td>
<td>No equivalent</td>
<td>crtmqm</td>
</tr>
<tr>
<td>(Delete queue manager)</td>
<td></td>
<td>No equivalent</td>
<td>dltmqm</td>
</tr>
<tr>
<td>(Display queue managers)</td>
<td></td>
<td>No equivalent</td>
<td>dspmq</td>
</tr>
<tr>
<td>Inquire Queue Manager</td>
<td>DISPLAY QMGR</td>
<td>No equivalent</td>
<td>No equivalent</td>
</tr>
<tr>
<td>(Stop queue manager)</td>
<td>No equivalent</td>
<td>endmqm</td>
<td></td>
</tr>
<tr>
<td>Ping Queue Manager</td>
<td>PING QMGR</td>
<td>No equivalent</td>
<td>No equivalent</td>
</tr>
<tr>
<td>(Start queue manager)</td>
<td>No equivalent</td>
<td>strmqm</td>
<td></td>
</tr>
</tbody>
</table>

Notes:
1. Not available as a PCF command

Commands for command server administration

Table 38. Commands for command server administration

<table>
<thead>
<tr>
<th>Description</th>
<th>PCF command</th>
<th>MQSC command</th>
<th>Control command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Display command server</td>
<td>No equivalent</td>
<td>No equivalent</td>
<td>dspmqcsv</td>
</tr>
<tr>
<td>Start command server</td>
<td>No equivalent</td>
<td>No equivalent</td>
<td>strmqcsv</td>
</tr>
<tr>
<td>Stop command server</td>
<td>No equivalent</td>
<td>No equivalent</td>
<td>endmqcsv</td>
</tr>
</tbody>
</table>

Commands for queue administration

Table 39. Commands for queue administration

<table>
<thead>
<tr>
<th>Description</th>
<th>PCF command</th>
<th>MQSC command</th>
<th>Control command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Change Queue</td>
<td>ALTER QLOCAL</td>
<td>ALTER QALIAS</td>
<td>No equivalent</td>
</tr>
<tr>
<td></td>
<td>ALTER QMODEL</td>
<td>ALTER QREMOTE</td>
<td></td>
</tr>
<tr>
<td>Clear Queue</td>
<td>CLEAR QUEUE</td>
<td>No equivalent</td>
<td>No equivalent</td>
</tr>
<tr>
<td>Copy Queue</td>
<td>DEFINE QLOCAL(x)</td>
<td>DEFINE QALIAS(x)</td>
<td>No equivalent</td>
</tr>
<tr>
<td></td>
<td>LIKE(y)</td>
<td>LIKE(y)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DEFINE QMODEL(x)</td>
<td>DEFINE QREMOTE(x)</td>
<td>No equivalent</td>
</tr>
<tr>
<td></td>
<td>LIKE(y)</td>
<td>LIKE(y)</td>
<td></td>
</tr>
</tbody>
</table>
### Comparing command sets

**Table 39. Commands for queue administration (continued)**

<table>
<thead>
<tr>
<th>PCF command</th>
<th>MQSC command</th>
<th>Control command</th>
</tr>
</thead>
</table>
| Create Queue      | DEFINE QLOCAL
|                   | DEFINE QALIAS
|                   | DEFINE QMODEL
|                   | DEFINE QREMOTE                                                             | No equivalent            |
| Delete Queue      | DELETE QLOCAL
|                   | DELETE QALIAS
|                   | DELETE QMODEL
|                   | DELETE QREMOTE                                                             | No equivalent            |
| Inquire Queue     | DISPLAY QUEUE                                                               | No equivalent            |
| Inquire Queue Names | DISPLAY QUEUE                                                           | No equivalent            |

---

### Commands for process administration

**Table 40. Commands for process administration**

<table>
<thead>
<tr>
<th>PCF command</th>
<th>MQSC command</th>
<th>Control command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Change Process</td>
<td>ALTER PROCESS</td>
<td>No equivalent</td>
</tr>
<tr>
<td>Copy Process</td>
<td>DEFINE PROCESS(x) LIKE(y)</td>
<td>No equivalent</td>
</tr>
<tr>
<td>Create Process</td>
<td>DEFINE PROCESS</td>
<td>No equivalent</td>
</tr>
<tr>
<td>Delete Process</td>
<td>DELETE PROCESS</td>
<td>No equivalent</td>
</tr>
<tr>
<td>Inquire Process</td>
<td>DISPLAY PROCESS</td>
<td>No equivalent</td>
</tr>
<tr>
<td>Inquire Process Names</td>
<td>DISPLAY PROCESS</td>
<td>No equivalent</td>
</tr>
</tbody>
</table>

---

### Commands for channel administration

**Table 41. Commands for channel administration**

<table>
<thead>
<tr>
<th>PCF command</th>
<th>MQSC command</th>
<th>Control command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Change Channel</td>
<td>ALTER CHANNEL</td>
<td>No equivalent</td>
</tr>
<tr>
<td>Copy Channel</td>
<td>DEFINE CHANNEL(x) LIKE(y)</td>
<td>No equivalent</td>
</tr>
<tr>
<td>Create Channel</td>
<td>DEFINE CHANNEL</td>
<td>No equivalent</td>
</tr>
<tr>
<td>Delete Channel</td>
<td>DELETE CHANNEL</td>
<td>No equivalent</td>
</tr>
<tr>
<td>End Listener</td>
<td>No equivalent</td>
<td>endmqqlsr</td>
</tr>
<tr>
<td>Inquire Channel</td>
<td>DISPLAY CHANNEL</td>
<td>No equivalent</td>
</tr>
<tr>
<td>Inquire Channel Names</td>
<td>DISPLAY CHANNEL</td>
<td>No equivalent</td>
</tr>
<tr>
<td>Ping Channel</td>
<td>PING CHANNEL</td>
<td>No equivalent</td>
</tr>
<tr>
<td>Reset Channel</td>
<td>RESET CHANNEL</td>
<td>No equivalent</td>
</tr>
<tr>
<td>Resolve Channel</td>
<td>RESOLVE CHANNEL</td>
<td>No equivalent</td>
</tr>
<tr>
<td>Start Channel</td>
<td>START CHANNEL</td>
<td>runmqchl</td>
</tr>
<tr>
<td>Start Channel Initiator</td>
<td>START CHINIT</td>
<td>runmqchi</td>
</tr>
<tr>
<td>Start Channel Listener</td>
<td>START LISTENER</td>
<td>runmqqlsr</td>
</tr>
<tr>
<td>Stop Channel</td>
<td>STOP CHANNEL</td>
<td>No equivalent</td>
</tr>
</tbody>
</table>
### Other control commands

**Table 42. Other control commands**

<table>
<thead>
<tr>
<th>Description</th>
<th>PCF command</th>
<th>MQSC command</th>
<th>Control command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alter WebSphere MQ object attributes</td>
<td>No equivalent</td>
<td>No equivalent</td>
<td>altmqfls</td>
</tr>
<tr>
<td>Alter WebSphere MQ user information</td>
<td>No equivalent</td>
<td>No equivalent</td>
<td>altmqusr</td>
</tr>
<tr>
<td>Create code for data conversion exit</td>
<td>No equivalent</td>
<td>No equivalent</td>
<td>crtmqcvx</td>
</tr>
<tr>
<td>Dump authority</td>
<td>No equivalent</td>
<td>No equivalent</td>
<td>dmpmqaut</td>
</tr>
<tr>
<td>Display authority</td>
<td>No equivalent</td>
<td>No equivalent</td>
<td>dspmqaut</td>
</tr>
<tr>
<td>Display WebSphere MQ object attributes</td>
<td>No equivalent</td>
<td>No equivalent</td>
<td>dspmqfls</td>
</tr>
<tr>
<td>Display formatted trace</td>
<td>No equivalent</td>
<td>No equivalent</td>
<td>dspmqtrc</td>
</tr>
<tr>
<td>Display WebSphere MQ user information</td>
<td>No equivalent</td>
<td>No equivalent</td>
<td>dspmqusr</td>
</tr>
<tr>
<td>End trace</td>
<td>No equivalent</td>
<td>No equivalent</td>
<td>endmqtrc</td>
</tr>
<tr>
<td>Run dead letter queue handler</td>
<td>No equivalent</td>
<td>No equivalent</td>
<td>runmqdlq</td>
</tr>
<tr>
<td>Run MQSC commands</td>
<td>No equivalent</td>
<td>No equivalent</td>
<td>runmqsc</td>
</tr>
<tr>
<td>Run trigger monitor</td>
<td>No equivalent</td>
<td>No equivalent</td>
<td>runmqtrm</td>
</tr>
<tr>
<td>Grant or revoke authority</td>
<td>No equivalent</td>
<td>No equivalent</td>
<td>setmqaut</td>
</tr>
<tr>
<td>Start trace</td>
<td>No equivalent</td>
<td>No equivalent</td>
<td>strmqtrc</td>
</tr>
</tbody>
</table>
Comparing command sets
Appendix F. Environment variables

WebSphere MQ creates and uses a number of NonStop OS environment variables (PARAMs in NonStop OS). The MQNSKVARPATH and MQNSKOPTPATH variables must be present in the environment of all programs. If you are using WebSphere MQ applications and control commands, you must ensure that the TACLs, Pathway configurations, and OSS shells specify these variables.

Include the PARAM statements in your TACLCSTM files and export the corresponding environment variables in any OSS shell you use.

**MQNSKVARPATH**
Specifies the location of the WebSphere MQ databases and configuration files. For example, /home/mq/var/mqm.

For example:
```
export MQNSKVARPATH=/home/mq/var/mqm
```

The export command exports the variable to the shell environment and makes the variable global in scope.

**MQNSKOPTPATH**
Specifies the location of the WebSphere MQ product executables and libraries. For example, /home/mq/opt/mqm.

For example:
```
export MQNSKOPTPATH=/home/mq/opt/mqm
```

The export command exports the variable to the shell environment and makes the variable global in scope.

**MQCONNECTTYPE**
You can use this variable, if present, to disable the ability of applications to use FASTPATH connections. If this variable is set to the value STANDARD, applications use STANDARD connections even if they request FASTPATH. Any other value is treated as if the variable was not specified.

**MQEMSEVENTS**
This variable enables WebSphere MQ EMS Events. For example, specify the following to switch on all EMS events for WebSphere MQ:
```
PARAM MQEMSEVENTS 127
```

**MQSNOAUT**
If this variable is set to Yes when the `crtmqm` command is run, the new queue manager is created with the OAM disabled.

**SAVE-ENVIRONMENT**
Required when running non-native TAL and COBOL applications to pass the environment to the Common Runtime Environment (CRE). Set SAVE-ENVIRONMENT to ON.
Environment variables

Queue server tuning parameters

You can use the following PARAMs to override the built-in defaults of the queue server for various housekeeping operations. Define these PARAMs in the TACL environment of a queue server:

MQQSHKEEPINT
If this PARAM is set, you can specify a numeric value in seconds to override the default housekeeping interval of the queue server. The default interval is 60 seconds. The housekeeping interval controls the frequency that the queue server looks at queues to detect expired messages and to examine memory utilization to optimize operations.

MQQSSIGTIMEOUT
If this PARAM is set, you can specify a numeric value in seconds to override the default timeout for the delivery of a signal IPC to an application that has initiated an MQGET with the MQGMO_SET_SIGNAL option. The default timeout is 60 seconds. If a queue server is unable to deliver the signal within this timeout (after conditions for generating the signal have been met) the queue server logs the fact and then cancels the signal.

MQQSMAXBATCHEXPRI
If this PARAM is set, you can specify a numeric value to override the default maximum number of expired persistent messages that are discarded within a single transaction during housekeeping by a queue server. The default maximum is 100. When persistent messages expire, they must be physically removed from the queue database.

MQQSMAXMSGSEXPI
If this PARAM is set, you can specify a numeric value to override the default maximum number of expired messages that are detected and discarded within a single housekeeping instance of a queue server. The default maximum is 300.
Appendix G. Application programming reference

The following sections are new to WebSphere MQ for HP NonStop Server and you should use them in conjunction with the WebSphere MQ Application Programming Reference.

Structure data types

This section describes the MQI structure data types supported by WebSphere MQ for HP NonStop Server.

<table>
<thead>
<tr>
<th>Structure data type</th>
<th>Supported in Version 5.1</th>
<th>Supported in Version 5.3</th>
<th>For Version 5.3, works as described in the WebSphere MQ Application Programming Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQBO – Begin Options</td>
<td>No</td>
<td>No</td>
<td>Not applicable</td>
</tr>
<tr>
<td>MQCH – CICS Bridge Header</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>MQCNO – Connect Options</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes, but with some additional notes. See &quot;MQCNO – Connect Options&quot; on page 448 for more information.</td>
</tr>
<tr>
<td>MQDH – Distribution Header</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>MQDLH – Dead Letter Header</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>MQGMO – Get Message Options</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes, but with some additional notes. See &quot;MQGMO – Get Message Options&quot; on page 448 for more information.</td>
</tr>
<tr>
<td>MQIH – IMS™ Bridge Header</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>MQMD – Message Descriptor</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes, but with some additional notes. See &quot;MQMD – Message Descriptor&quot; on page 449 for more information.</td>
</tr>
<tr>
<td>MQMDE – Message Descriptor Extension</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>MQOD – object Descriptor</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>MQOR – Object Record</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>MQPMO – Put Message Options</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes, but with some additional notes. See &quot;MQPMO – Put Message Options&quot; on page 449 for more information.</td>
</tr>
<tr>
<td>MQPMR – Put Message Record</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>MQRMH – Message Reference Header</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Structure data types

<table>
<thead>
<tr>
<th>Structure data type</th>
<th>Supported in Version 5.1</th>
<th>Supported in Version 5.3</th>
<th>For Version 5.3, works as described in the WebSphere MQ Application Programming Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQRR – Response Record</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>MQTM – Trigger Message</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>MQTM2 – Trigger Message Character Format</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>MQWH – Workload Information Header</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>MQXQH – Transmission Queue Header</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

This section describes the following WebSphere MQ structure data types:

**MQCNO – Connect Options**

The MQCNO data structure is as specified in WebSphere MQ Application Programming Reference, with the following additional notes:

- The unit of execution is defined as a thread on OSS and a process on NonStop OS.
- MQCNO_FASTPATH_BINDING can be used only in a process that has a single connection to a queue manager.
- MQCNO_FASTPATH_BINDING requires that the application is run under the user ID associated with the principal mqm.
- You can use the OSS environment variable or NonStop OS parameter, MQCONNECTTYPE in association with the bind type specified by the Options field, to control the type of binding used. If you specify this parameter, it should have the value FASTPATH or STANDARD. If it has some other value, it is ignored. The value of the parameter is case sensitive.
- WebSphere MQ for HP NonStop Server supports up to MCNO_VERSION_4 but the ClientConnOffset, ClientConnPtr, ConnTag, SSLConfigPtr, and SSLConfigOffset fields are ignored.

**MQGMO – Get Message Options**

The MQGMO structure is an input and output parameter of the MQGET call. Note the following information about the MQGMO_SET_SIGNAL, MQGMO_WAIT, MQGMO_SYNCPOINT, and MQGMO_NO_SYNCPOINT options in WebSphere MQ for HP NonStop Server:

- If you are writing a NonStop OS application and want the application to proceed with other work while waiting for a message to arrive, consider using the signal option MQGMO_SET_SIGNAL instead of MQGMO_WAIT. However, the signal option is environment specific and should not be used by applications that are to be ported between different environments.
- If there is more than one MQGET call waiting for the same message with a mixture of wait and signal options, each waiting call is considered equally. It is an error to specify MQGMO_SET_SIGNAL with MQGMO_WAIT. It is also an error to specify this option with a queue handle for which a signal is outstanding.
• If an application specifies MQGET with MQGMO_SET_SIGNAL and a WaitInterval of 0, the MQGMO_SET_SIGNAL option is ignored and treated as an MQGET with MQGMO_NO_WAIT.

This means that an application must be prepared to receive MQRC_NO_MSG_AVAILABLE on an MQGET with MQGMO_SET_SIGNAL if the WaitInterval can ever be zero. Applications receive a signal IPC only if:
  – The application experiences MQRC_SIGNAL_REQUEST_ACCEPTED from the MQGET (indicates that a signal has been posted).
  – The application has been able to process the file_open_system message and accept the signal IPC within the queue server’s timeout for signal delivery. This is 60 seconds by default, but you can override this default for a queue server by specifying the MQQSSIGTIMEOUT PARAM in the environment of the queue server.

The queue manager logs the failure to deliver an IPC message to an application if it has not been able to open the process and send the IPC before the timeout expires. At this point the queue manager does not attempt delivery again. Applications should be resilient to this by not waiting indefinitely for an IPC signal.

• If neither of the options MQGMO_SYNCPOINT or MQGMO_NO_SYNCPOINT is set, WebSphere MQ for HP NonStop Server defaults to MQGMO_SYNCPOINT.

• WebSphere MQ for HP NonStop Server does not support the MsgToken field.

**MQMD – Message Descriptor**

The MQMD structure contains the control information that describes a message. Note the following information:

• The BackoutCount functions as described in the *WebSphere MQ Application Programming Reference*. This is a count of the number of times the message has been previously returned by the MQGET call as part of a unit of work and subsequently backed out. It is provided as an aid to the application in detecting processing errors that are based on message content.

• In WebSphere MQ for HP NonStop Server, the discarding of a message (and generation of a report, if required) is not performed during an MQGET call. Instead it is under the control of the queue server that performs this function periodically, according to settings for the queue manager.

• The value of the UserIdentifier field, when set by the queue manager during an MQPUT or MQPUT1 is the WebSphere MQ Principal name. This name is found in the queue manager’s Principal database and corresponds to the effective user ID of the application.

**MQPMO – Put Message Options**

The MQPMO structure is an input and output parameter of the MQPUT and MQPUT1 calls. If neither of the options MQPMO_SYNCPOINT or MQPMO_NO_SYNCPOINT is set, WebSphere MQ for HP NonStop Server defaults to MQPMO_SYNCPOINT.
This section describes the MQI calls supported by WebSphere MQ for HP NonStop Server.

<table>
<thead>
<tr>
<th>MQI Call Description</th>
<th>Supported in Version 5.1</th>
<th>Supported in Version 5.3</th>
<th>For Version 5.3, works as described in the WebSphere MQ Application Programming Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQBACK – Back Out Changes</td>
<td>Returns error</td>
<td>Yes</td>
<td>Yes, but for local units of work only.</td>
</tr>
<tr>
<td>MQBEGIN – Begin Unit of Work</td>
<td>Returns error</td>
<td>Returns error</td>
<td>No</td>
</tr>
<tr>
<td>MQCLOSE – Close object</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes, but with some additional notes. See “MQCLOSE – Close Object” on page 451 for more information.</td>
</tr>
<tr>
<td>MQCMIT – Commit changes</td>
<td>Returns error</td>
<td>Yes</td>
<td>Yes, but for local units of work only.</td>
</tr>
<tr>
<td>MQCONN – Connect queue manager</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>MQCONNX – Connect Queue manager (Extended)</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>MQDISC – Disconnect queue manager</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes, but with some additional notes. See “MODISC – Disconnect queue manager” on page 451 for more information.</td>
</tr>
<tr>
<td>MQGET – Get Message</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>MQINQ – Inquire About Object Attributes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes, but with some additional notes. See “MQINQ – Inquire about object attributes” on page 451 for more information.</td>
</tr>
<tr>
<td>MQOPEN – Open object</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes, but with some additional notes. See “MQOPEN – Open Object” on page 451 for more information.</td>
</tr>
<tr>
<td>MQPUT – Put Message</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>MQPUT1 – Put One Message</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>MQSET – Set Object Attributes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes, but with some additional notes. See “MQSET – Set Object Attributes” on page 451 for more information.</td>
</tr>
<tr>
<td>MQSYNC – Synchronize Statistics Updates</td>
<td>Returns error</td>
<td>No</td>
<td></td>
</tr>
</tbody>
</table>
Notes:
1. The MQI call can be issued by the application, but always returns completion code MQCC_FAILED and reason code MQRC_ENVIRONMENT_ERROR.
2. This call always returns a CompCode of MQCC_OK and a reason code of MQRC_NONE.

MQCLOSE – Close Object
The MQCLOSE call, which is the inverse of the MQOPEN call, relinquishes access to an object.

On WebSphere MQ for HP NonStop Server, if there is a MQGET request with the MQGMO_SET_SIGNAL option outstanding against the queue handle being closed, the request is canceled. Signal requests for the same queue but lodged against different handles (Hobj) are not affected (unless it is a dynamic queue that is being deleted, in which case, they are also canceled.)

FASTPATH applications have specific requirements concerning transactions when opening or closing dynamic queues. For more information, see “The number of concurrent active transactions for an application” on page 183.

MQDISC – Disconnect queue manager
The MQDISC call, which is the inverse of MQCONN, breaks the connection between the WebSphere MQ queue manager and the application program.

Usage note 2 in the WebSphere MQ Application Programming Reference is not applicable to WebSphere MQ for HP NonStop Server. On WebSphere MQ for HP NonStop Server, an implicit syncpoint occurs if a local unit of work (that is, initiated by the queue manager) is in progress when MQDISC is called. An implicit syncpoint does not occur if a global unit of work is in progress when MQDISC is called.

MQINQ – Inquire about object attributes
The MQINQ call returns an array of integers and a set of character strings containing the attributes of an object.

MQOPEN – Open Object
The MQOPEN call establishes access to an object. On WebSphere MQ for HP NonStop Server, the MaxHandles attribute of the queue manager is ignored.

FASTPATH applications have specific requirements concerning transactions when opening or closing dynamic queues. For more information, see “The number of concurrent active transactions for an application” on page 183.

MQSET– Set Object Attributes
The MQSET call changes the attributes of an object represented by a handle. The object must be a queue. On WebSphere MQ for HP NonStop Server, the MQIA_DIST_LISTS selector is supported.

FASTPATH applications have specific requirements concerning transactions when changing an object’s attributes using MQSET. For more information, see “The number of concurrent active transactions for an application” on page 183.
Attributes of objects

Attributes of WebSphere MQ objects

In WebSphere MQ for HP NonStop Server, the attributes of all objects are as described in the WebSphere MQ Application Programming Reference with the following exceptions and additions.

Attributes for all queues

In WebSphere MQ for HP NonStop Server, the attributes of all queues are as described in the WebSphere MQ Application Programming Reference with the following exceptions and additions.

- The AlterationDate and AlterationTime attributes are updated only when administrative changes are made to attributes of an object.
- CurrentQDepth, OpenInputCount and OpenOutputCount attributes can only be changed dynamically.
- QDepthHighCount, QDepthLowEvent, QDepthMaxEvent and QServiceIntervalEvent can be changed both dynamically and administratively, but only the administrative changes (for example performed using MQSC commands or using MQSET) cause a change in the AlterationDate and AlterationTimes attributes.

Attributes of local and model queues

In WebSphere MQ for HP NonStop Server:

- The Archive attribute is ignored.
- The HardenGetBackout attribute is ignored because the backout count is not saved to disk. There is no ability to archive messages.
- For persistent messages, the BackoutCount attribute is always hardened. For nonpersistent messages, the BackoutCount attribute is never hardened. However, if the Local Queue has its --qoptions C option attribute set, BackoutCount is checkpointed to the backup queue server. Messages that have a checkpoint taken in this way are resilient against queue server failure. To maintain compatibility with other WebSphere MQ platforms, the attribute can be queried by the MQINQ call using the MQIA_HARDEN_GET_BACKOUT selector.

Attributes of queue managers

- The MaxMsgLength attribute is 100 MB.
- The CommandLevel attribute is MQCMDL_LEVEL_530.
-.SyncPoint attribute is MQSPAVAILABLE.
- The value of the CodedCharSetId attribute is as specified when the queue manager instance was created.
- The MaxHandles attribute is ignored. You can not specify a maximum number of open handles for WebSphere MQ for HP NonStop Server. The maximum value is determined by system resource constraints.
- The MaxUncommittedMsgs attribute is ignored. You can not specify a maximum number of messages to be allowed within a single unit of work. The maximum value is determined by resource constraints.
- CCSID can be altered.
- The attributes SSLCRLNamelist and SSLCryptoHardware can be set and inquired against but are ignored.
Data conversion

See Appendix J, “User exits for Shared Resource Library (SRL) applications,” on page 471, which describes the scheme for supporting all exits on WebSphere MQ for HP NonStop Server, V5.3. The mechanism has changed from previous versions to support a more consistent and portable exit implementation.
Appendix H. Building and running applications

This chapter provides information about building and running applications in the NonStop OS and OSS environments.

Detailed information for the C, C++, COBOL, and TAL languages is provided as follows:

- “Building C language applications” on page 458
- “Building C++ applications” on page 459
- “Building COBOL applications” on page 461
- “Building non-native applications” on page 461

Floating point

The native HP NonStop Server platform (TNS/R and TNS/E) supports two formats for floating-point numbers: the TandemFloat format (also called TNSFloat) and IEEEFloat. IEEEFloat is available only for native applications. Non-native (TNS) applications must use TandemFloat.

WebSphere MQ supports native applications that use either format. Most WebSphere MQ libraries are neutral with respect to floating point type, which means that they can be used by applications of any type. However, WebSphere MQ C++ libraries are sensitive to floating point format and so they are supplied in two varieties to support both types. For C++, applications must be linked with the correct library.

Considerations for creating applications with threads

WebSphere MQ supports multithreaded OSS application programs that use T1248 Standard Posix Threads (SPT) provided by the TNS/R and TNS/E platforms. Using threads is supported in the OSS environment only; the NonStop OS environment does not support multithreaded programming. The MQI is thread-aware when used within an application linked with the correct WebSphere MQ libraries. For example, an MQGET call with the MQGMO_WAIT option blocks the current thread and allows other ready threads to run.

Transactions and the XA interface

No XA interface is provided by WebSphere MQ for global unit of work (UOW) coordination. All transactional UOW coordination is performed by TMF.

Triggered applications

Triggered WebSphere MQ applications in the NonStop OS environment receive user data through environment variables set up in the TACL process that is running. This is because there is a limit to the length of the argument list that can be passed to a NonStop OS C process. In order to access this information, ensure triggered applications contain code similar to the following (see sample amqsinqa for more details):
Locating product files

WebSphere MQ product files are located using the MQNSKOPTPATH and MQNSKVARPATH environment variables in OSS and the identically-named PARAMs in NonStop OS. For the rest of this appendix:

- $MQNSKOPTPATH refers to the OSS path defined in the MQNSKOPTPATH environment variable or PARAM.
- $MQNSKVARPATH refers to the OSS path defined in the MQNSKVARPATH environment variable or PARAM.
- <GInstBin>, <GInstInclude>, and <GInstSamp> refer to the NonStop OS subvolumes for binaries, include files, and samples that were defined during WebSphere MQ product installation.

The WebSphere MQ for HP NonStop Server product files are located in two OSS file trees and three NonStop OS subvolumes as follows:

```c
MQTMC2 *trig;   /* Trigger message structure */
MQTMC2 trigdata; /* Trigger message structure */
char *applId;
char *envData;
char *usrData;
char *qmName;

/************************************************************
/* */
/* Set the program argument into the trigger message */
/* */
/************************************************************

trig = (MQTMC2*)argv[1];  /* -> Trigger message */

/* Get the environment variables and load the rest of the trigger */
memcpy(&trigdata, trig, sizeof(trigdata));

memset(trigdata.ApplId, ' ', sizeof(trigdata.ApplId));
memset(trigdata.EnvData, ' ', sizeof(trigdata.EnvData));
memset(trigdata.UserData, ' ', sizeof(trigdata.UserData));
memset(trigdata.QMgrName, ' ', sizeof(trigdata.QMgrName));

if ((applId = getenv("TRIGAPPLID")) != 0)
{
    strncpy(trigdata.ApplId, applId, strlen(applId));
}

if ((envData = getenv("TRIGENVDATA")) != 0)
{
    strncpy(trigdata.EnvData, envData, strlen(envData));
}

if ((usrData = getenv("TRIGUSERDATA")) != 0)
{
    strncpy(trigdata.UserData, usrData, strlen(usrData));
}

if ((qmName = getenv("TRIGQMGRNAME")) != 0)
{
    strncpy(trigdata.QMgrName, qmName, strlen(qmName));
}

trig = &trigdata;
```
• The WebSphere MQ product executables and libraries in OSS are located in the path defined by $MQNSKOPTPATH.

• The WebSphere MQ product data in OSS is located in the path defined by $MQNSKVARPATH.

• The WebSphere MQ product executables, libraries, and data files in NonStop OS are located in the NonStop OS binary subvolume chosen during installation. This subvolume is pointed to by the symbolic link: $MQNSKOPTPATH/G/bin
You supply the NonStop OS form of this path <GInstBin> during WebSphere MQ installation.

• The WebSphere MQ product headers and include files in NonStop OS are located in the NonStop OS includes subvolume chosen during installation. This subvolume is pointed to by the symbolic link: $MQNSKOPTPATH/G/inc
You supply the NonStop OS form of this path <GInstInclude> during WebSphere MQ installation.

• The WebSphere MQ product samples in NonStop OS are located in the NonStop OS samples subvolume chosen during installation. This subvolume is pointed to by the symbolic link: $MQNSKOPTPATH/G/samp
You supply the NonStop OS form of this path <GInstSamp> during WebSphere MQ installation.

Supported languages and environments

WebSphere MQ supports the languages and environments described in the following table. The table also describes whether the application can use FASTPATH or STANDARD bindings.

<table>
<thead>
<tr>
<th>Language</th>
<th>Runs in NonStop OS and OSS</th>
<th>Standard bindings</th>
<th>Fastpath bindings</th>
<th>Multithreading supported</th>
</tr>
</thead>
<tbody>
<tr>
<td>C non-native</td>
<td>NonStop OS only</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>C native PIC</td>
<td>Both</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>C native non-PIC</td>
<td>Both</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>C++ native PIC</td>
<td>Both</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>C++ native non-PIC</td>
<td>Both</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>COBOL native PIC</td>
<td>Both</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>COBOL native non-PIC</td>
<td>Both</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>COBOL non-native</td>
<td>NonStop OS only</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>TAL non-native</td>
<td>NonStop OS only</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Java</td>
<td>OSS only</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Appendix H. Building and running applications 457
Running PIC applications

To run a C, C++, or COBOL PIC application that uses WebSphere MQ, ensure that the MQNSKOPTPATH and MQNSKVVARPATH environment variables or PARAMs are defined to locate your WebSphere MQ installation.

Also, ensure the WebSphere MQ DLLs are available by including the WebSphere MQ library path ($MQNSKOPTPATH/lib) in the list of paths defined by the _RLD_LIB_PATH environment variable or NonStop OS DEFINE.

For example, in OSS:
export MQNSKOPTPATH=OSS location of WMQ OPT tree
export MQNSKVVARPATH=OSS location of WMQ VAR tree
export _RLD_LIB_PATH=other user paths : $MQNSKOPTPATH/lib

In NonStop OS:
param MQNSKOPTPATH OSS location of WMQ OPT tree
param MQNSKVVARPATH OSS location of WMQ VAR tree
ADD DEFINE =_RLD_LIB_PATH,CLASS SEARCH,SUBVOL0 <GInstBin>

If you are using OSS, you can define these environment variables by sourcing the wmqprofile script located in $MQNSKVVARPATH into the shell.

If you are using NonStop OS, you can configure these PARAMs and defines by running the OBEY command against the WMQCSTM file located in <GInstSamp>.

Running non-PIC applications

Non-PIC applications include both non-native TNS applications and native TNS-R applications. These applications require the MQNSKOPTPATH and MQNSKVVARPATH environment variables or PARAMs to be set in the same way as PIC applications. Non-PIC applications ignore the _RLD_LIB_PATH environment variable or NonStop OS DEFINE.

If you are using non-PIC applications, you can set these variables using the $MQNSKVVARPATH/wmqprofile script (for OSS) and the WMQCSTM obey file (for NonStop OS) in the same way as described above for PIC applications.

Building C language applications

This section provides details of how to build and link applications in C.

Native PIC application programs and DLLs

PIC application programs and DLLs are supported in the native NonStop OS and OSS environments only.

Include files needed by WebSphere MQ applications are in $MQNSKOPTPATH/inc or <GInstInclude>

Link unthreaded applications using the ld command with the following libraries:

In OSS:
-LMQNSKOPTPATH/1ib -lmqm

In NonStop OS:
-L<InstBin> -lmqm

Link multithreaded applications with the following libraries in OSS:
-1zsptrsl -L$MQNSKOPTPATH/lib -lmqm_r

The following table summarizes the include file paths, compile options, and libraries used to link and compile PIC application programs and DLLs that use WebSphere MQ:

<table>
<thead>
<tr>
<th>Application or DLL</th>
<th>OSS development environment</th>
<th>NonStop OS development environment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Include files</td>
<td>-I$MQNSKOPTPATH/inc</td>
<td>SSVn &quot;&lt;GInstInclude&gt;&quot;</td>
</tr>
<tr>
<td>PIC compile</td>
<td>-Wcall_shared or -Wshared</td>
<td>CALL_SHARED or SHARED</td>
</tr>
<tr>
<td>Libraries (unthreaded)</td>
<td>mqm</td>
<td>mqm</td>
</tr>
<tr>
<td>Libraries (threaded)</td>
<td>mqm_r</td>
<td>Not applicable</td>
</tr>
</tbody>
</table>

Native non-PIC application programs (TNS-R only)

The WebSphere MQ non-PIC libraries are supplied in both user library (SRL) form and in static (relinkable SRL) form. All non-PIC libraries are located in the NonStop OS installation subvolume <InstBin>, which can be found by following the symbolic link located in $MQNSKOPTPATH/G/lib.

<table>
<thead>
<tr>
<th>File</th>
<th>OSS development environment</th>
<th>NonStop OS development environment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Include files</td>
<td>-I$MQNSKOPTPATH/inc</td>
<td>SSVn &quot;&lt;GInstInclude&gt;&quot;</td>
</tr>
<tr>
<td>Non-PIC compile</td>
<td>-Wnon_shared or -Wsrl</td>
<td>NON_SHARED or SRL</td>
</tr>
</tbody>
</table>

The following table shows which non-PIC libraries are available and how they can be used with the native non-PIC linker (NLD) to produce an executable WebSphere MQ application program:

<table>
<thead>
<tr>
<th>Library</th>
<th>Unthreaded applications</th>
<th>Multithreaded applications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static library</td>
<td>&lt;InstBin&gt;.mqmlib</td>
<td>&lt;InstBin&gt;.mqmrlib</td>
</tr>
<tr>
<td>Static library</td>
<td>&lt;InstBin&gt;.mqlib</td>
<td>&lt;InstBin&gt;.mqmfrlib</td>
</tr>
<tr>
<td>User library</td>
<td>&lt;InstBin&gt;.mqmsrl</td>
<td>&lt;InstBin&gt;.mqmfrsrl</td>
</tr>
<tr>
<td>User library</td>
<td>&lt;InstBin&gt;.mqmsrl</td>
<td>-libname &lt;InstBin&gt;.mqmsrl</td>
</tr>
</tbody>
</table>

Note:
1. These libraries support STANDARD-bound connections only.

Building C++ applications

This section provides details of how to build and link applications in C++. 

Appendix H. Building and running applications 459
Native PIC C++ applications and DLLs

PIC application programs and DLLs are supported in the native NonStop OS and OSS environments only.

Include files needed by WebSphere MQ applications are in:

- $MQNSKOPTPATH/inc for OSS
- <GInstInclude> for NonStop OS

Link applications with a library of the form imqi<nnn> where <nnn> is chosen from the table below.

- L$MQNSKOPTPATH/lib –limqi<nnn> for OSS
- <GInstBin> –limqi<nnn> for NonStop OS

Table 47. Native PIC: library names for applications and DLLs in OSS and NonStop OS

<table>
<thead>
<tr>
<th>Application, DLL, or file</th>
<th>OSS environment</th>
<th>NonStop OS environment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Include files</td>
<td>-$MQNSKOPTPATH/inc</td>
<td>SSVn “&lt;GInstInclude&gt;”</td>
</tr>
<tr>
<td>PIC Compile</td>
<td>-Wcall_shared or –Wshared</td>
<td>CALL_SHARED or SHARED</td>
</tr>
<tr>
<td>C++ Version2 IEEE_Float unthreaded</td>
<td>-limqi2</td>
<td>-limqi2</td>
</tr>
<tr>
<td>C++ Version3 IEEE_Float unthreaded</td>
<td>-limqi3</td>
<td>-limqi3</td>
</tr>
<tr>
<td>C++ Version2 IEEE_Float</td>
<td>-limqi2_r</td>
<td>-limqi2r</td>
</tr>
<tr>
<td>C++ Version3 IEEE_Float multithreaded</td>
<td>-limqi3_r</td>
<td>-limqi3r</td>
</tr>
<tr>
<td>C++ Version2 Tandem_Float unthreaded</td>
<td>-limqi2t</td>
<td>-limqi2t</td>
</tr>
<tr>
<td>C++ Version3 Tandem_Float unthreaded</td>
<td>-limqi3t</td>
<td>-limqi3t</td>
</tr>
<tr>
<td>C++ Version2 Tandem_Float multithreaded</td>
<td>-limqi2tr</td>
<td>-limqi2tr</td>
</tr>
<tr>
<td>C++ Version3 Tandem_Float multithreaded</td>
<td>-limqi3tr</td>
<td>-limqi3tr</td>
</tr>
</tbody>
</table>

Native non-PIC C++ applications and SRLs (TNS-R Only)

Non-PIC application programs are supported in the native NonStop OS and OSS environment only. There is no non-native (TNS) support for C++ applications.

Include files needed by WebSphere MQ C++ applications are in:

- $MQNSKOPTPATH/inc for OSS
- <GInstInclude> for NonStop OS

Non-PIC WebSphere MQ C++ applications must link with a non-PIC library as defined in “Building C language applications” on page 458 and must additionally statically link with the WebSphere MQ C++ library of the form im<nnn>lib (where <nnn> is chosen from the table below)

For example, to link a C++ Version2 application that uses IEEE_Float and is unthreaded, use either of the following commands:

OSS non-PIC link:
Building C++ applications

nld myapp.o
....
$MQNSKOPTPATH/G/lib/im2lib
-libname <GInstBin>.mqmsrl
-o myapp

NonStop OS non-PIC link:

nld myappo
....
<GInstBin>.im2lib
-libname <GInstBin>.mqmsrl
-o myapp

Table 48. Non-native PIC: library names for applications and DLLs in OSS and NonStop OS

<table>
<thead>
<tr>
<th>Application or DLL</th>
<th>OSS build environment</th>
<th>NonStop OS build environment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Include files</td>
<td>-I$MQNSKOPTPATH/inc</td>
<td>SSVn “&lt;GInstInclude&gt;”</td>
</tr>
<tr>
<td>Non-PIC compile</td>
<td>-Wnon_shared or -Wsrl</td>
<td>NON_SHARED or SRL</td>
</tr>
<tr>
<td>C++ Version2 IEEE_Float unthreaded</td>
<td>$MQNSKOPTPATH/G/lib/im2lib</td>
<td>&lt;GInstBin&gt;.im2lib</td>
</tr>
<tr>
<td>C++ Version3 IEEE_Float unthreaded</td>
<td>$MQNSKOPTPATH/G/lib/im3lib</td>
<td>&lt;GInstBin&gt;.im3lib</td>
</tr>
<tr>
<td>C++ Version2 IEEE_Float multithreaded</td>
<td>$MQNSKOPTPATH/G/lib/im2rlib</td>
<td>&lt;GInstBin&gt;.im2rlib</td>
</tr>
<tr>
<td>C++ Version3 IEEE_Float multithreaded</td>
<td>$MQNSKOPTPATH/G/lib/im3rlib</td>
<td>&lt;GInstBin&gt;.im3rlib</td>
</tr>
<tr>
<td>C++ Version2 Tandem_Float unthreaded</td>
<td>$MQNSKOPTPATH/G/lib/im2tlib</td>
<td>&lt;GInstBin&gt;.im2tlib</td>
</tr>
<tr>
<td>C++ Version3 Tandem_Float unthreaded</td>
<td>$MQNSKOPTPATH/G/lib/im3tlib</td>
<td>&lt;GInstBin&gt;.im3tlib</td>
</tr>
<tr>
<td>C++ Version2 Tandem_Float multithreaded</td>
<td>$MQNSKOPTPATH/G/lib/im2trlib</td>
<td>&lt;GInstBin&gt;.im2trlib</td>
</tr>
<tr>
<td>C++ Version3 Tandem_Float multithreaded</td>
<td>$MQNSKOPTPATH/G/lib/im3trlib</td>
<td>&lt;GInstBin&gt;.im3trlib</td>
</tr>
</tbody>
</table>

Building COBOL applications

Native COBOL applications can be built in either PIC or non-PIC form. Link native COBOL PIC applications with the same WebSphere MQ DLLs that are required for PIC C language applications.

Native non-PIC COBOL applications must use either the static or user library SRLs defined for non-PIC C language applications.

See “Building C language applications” on page 458 for details.

Building non-native applications

C Language, COBOL and TAL are supported in the non-native TNS environment. A single static library (called MQMTNS) is provided for WebSphere MQ non-native applications. MQMTNS is in the installation NonStop OS subvolume <GInstBin> that you supplied during WebSphere MQ installation.

Only STANDARD-bound connections are supported by WebSphere MQ in the non-native environment.
Appendix I. WebSphere MQ for HP NonStop Server sample programs

Two sets of sample programs are supplied with WebSphere MQ for HP NonStop Server: a set of sample programs for running on OSS, and another set of sample programs for running on NonStop OS. The OSS sample programs are in the directory opt_installation_path/samp, and the NonStop OS sample programs are in the subvolume specified for the NonStop OS sample programs at installation.

OSS sample programs

Table 49 lists the C and COBOL sample programs for running on OSS.

Table 49. C and COBOL sample programs for running on OSS

<table>
<thead>
<tr>
<th>Description</th>
<th>C source</th>
<th>COBOL source</th>
<th>C executable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Putting messages using the MQPUT call</td>
<td>amqspput0</td>
<td>amq0put0</td>
<td>amqspput</td>
</tr>
<tr>
<td>Putting a single message using the MQPUT1 call</td>
<td>amqsinqa</td>
<td>No sample</td>
<td>amqsinq</td>
</tr>
<tr>
<td></td>
<td>amqsecha</td>
<td>amqsech</td>
<td></td>
</tr>
<tr>
<td>Putting messages to a distribution list</td>
<td>amqsp0l0</td>
<td>No sample</td>
<td>amqsp0l</td>
</tr>
<tr>
<td>Replying to a request message</td>
<td>amqsinqa</td>
<td>No sample</td>
<td>amqsinq</td>
</tr>
<tr>
<td>Getting messages (no wait)</td>
<td>amqsgbr0</td>
<td>amq0gbr0</td>
<td>amqsgbr</td>
</tr>
<tr>
<td>Getting messages (wait with a time limit)</td>
<td>amqget0</td>
<td>amq0get0</td>
<td>amqget</td>
</tr>
<tr>
<td>Getting messages (with data conversion)</td>
<td>amqsecha</td>
<td>No sample</td>
<td>amqsech</td>
</tr>
<tr>
<td>Putting reference messages to a queue</td>
<td>amqsgprma</td>
<td>No sample</td>
<td>amqsgprm</td>
</tr>
<tr>
<td>Getting reference messages from a queue</td>
<td>amqsgrma</td>
<td>No sample</td>
<td>amqsgrm</td>
</tr>
<tr>
<td>Reference message channel exit</td>
<td>amqsrma</td>
<td>No sample</td>
<td>amqsrm</td>
</tr>
<tr>
<td></td>
<td>amqsxrm</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Browsing the first 20 characters of a message</td>
<td>amqsgbr0</td>
<td>amq0gbr0</td>
<td>amqsgbr</td>
</tr>
<tr>
<td>Browsing complete messages</td>
<td>amqsb0cg</td>
<td>No sample</td>
<td>amqsb0cg</td>
</tr>
<tr>
<td>Using an exclusive input queue</td>
<td>amqstrg0</td>
<td>amq0req0</td>
<td>amqstrg</td>
</tr>
<tr>
<td>Using the MQINQ call</td>
<td>amqsinqa</td>
<td>No sample</td>
<td>amqsinq</td>
</tr>
<tr>
<td>Using the MQSET call</td>
<td>amqsseta</td>
<td>No sample</td>
<td>amqsset</td>
</tr>
<tr>
<td>Using a reply-to queue</td>
<td>amqsreq0</td>
<td>amq0req0</td>
<td>amqsreq</td>
</tr>
<tr>
<td>Requesting message exceptions</td>
<td>amqsreq0</td>
<td>amq0req0</td>
<td>amqsreq</td>
</tr>
<tr>
<td>Accepting a truncated message</td>
<td>amqsgbr0</td>
<td>amq0gbr0</td>
<td>amqsgbr</td>
</tr>
<tr>
<td>Using a resolved queue name</td>
<td>amqsgbr0</td>
<td>amq0gbr0</td>
<td>amqsgbr</td>
</tr>
<tr>
<td>Triggering a process</td>
<td>amqstrg0</td>
<td>No sample</td>
<td>amqstrg</td>
</tr>
<tr>
<td>Using data conversion</td>
<td>amqsvfc0.c</td>
<td>No sample</td>
<td></td>
</tr>
<tr>
<td>Dead letter queue handler</td>
<td>[1]</td>
<td>No sample</td>
<td>amqsd1q</td>
</tr>
<tr>
<td>Using API exits</td>
<td>amqsa0xe.c</td>
<td>No sample</td>
<td>amqsa0xe</td>
</tr>
<tr>
<td>Cluster workload balancing exit</td>
<td>amqswlm0.c</td>
<td>No sample</td>
<td>amqswlm</td>
</tr>
</tbody>
</table>

© Copyright IBM Corp. 2006
## OSS sample programs

Table 49. C and COBOL sample programs for running on OSS (continued)

<table>
<thead>
<tr>
<th>Description</th>
<th>C source</th>
<th>C executable</th>
<th>COBOL source</th>
<th>COBOL executable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Notes:</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1. The source for the dead letter queue handler is made up of several files</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

## Using the OSS sample programs

For more information about the OSS sample programs, including how to prepare and run them, see the [WebSphere MQ Application Programming Guide](#). For platform specific information, use the information that is provided for UNIX systems.

## NonStop OS sample programs

Table 50 lists the C and COBOL sample programs for running on NonStop OS.

Table 50. C and COBOL sample programs for running on NonStop OS

<table>
<thead>
<tr>
<th>Description</th>
<th>C source</th>
<th>C executable</th>
<th>COBOL85 source</th>
<th>COBOL85 executable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create a local queue using the MQAI</td>
<td>mqsaiqc</td>
<td>mqsaiq</td>
<td>No sample</td>
<td>No sample</td>
</tr>
<tr>
<td>Basic event monitor using the MQAI</td>
<td>mqsaiem</td>
<td>mqsaiem</td>
<td>No sample</td>
<td>No sample</td>
</tr>
<tr>
<td>Inquire current depth of local queues using the MQAI</td>
<td>mqsailqc</td>
<td>mqsailq</td>
<td>No sample</td>
<td>No sample</td>
</tr>
<tr>
<td>Sample API exit which traces MQI calls</td>
<td>mqsaxe0c</td>
<td>mqsaxe0 (DLL)</td>
<td>No sample</td>
<td>No sample</td>
</tr>
<tr>
<td>Read and output message descriptor and context for each message on a queue</td>
<td>mqsbcg0c</td>
<td>mqsbcg0</td>
<td>No sample</td>
<td>No sample</td>
</tr>
<tr>
<td>Bandwidth and connectivity tester</td>
<td>mqsblstc</td>
<td>mqsblst</td>
<td>No sample</td>
<td>No sample</td>
</tr>
<tr>
<td>Echo a message from a message queue to the reply-to queue</td>
<td>mqsechac</td>
<td>mqsea</td>
<td>No sample</td>
<td>No sample</td>
</tr>
<tr>
<td>Write messages from a queue to stdout, leave messages on the queue (Browse)</td>
<td>mqsgrbr0c</td>
<td>mqsgrbr0</td>
<td>mqsgrbr0l</td>
<td>mqsgrbr0</td>
</tr>
<tr>
<td>Remove messages from the named queue and write to stdout</td>
<td>mqsget0c</td>
<td>mqsget0</td>
<td>mqsget0l</td>
<td>mqsget0</td>
</tr>
<tr>
<td>Get reference messages</td>
<td>mqsgrmac</td>
<td>mqsgrma</td>
<td>No sample</td>
<td>No sample</td>
</tr>
<tr>
<td>Read the triggered queue, respond with queue information</td>
<td>mqsinqac</td>
<td>mqsinqa</td>
<td>No sample</td>
<td>No sample</td>
</tr>
<tr>
<td>Use a shared input queue</td>
<td>No sample</td>
<td>No sample</td>
<td>mqsinq0l</td>
<td>mqsinq0</td>
</tr>
<tr>
<td>Sample skeleton for MQLOADEXIT</td>
<td>mqsixp0c</td>
<td>No sample</td>
<td>No sample</td>
<td>No sample</td>
</tr>
<tr>
<td>Create and put reference messages</td>
<td>mqsprmac</td>
<td>mqsprma</td>
<td>No sample</td>
<td>No sample</td>
</tr>
<tr>
<td>Put messages to a distribution list</td>
<td>mqsptl0c</td>
<td>mqsptl0</td>
<td>mqsptl0l</td>
<td>mqsptl0</td>
</tr>
<tr>
<td>Copy stdin to a message and put the message on a specified queue</td>
<td>mqsput0c</td>
<td>mqsput0</td>
<td>mqsput0l</td>
<td>mqsput0</td>
</tr>
<tr>
<td>Put a request message on a specified queue and display the replies</td>
<td>mqsreq0c</td>
<td>mqsreq0</td>
<td>mqsreq0l</td>
<td>mqsreq0</td>
</tr>
<tr>
<td>(Trigger function) inhibit puts on a named queue and respond with a statement of the result</td>
<td>mqssetac</td>
<td>mqsseta</td>
<td>mqsset0l</td>
<td>mqsset0</td>
</tr>
</tbody>
</table>
Table 50. C and COBOL sample programs for running on NonStop OS (continued)

<table>
<thead>
<tr>
<th>Description</th>
<th>C source</th>
<th>C executable</th>
<th>COBOL85 source</th>
<th>COBOL85 executable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trigger monitor</td>
<td>mqstrg0c</td>
<td>mqstrg0</td>
<td>No sample</td>
<td>No sample</td>
</tr>
<tr>
<td>Sample skeleton for channel exit</td>
<td>mqsvehnc</td>
<td>No sample</td>
<td>No sample</td>
<td>No sample</td>
</tr>
<tr>
<td>Sample skeleton for data conversion exit</td>
<td>mqsvef0c</td>
<td>No sample</td>
<td>No sample</td>
<td>No sample</td>
</tr>
<tr>
<td>Channel message exit that processes reference messages</td>
<td>mqsxrmac</td>
<td>mqsxrma (DLL)</td>
<td>No sample</td>
<td>No sample</td>
</tr>
<tr>
<td>Sample skeleton for cluster workload exit</td>
<td>mqsxml0c</td>
<td>No sample</td>
<td>No sample</td>
<td>No sample</td>
</tr>
</tbody>
</table>

Table 51 lists the C++ sample programs for running on NonStop OS.

Table 51. C++ sample programs for running on NonStop OS

<table>
<thead>
<tr>
<th>Description</th>
<th>C++ source</th>
<th>C++ executable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Put messages to a named queue</td>
<td>imqsputp</td>
<td>imqsput</td>
</tr>
<tr>
<td>Get messages from a named queue</td>
<td>imqsgetp</td>
<td>imqsget</td>
</tr>
<tr>
<td>Put and get messages</td>
<td>imqwrldp</td>
<td>imqwrld</td>
</tr>
<tr>
<td>Put messages to a distribution list</td>
<td>imqwpuput</td>
<td>imqwpuld</td>
</tr>
</tbody>
</table>

Table 52 lists the TAL sample programs for running on NonStop OS.

Table 52. TAL sample programs for running on NonStop OS

<table>
<thead>
<tr>
<th>Description</th>
<th>TAL source</th>
<th>TAL executable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read n messages from a queue</td>
<td>zmqreadt</td>
<td>zmqread</td>
</tr>
<tr>
<td>Write n messages of length m bytes to a queue</td>
<td>zmqwritt</td>
<td>zmqwrit</td>
</tr>
</tbody>
</table>

**TACL macro files for building C sample programs**

**Non-Native (using non-native static library MQMTNS)**

The samples subvolume contains the following TACL macro files to be used for building non-native sample C applications:

**CSAMP**  
Usage: CSAMP source-code-file-name

This is a basic macro for compiling a C source file using the WebSphere MQ include files. For example, to compile the sample MQSBGC0C, use CSAMP MQSBGC0C. If the compilation is successful, the macro produces an object file with the last character of the file name replaced by the letter O; for example, MQSBGC0O.

**BSAMP**  
Usage: BSAMP exe-file-name

This is a basic macro used to bind an object file with the user library MQMTNS in the executables subvolume. For example, to bind the compiled sample MQSBGC0C, use BSAMP MQSBGC0. The macro produces an executable file called exe-file-name; for example, MQSBGC0.
TACL macro files for building C sample programs

This TACL macro compiles each of the sample source code files using the CSAMP macro.

**BINDALL**
Usage: **BINDALL**
This TACL macro binds each of the sample object files into executables using the BSAMP macro.

**BUILDC**
Usage: **BUILDC**
This TACL macro compiles and binds all of the C sample files using the macros COMPALL and BINDALL.

**Native (using native static library MQMLIB)**
For a native install, the following TACL macro files are to be used for building sample MQI applications:

**NMCALL**
Usage: **NMCALL**
Macro to compile all samples native using NMCSAMP.

**NMCSAMP**
Usage: **NMCSAMP source-code-file-name**
This is a basic macro for compiling a C source file using the WebSphere MQ include files. For example, to compile the sample MQSBCG0C, use NMCSAMP MQSBCG0C. If the compilation is successful, the macro produces an object file with the last character of the file name replaced by the letter O; for example, MQSBCG0O.

**NMLDSAMP**
Usage: **NMLDSAMP exe-file-name**
This basic macro links an object file with the static Native MQI library MQMLIB in the executables subvolume.

**NMLDALL**
Usage: **NMLDALL**
This TACL macro binds each of the sample object files into executables using the NMLDSAMP macro.

**NMBUILDC**
Usage: **NMBUILDC**
This TACL macro compiles and binds all of the Native C sample files using the macros NMCALL and NMLDALL.

**Native (using SRL MQMSRL)**

**NMLDSSMP**
Usage: **NMLDSSMP exe-file-name**
This basic macro links an object file with the Native MQ SRL MQMSRL in the executables subvolume.

**NMLDSALL**
Usage: **NMLDSALL**
This TACL macro binds each of the sample object files into executables using the NMLDSSMP macro.

**NMBULDSC**
Usage: **NMBULDSC**
This TACL macro compiles and binds all of the Native C sample files using the macros NMCALL and NMLDSALL.

TACL macro files for building C++ sample programs

**Native (using native static library MQMLIB)**

**NMCPPALL**
Usage: **NMCPPALL**
TACL macro files for building C++ sample programs

Macro to compile all the sample programs using NMCCPP.

NMCCPP Usage: NMCCPP source-code-file-name C++-version
This is a basic macro for compiling a C++ source file using the WebSphere MQ include files. For example, to compile the sample IMQSGETP using C++ Version 2, use NMCCPP IMQSGETP version2. If the compilation is successful, the macro produces an object file with the last character of the file name replaced by the letter O; for example, IMQSGETO.

NMLDCPP Usage: NMLDCPP exe-file-name
This basic macro links an object file with the Static Native MQI library MQMLIB in the executables subvolume.

NMLDCPPA Usage: NMLDCPPA
This TACL macro binds each of the sample object files into executables using the NMLDSAMP macro.

NMBLDCPP Usage: NMBU1DC
This TACL macro compiles and binds all of the Native C++ sample files using the macros NMCPPALL and NMLDCPPA.

Native (using SRL MQMSRL)

NMLDCPPS Usage: NMLDCPPS exe-file-name
This basic macro links an object file with the Native MQ SRL MQMSRL in the executables subvolume.

NMLDCPSA Usage: NMLDCPSA
This TACL macro binds each of the sample object files into executables using the NMLDCPPS macro.

NMLDSACP Usage: NMLDSACP
This TACL macro compiles and binds all of the Native C sample files using the macros NMCPPALL and NMLDCPSA.

TACL macro files for building COBOL sample programs

Non-Native (using non-native static library MQMTNS)
The samples subvolume contains the following files to be used for building sample COBOL applications.

COBSAMP Usage: COBSAMP source-code-file-name
This is a basic macro for compiling a COBOL source file using the WebSphere MQ definition files. For example, to compile the program MQSGBR0L, use COBSAMP MQSGBR0L. If the compilation is successful, the macro produces an object file with the last character of the file name replaced by the letter O; for example, MQSGBR0.

BCOBSAMP Usage: BCOBSAMP exe-file-name
This is a basic macro used to bind an object with the user libraries in the executables subvolume. For example, to bind the compiled sample MQSGBR0L, use BCOBSAMP MQSGBR0L. The macro produces an executable called exe-file-name MQSGBR0.

CCBSMPLS Usage: CCBSMPLS
This TACL macro compiles each of the COBOL sample source code files.

**BCBSMPLS**
Usage: BIND /IN BCBSMPLS/

This bind input file binds each of the COBOL sample object files into executables.

**BUILDCOB**
Usage: BUILDCOB

This TACL macro compiles and binds all of the COBOL sample files using the macros CCBSMPLS and BCBSMPLS.

**Native (using native static library MQMLIB)**

**NMCOBSMP**
Usage: NMCOBSMP source-code-file-name

This is a macro for compiling a COBOL source file using the native mode COBOL compiler, NMCOBOL. The macro uses the WebSphere MQ native library, MQMLIB, in the executables subvolume. For example, to compile the program MQSGBR0L, use NMCOBSMP MQSGBR0L. If the compilation is successful, the macro produces an object file with the last character of the file name replaced by the letter O; for example, MQSGBR0O.

**NMLDCOB**
Usage: NMLDCOB exe-file-name

This macro binds object with the WebSphere MQ Native library MQMLIB in the executables subvolume. For example, to bind the compiled sample MQSGBR0L, use NMLDCOB MQSGBR0L. The macro produces an executable called exe-file-name MQSGBR0.

**NMCOBALL**
Usage: NMCOBALL

This TACL macro compiles each of the COBOL sample source code files using NMCOBSMP.

**NMLDACOB**
Usage: NMLDACOB

This bind input file binds each of the NMLDCOB sample object files into executables.

**NMBLDCOB**
Usage: NMBLDCOB

This TACL macro compiles and binds all of the COBOL sample files using the macros NMCOBALL and NMLDACOB.

**Native (using SRL MQSRL)**

**NMCBSSMP**
Usage: NMCBSSMP source-code-file-name

This is a macro for compiling a COBOL source file using the native mode COBOL compiler, NMCOBOL. The macro uses the WebSphere MQ SRL, MQSRL, in the executables subvolume. For example, to compile the program MQSGBR0L, use NMCBSSMP MQSGBR0L. If the compilation is successful, the macro produces an object file with the last character of the file name replaced by the letter O; for example, MQSGBR0O.

**NMLDSCOB**
Usage: NMLDSCOB exe-file-name

This macro binds object with the WebSphere MQ SRL MQSRL in the executables subvolume. For example, to bind the compiled sample MQSGBR0, use NMLDSCOB MQSGBR0. The macro produces an executable called exe-file-name MQSGBR0.
TACL macro files for building COBOL sample programs

NMCBSALL Usage: NMCBSALL
This TACL macro compiles each of the COBOL sample source code files using NMCBSSMP.

NMLDSCOB Usage: NMLDSCOB
This bind input file binds each of the NMLDSCOB sample object files into executables.

NMBLDSCB Usage: NMBLDSCB
This TACL macro compiles and binds all of the COBOL sample files using the macros NMCBSALL and NMLDSCOB.

TACL macro files for building TAL sample programs

Non-Native (using non-native static library MQMLIB)
The samples subvolume contains the following files to be used for building sample TAL programs.

TALSAMP Usage: TALSAMP source-code-file-name This is a basic macro for compiling a TAL source file using the WebSphere MQ definition files. For example, to compile the program ZMQWRITT, use TALSAMP ZMQWRITT. If the compilation is successful, the macro produces an object file with the last character of the file name replaced by the letter O; for example, ZMQWRITO.

BTALSAMP Usage: BTALSAMP exe-file-name
This is a basic macro used to bind an object with the user libraries in the executables subvolume. For example, to bind the compiled sample ZMQWRITO, use BTALSAMP ZMQWRIT.

CTLSMPLS Usage: CTLSMPLS
This TACL macro compiles each of the TAL sample source code files.

BTLSMPLS Usage: BIND /IN BTLSMPLS/
This bind input file binds each of the TAL sample object files into executables.

BUILD TAL Usage: BUILD TAL
This TACL macro compiles and binds all of the TAL sample files using the macros CTLSMPLS and BTLSMPLS.
TACL macro files for building TAL sample programs
Appendix J. User exits for Shared Resource Library (SRL) applications

WebSphere MQ V5.3 defines many user exits that run inside internal WebSphere MQ processes, for example channels. Other user exits (for example, data conversion and API exits) run inside the application process and you must build and install these exits in both the PIC and non-PIC environments. You must also build some exits in both threaded and unthreaded form, although for other exits you need build them in one of these forms only.

The following table lists each exit supported by WebSphere MQ and the type of build and runtime environment the exit must accommodate:

<table>
<thead>
<tr>
<th>Exit</th>
<th>Non-native TNS</th>
<th>Native non-PIC (SRL)</th>
<th>Native PIC</th>
<th>Unthreaded or multithreaded forms required</th>
</tr>
</thead>
<tbody>
<tr>
<td>All channel exits</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Both</td>
</tr>
<tr>
<td>Cluster workload management exit</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Both</td>
</tr>
<tr>
<td>Data-conversion exits</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Both</td>
</tr>
<tr>
<td>API exits</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Both</td>
</tr>
<tr>
<td>MQ_LOAD exit</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Both</td>
</tr>
</tbody>
</table>

Exits that are built for a native PIC environment are physical DLLs that are installed and loaded as defined in the WebSphere MQ family documentation. Exits that are built for either non-native TNS or native non-PIC environments are functions that are installed into the appropriate WebSphere MQ product libraries. The procedures for building exits in these environments are described later in this appendix.

User exits in the PIC environment

A user exit that runs in a PIC environment must be built as a PIC DLL using a compile and link command. This command differs for unthreaded and multithreaded PIC exits. For example:

To compile and link an unthreaded PIC exit, use a command like:

```
c89 -Wshared -WIEEE_float -Wsystype=oss -I$MQNSKOPTPATH/inc
-Wld="-e MQStart" -L$MQNSKOPTPATH/lib -lmqm
exit.c -o exitname
```

To compile and link a multithreaded PIC exit, use a command like:

```
c89 -Wshared -WIEEE_float -Wsystype=oss -I$MQNSKOPTPATH/inc
-Wld="-e MQStart" -L$MQNSKOPTPATH/lib -lmqm_r -lzspsr
exit.c -o exitname_r
```

Note that the multithreaded exit must have a name ending in _r.
User exits in the non-PIC environment

Some types of WebSphere MQ user exits run in the application program environment. In non-PIC application environments (non-PIC native SRL and non-native TNS), install these exits into the relevant WebSphere MQ libraries. The exits that you install into the non-PIC WebSphere MQ libraries are:

- The MQ_LOAD_ENTRY_POINT_EXIT
- Data-conversion exits
- API exits

**MQ_LOAD_ENTRY_POINT_EXIT - Loading user exits**

Link data conversion and API exits into the WebSphere MQ non-PIC libraries. Before enabling any other WebSphere MQ user exit in the non-PIC environment, you must install an MQ_LOAD_ENTRY_POINT_EXIT program to map your exit names to entry-point addresses. Your MQ_LOAD_ENTRY_POINT_EXIT program must be linked into the WebSphere MQ SRLs or static libraries and this program is called by WebSphere MQ whenever one of the other user exits is enabled. The MQ_LOAD_ENTRY_POINT_EXIT program’s name is fixed, that is, its external function name must remain MQ_LOAD_ENTRY_POINT. The MQ_LOAD_ENTRY_POINT_EXIT is free to map an exit name to any entry point address or to map many exit names to the same entry-point address.

WebSphere MQ supplies a default MQ_LOAD_ENTRY_POINT_EXIT function that always returns MQXCC_FAILED (exit name not found) when called. Replace this default exit with your own before enabling any data-conversion exits or API exits.

**Parameters**

Parameters:

- `Exitparms (PMQLXP)` – input/output
  - LoadExit Parameter Block

**Usage notes:** The function performed by the MQ_LOAD_ENTRY_POINT_EXIT program is defined by the provider of the exit.

Figure 31 on page 474 contains a sample MQ_LOAD_ENTRY_POINT_EXIT program that maps three exit names to entry point addresses.

**MQLXP - MQ_LOAD_ENTRY_POINT_EXIT parameter structure**

The MQLXP structure describes the information that is passed to the load exit. This structure is supported for NonStop OS only.

**Fields**

- **StrucId (MQCHAR4)**
  - Identifier for load exit parameter structure.

  The value is MQLXP_STRUC_ID.

For the C programming language, the constant MQLXP_STRUC_ID_ARRAY is also defined. This has the same value as MQLXP_STRUC_ID, but is an array of characters instead of a string. This is an input field to the exit.
Version (MQLONG)
Version-1 load exit parameter structure.

The value is MQLXP_VERSION_1. The MQLXP_CURRENT_VERSION (Current version of load exit parameter structure) constant specifies the version number of the current version. This is an input field to the exit.

QMgrName (MQCHAR48)
Name of the queue manager that has invoked the load exit.

This is an input field to the exit. The name is padded with blanks to the length of the field.

EntryPointName (MQCHAR32)
Name of the entry point that the load exit needs to resolve to a callable address.

This is an input field to the exit. The name is padded with blanks to the length of the field.

EntryAddress (PMQFUNC)
Returned callable address of the requested entry point.

This is an output field from the exit.

ExitResponse (MQLONG)
Response from exit.

This is set by the exit to indicate whether resolving of the Entry Point Name to a callable address was successful. It must be one of the following:

MQXCC_OK
Success.
This indicates that processing of the exit successfully resolved the EntryPointName supplied in the ExitParms to a callable address. The callable address is returned in the EntryAddress field in the MQLXP structure.

MQXCC_FAILED
Failed.
This indicates that the exit was unable to resolve the EntryPointName supplied in the ExitParms to a callable address.

Any other value that is returned in the ExitResponse field has the same meaning as MQXCC_FAILED.

MQ_LOAD_ENTRY_POINT_EXIT example
Figure 31 on page 474 is an example of a working MQ_LOAD_ENTRY_POINT_EXIT program that maps three exit names (two channels exits and one data-conversion exit) to entry point addresses. The source code for the MQ_LOAD_ENTRY_POINT_EXIT sample program is provided in the samples subvolume (AMQSLXP0).
AMQSLXP0 is a sample C skeleton of a Load Exit function.
The function resolves EntryNames to callable addresses.
Once complete the code should be compiled into a loadable
object, the name of the object should be the name of the
format to be converted. Instructions on how to do this are
contained in the README file in this directory.

AMQSLXP0 takes the parameters defined for a Load Exit
routine in the CMQXC.H header file.

Figure 31. Sample MQLOADEXIT
**Installing a non-PIC exit in the WebSphere MQ native non-PIC libraries**

To compile and link an unthreaded non-PIC exit, use a command like:

```
c89 -Wnon_shared -WIEE_float -Wsystype=oss -I$MQNSKOPTPATH/inc exit.c -o exitname.o
```

To compile and link a multithreaded non-PIC exit, use a command like:

```
c89 -Wnon_shared -WIEE_float -Wsystype=oss -I$MQNSKOPTPATH/inc exit.c -o exitname_r.o
```

Note that the multithreaded exit object must have a name ending in _r.o.

The procedures for installing these exits into the WebSphere MQ product libraries are described below.

To install one or more exit objects into the WebSphere MQ runtime and static SRLs, run the mqlinksrls script in the OSS environment. The mqlinksrls script accepts a single argument that names the OSS directory containing the objects to be linked. mqlinksrls accepts any or all of the following types of files:

- Unthreaded object files ending with .o
- Unthreaded relinkable SRL files ending with .lib
- Unthreaded archive files ending with .a
- Multithreaded object files ending with _r.o
- Multithreaded relinkable SRL files ending with _r.lib
- Multithreaded archive files ending with _r.a

You must compile reentrant objects correctly for the T1248 multithreaded environment. The files that are not reentrant are linked into the non-threaded libraries and the reentrant files are linked into the multithreaded libraries.

1. In OSS, compile the exit functions into an empty directory, for example myexitdir:
   ```
cd myexitdir
c89 --o my_myexitdir.o -Wsr1 -Wsystype=oss -c my_myexitdir.o
   ```

2. Compile the MQLOADENTRYPOINT exit function. For example:
   ```
cd myexitdir
c89 --o my_mqload_exit.o -Wsr1 -Wsystype=oss -c my_mqload_exit.o
   ```

3. Stop all non-PIC applications that use the WebSphere MQ product SRLs.
4. Link the exits into the non-PIC product libraries using the mqlinksrls script and specify the directory containing the user exit objects to be linked. Note that all objects (files ending with .o) are linked with the WebSphere MQ libraries. Carry out this step as the WebSphere MQ administrator.

With the above MQ_LOAD_ENTRY_POINT_EXIT program and the channel and data-conversion exits installed, you can enable your channel receive and send exit using the following MQSC command:

```
ALTER CHANNEL(CHAN) CHLTYPE(SDR) SENDEXIT(MY_CHANNEL_SEND_EXIT)
ALTER CHANNEL(CHAN) CHLTYPE(SDR) RCVEXIT(MY_CHANNEL_RCV_EXIT)
```

The Data Conversion exit will be called by WebSphere MQ when an MQGET is done with conversion enabled (MQGMO_CONVERT) and the message format name is MY_FORMAT.
Installing non-PIC exit in WebSphere MQ SRLs

```
mqlinksrls myexitdir
```

To remove all user exits from the WebSphere MQ product SRLs and restore WebSphere MQ product SRLs to their as-shipped state, rerun mqlinksrls without any arguments:
```
mqlinksrls
```

The mqlinksrls script saves a log file called mqlinksrls.log in $MQNSKVARPATH/errors.

Installing an exit in the WebSphere MQ non-native TNS library

When compiled, exit functions must be bound directly into the target executable or library to be accessible by WebSphere MQ. The TACL macro BEXITE is used for this purpose. This procedure modifies the target executable, so ensure you make a backup copy of the target executable or library before using the macro. For example:
```
NMLDUSRL OBJECTS EXITS
MAKEPSRL EXITS $VOL.ZMQSLIB NEWMQSRL
NMLDEXES $VOL.ZMQSLIB.NEWMQSRL $VOL.ZMQSEXE
NMCSSAMP AMQSGET0
AMQSGET NMLDPSSRL AMQSGET
NMLDEXIT Object-File Exit-Object-File
```

For example, to bind the sample data conversion exit into the sample MQSGETA, follow these steps:

1. Compile the exit function. For example:
```
CSAMP AMQSVFCN
```

2. Compile the MQLOAD entry point function. For example:
```
CSAMP AMQSLXPO
```

3. Compile the get application. For example:
```
CSAMP AMQSGET0
```

4. Bind the get application. For example:
```
BSAMP AMQSGET
```

5. Bind the exit function into the get application. For example:
```
BEXITE AMQSGET AMQSVFCO
```

6. Bind the entry point function into the get application. For example:
Installing exit in WebSphere MQ non-native TNS library

Alternatively, if all applications need to have this data conversion exit, the following steps create both a user library and an application with the exit bound in.

1. Compile the exit function. For example:

```
CSAMP AMQSVFCN
```

2. Compile the MQLOAD entry point function. For example:

```
BEXITE Target-Executable-Or-Library Source-Exit-File-Or-Library
CSAMP AMQSVFCN
CSAMP AMQSLXPO
CSAMP AMQSGET0
BSAMP AMQSGET
BEXITE AMQSGET AMQSVFCO
BEXITE AMQSGET AMQSLXPO
CSAMP AMQSVFCN
CSAMP AMQSLXPO
```

3. Bind the exit function into the user library. For example:

```
CSAMP AMQSGET0
```

4. Bind the exit function into the user library. For example:

```
BEXITE ZMQSLIB.MQMLIB AMQSVFCO
```

5. Bind the get application with the modified library. For example:

```
CSAMP AMQSGET0
BEXITE ZMQSLIB.MQMLIB AMQSVFCO
BEXITE ZMQSLIB.MQMLIB AMQSLXPO
BSAMP AMQSGET
```
Installing exit in WebSphere MQ non-native TNS library
Appendix K. Setting up communications

This appendix describes how to configure channels that use TCP/IP or SNA LU 6.2 as the communications protocol. The information provided is specific to the NonStop OS platform. For information about how to configure channels on other WebSphere MQ platforms, and for general information about how to manage channels, see [WebSphere MQ Intercommunication](#) for message channels and [WebSphere MQ Clients](#) for MQI channels.

The appendix contains the following sections:

- “Supported communications protocols”
- “Configuring TCP/IP channels”
- “Configuring SNA LU 6.2 channels” on page 482
- “Channel initiators” on page 484

### Supported communications protocols

WebSphere MQ for HP NonStop Server supports two communications protocols:

- TCP/IP
- SNA LU 6.2

The TRPTYPE parameter on the DEFINE CHANNEL command specifies which communications protocol a channel uses. If you omit the parameter, the channel uses TCP/IP by default. A queue manager can use both communications protocols, with some channels using TCP/IP and the remaining channels using SNA LU 6.2.

### Configuring TCP/IP channels

This section describes how to configure TCP/IP channels for a WebSphere MQ for HP NonStop Server queue manager. To configure a TCP/IP channel, you must create a channel definition at both ends of the channel. In addition, a queue manager must have at least one listener to listen for incoming requests to start channels.

#### Configuring the calling end of TCP/IP channels

The calling end of a channel is the end that starts the channel by sending a request to the remote queue manager. The definition of a channel at the calling end must contain the CHLTYPE parameter to specify the type of the channel at that end, and the CONNAME parameter to specify a connection name. The connection name is the host name or IP address of the system on which the remote queue manager is running. Optionally, the connection name can include a port number. See the following DEFINE CHANNEL command, for example:

```
DEFINE CHANNEL(GANYMEDE.TO.PHOBOS) +
    CHLTYPE(SDR) +
    CONNAME('MARS(1822)') +
    TRPTYPE(TCP)
```

This command defines the sender end of a message channel called GANYMEDE.TO.PHOBOS. The connection name is MARS(1822), where MARS is
Setting up communications

the host name of the system on which the remote queue manager is running, and 1822 is the number of the port on which a listener for that queue manager is listening.

If the connection name does not include a port number, the command uses the port number specified by the Port entry in the TCP stanza in the queue manager configuration file, qm.ini. If there is no Port entry in the queue manager configuration file, the command uses port number 1414 by default. Port number 1414 is assigned to WebSphere MQ by IANA, the Internet Assigned Numbers Authority.

For more information about the Port entry in the queue manager configuration file, see “TCP” on page 140.

Configuring the responding end of TCP/IP channels

The responding end of a channel is the end that responds to an incoming request to start the channel. The definition of a channel at the responding end must contain the CHLTYPE parameter to specify the type of the channel at that end. See the following DEFINE CHANNEL command, for example:

```
DEFINE CHANNEL (PHOBOS.TO.GANYMEDE) +
   CHLTYPE (RCVR) +
   TRPTYPE (TCP)
```

This command defines the receiver end of a message channel called PHOBOS.TO.GANYMEDE.

A queue manager must have at least one listener to listen for incoming requests to start channels. Only the listener supplied with WebSphere MQ for HP NonStop Server is supported. You cannot use the OSS inet daemon or the NonStop OS listener for TCP/IP.

You can start a WebSphere MQ listener in either of the following ways:

- By entering the `runmqlsr` command at an OSS shell command prompt or a TACL command prompt
- By starting a TCP/IP listener, which is configured as a server class within Pathway

The remainder of this section describes each of these options.

Starting a WebSphere MQ listener by entering the `runmqlsr` command at a command prompt

You can start a WebSphere MQ listener for a queue manager by entering the `runmqlsr` command at an OSS shell command prompt or a TACL command prompt. Here is an example of the `runmqlsr` command:

```
runmqlsr -t tcp -p 1822 -m GANYMEDE
```

The `-t` parameter is required and specifies the communications protocol to be used, TCP/IP in this case. The `-m` parameter is optional and identifies the queue manager. If you omit the parameter, a listener is started for the default queue manager. The `-p` parameter is also optional and identifies the number of the port on which the listener listens. If you omit the parameter, the listener listens on the port specified by the Port entry in the TCP stanza in the queue manager configuration file. If there is no Port entry in the queue manager configuration file, the listener listens on port number 1414 by default. If you are running more than one listener on a system, each listener must listen on a different port.
By default, the listener starts each channel using an MCA that runs as a thread. However, if you include the -u parameter on the runmqslr command, as shown in the following example, the listener starts each channel using an MCA that runs as a process:

```
runmqslr -t tcp -p 1822 -u -m GANYMEDE
```

You can use the -b parameter on the runmqslr command to specify the maximum number of connection requests that can be waiting to be accepted by the listener, as shown in the following example:

```
runmqslr -t tcp -p 1822 -b 10 -m GANYMEDE
```

If you omit the parameter, the maximum number of connection requests is specified by the ListenerBacklog entry in the TCP stanza in the queue manager configuration file. If there is no ListenerBacklog entry in the queue manager configuration file, the maximum number of connection requests is 5 by default.

For more information about the runmqslr command, see "runmqslr (run listener)" on page 279. For more information about the Port and ListenerBacklog entries in the queue manager configuration file, see "TCP" on page 140.

If you start a listener by entering the runmqslr command at a command prompt, the listener does not end automatically when the queue manager ends. To stop all WebSphere MQ listeners for a queue manager that is not running, enter the endmqslr command at an OSS shell command prompt or a TACL command prompt. Here is an example of the endmqslr command:

```
endmqslr -m GANYMEDE
```

The -m parameter is optional and identifies the queue manager. If you omit the parameter, the command stops all listeners for the default queue manager. For more information about the endmqslr command, see "endmqslr (end listener)" on page 271.

**Starting a WebSphere MQ listener by starting a TCP/IP listener**

You can start a WebSphere MQ listener for a queue manager by starting a TCP/IP listener, which is configured as a server class within Pathway. When you create a queue manager, a default TCP/IP listener, with server class MQS-TCPLIS00, is configured automatically, but you can configure more.

The program that runs when you start a TCP/IP listener is runmqslr, and you can use the ARGLIST attribute of the TCP/IP listener to specify the parameters for runmqslr. These parameters are the same as those for the runmqslr command. For more information about the ARGLIST attribute of a TCP/IP listener, see "The ARGLIST attribute of a TCP/IP listener" on page 120.

A TCP/IP listener ends automatically when the queue manager ends. Typically, a TCP/IP listener uses the same home terminal as the other server processes of a queue manager. Because a home terminal is always available, a TCP/IP listener is generally more suitable for use in production than a listener started by entering the runmqslr command at a command prompt.

**Specifying the TCP/IP process for TCP/IP channels**

On NonStop OS, you can configure multiple TCP/IP interfaces for applications to use. Each TCP/IP interface is a NonStop OS process-pair and is referred to a TCP/IP process. On a NonStop OS system, you might want to configure more than
Setting up communications

one TCP/IP process in order to spread the communications processing load across more than one CPU, or to segregate certain network traffic.

The name of the default TCP/IP process is $ZTC0. Unless you specify otherwise, MCAs and the WebSphere MQ listener use the default TCP/IP process.

To cause caller MCAs to use a different TCP/IP process, you must modify the process management rules configuration file, qmproc.ini. The TransportName entry in the appropriate channel rules stanza specifies the name of a TCP/IP process. Consider the following ChlRule3-ChannelProtocolMatch stanza, for example:

```
ChlRule3-ChannelProtocolMatch:
    ChannelProtocolMatch=TCP/IP
    TransportName=$ZTC4
```

This stanza causes all caller MCAs to use the TCP/IP process $ZTC4 instead of the default TCP/IP process.

If you want the MCA at the calling of a specific channel to use a different TCP/IP process, you must create a ChlRule1-ChannelNameMatch stanza with a ChannelNameMatch entry, as in the following example:

```
ChlRule1-ChannelNameMatch:
    ChannelNameMatch=GANYMEDE.TO.PHOBOS
    TransportName=$ZTC3
```

This stanza causes the MCA at the calling end of the channel GANYMEDE.TO.PHOBOS to use the TCP/IP process $ZTC3.

You can use the -g parameter on the runmqlsr command to specify the TCP/IP process to be used by a listener and all the responder MCAs that implement channels started by the listener. For example, the following command, entered at an OSS shell command prompt, causes the listener and responder MCAs to use the TCP/IP process $ZTC4 instead of the default TCP/IP process:

```
runmqlsr -t tcp -p 1822 -g $ZTC4 -m GANYMEDE
```

The following command, entered at a TACL command prompt, performs the same function:

```
runmqlsr -t tcp -p 1822 -g $ZTC4 -m GANYMEDE
```

The TCP/IP keep alive function

By default, TCP/IP checks periodically whether the other end of a TCP/IP connection is still available. If it is not available, the channel that was using the connection ends.

If you don’t want TCP/IP to perform these checks, you must add the following entry to the TCP stanza in the queue manager configuration file, qm.ini:

```
KeepAlive=NO
```

Configuring SNA LU 6.2 channels

This section describes how to configure SNA LU 6.2 channels for a WebSphere MQ for HP NonStop Server queue manager. To configure an SNA LU 6.2 channel, you must create a channel definition at both ends of the channel.

However, before you can use SNA LU 6.2 channels, you must configure an SNA node on your NonStop OS system. You must configure the node in such a way
that a caller MCA running on your system can establish an LU 6.2 conversation with a responder MCA running on a remote system. You must also configure the node so that a caller MCA running on a remote system can establish an LU 6.2 conversation with a responder MCA running on your system. For information about how to configure an SNA node in this way, see the appropriate SNAX/APC or Insession ICE documentation.

Configuring the calling end of SNA LU 6.2 channels

The calling end of a channel is the end that starts the channel by sending a request to the remote queue manager. The definition of a channel at the calling end must contain the following parameters:

CHLTYPE
This parameter specifies the type of the channel at the calling end.

CONNAME
This parameter specifies the network qualified name of the partner LU, which is the LU supporting the remote queue manager.

MODENAME
This parameter specifies the mode name for the LU 6.2 session to be allocated for the conversation between the caller MCA and the responder MCA on the remote system.

TPNAME
This parameter specifies the TP name that identifies the responder MCA to be attached at the other end of the conversation. The responder MCA must be defined as a transaction program on the remote system. The definition of the transaction program includes the TP name. For information about how to define a transaction program at the responding end of a channel on platforms other than NonStop OS, see WebSphere MQ Intercommunication.

TRPTYPE
This parameter specifies the communications protocol to be used. The value must be LU62.

See the following DEFINE CHANNEL command, for example:

```
DEFINE CHANNEL (GANYMEDE.TO.TITAN) +
   CHLTYPE (SDR) +
   CONNAME (SOLAR.SATURN) +
   MODENAME ('#INTER') +
   TPNAME (TITAN) +
   TRPTYPE (LU62)
```

This command defines the sender end of a message channel called GANYMEDE.TO.TITAN. The network qualified name of the partner LU is SOLAR.SATURN, and the mode name for the session to be allocated for the conversation is #INTER. The TP name of the responder MCA to be attached at the other end of the conversation is TITAN.

Configuring the responding end of SNA LU 6.2 channels

The responding end of a channel is the end that responds to an incoming request to start the channel. The definition of a channel at the responding end must contain the CHLTYPE parameter to specify the type of the channel at that end, and the TRPTYPE parameter to specify the communications protocol to be used. See the following DEFINE CHANNEL command, for example:
# Setting up communications

```plaintext
DEFINE CHANNEL(TITAN.TO.GANYMEDE) +
  CHLTYPE(RCVR) +
  TRPTYPE(LU62)
```

This command defines the receiver end of a message channel called TITAN.TO.GANYMEDE.

## Channel initiators

When you create a queue manager, a default channel initiator is configured as server class MQS-CHANINIT00 in Pathway. The default channel initiator starts automatically when the queue manager starts, and typically uses the same home terminal as the other server processes of the queue manager.

Under normal circumstances, a queue manager needs only one channel initiator. However, you can configure more channel initiators in Pathway, and you can also start a channel initiator by entering the `runmqchi` command at an OSS shell command prompt or a TACL command prompt. Here is an example of the `runmqchi` command:

```plaintext
runmqchi -q INITQ -m GANYMEDE
```

The `-q` parameter is optional and specifies the name of the initiation queue. If you omit the parameter, the channel initiator uses the system initiation queue, `SYSTEM.CHANNEL.INITQ`. The `-m` parameter is also optional and identifies the queue manager. If you omit the parameter, a channel initiator is started for the default queue manager. For more information about the `runmqchi` command, see "runmqchi (run channel initiator)" on page 276.

The program that runs when you start a channel initiator that is configured in Pathway is `runmqchi`, and you can use the `ARGLIST` attribute of the server class to specify the parameters for `runmqchi`. These parameters are the same as those for the `runmqchi` command. For more information about the `ARGLIST` attribute of a channel initiator, see "The `ARGLIST` attribute of a channel initiator" on page 120.

If you use a channel initiator to start a channel, the `TriggerData` attribute of the transmission queue can specify the name of the channel. The `ProcessName` attribute of the transmission queue does not need to identify a process whose `UserData` attribute specifies the name of the channel.
Appendix L. 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.
Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM United Kingdom Laboratories,
Mail Point 151,
Hursley Park,
Winchester,
Hampshire,
England
SO21 2JN.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Programming License Agreement, or any equivalent agreement between us.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM’s application programming interfaces.

Trademarks

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

- CICS
- FFST
- First Failure Support Technology
- IBM
- IMS
- MQSeries
- SupportPac
- WebSphere
- z/OS

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.
Microsoft and Windows are trademarks of Microsoft Corporation in the United States, other countries, or both.

Intel and Itanium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, or service names may be trademarks or service marks of others.
Index

Special characters

.ini files
   See configuration files

A

ABORTTRANSACTION call 181
   access control
      how it is implemented 145
         using the OAM 149
   access control checks, preventing 153
   access settings
      displaying 153
         dumping 152
ACTION keyword, rules table 191
   adding server classes 124
   administration
      authority 143
      control commands 23
         introduction to 15
      local, definition of 15
   MQAI, using 60
   MQSC commands
      overview 16
         performing local administration 32
      object name transformation 19
      PCF commands 59
      queue manager name transformation 18
      remote administration, definition of 15
      remote objects 61
      understanding WebSphere MQ file names 18
         using control commands 16
         using PCF commands 16
   agent attributes 199
   alias queues
      creating an alias queue 45
         DEFINE QALIAS command 45
      remote queues as queue manager aliases 71
      reply-to queues 71
      working with alias queues 44
   AllQueueManagers stanza, mqs.ini 135
   alter WebSphere MQ object attributes command (altmqfls)
      See altmqfls (alter WebSphere MQ object attributes) command (continued)
   alter WebSphere MQ object attributes command
      changing the message overflow threshold 92
      checkpoint nonpersistent messages, queue server option 94
      definition 244
      increasing the capacity of a local queue 91
      load at startup, queue server option 94
      lock in memory, queue server option 94
      measuring queue depth 93
      moving the files associated with a queue 90
      putting a local queue into maintenance mode 95
      reassigning a WebSphere MQ object to another queue server 89
      setting the browse threshold 93
      specifying the location of message overflow files 91
   altmqsar (alter WebSphere MQ user information) command
      definition 249
   amqslq, the sample DLQ handler 188
   API exits
      configuring 373
      introduction 371
      MQ_BACK_EXIT call 397
         MQ_CLOSE_EXIT call 398
         MQ_CMNT_EXIT call 399
         MQ_CONNX_EXIT call 400
         MQ_DISC_EXIT call 402
         MQ_GET_EXIT call 403
         MQ_INIT_EXIT call 405
         MQ_INQ_EXIT call 406
         MQ_OPEN_EXIT call 408
         MQ_PUT_EXIT call 409
         MQ_PUT1_EXIT call 411
         MQ_SET_EXIT call 413
         MQ_TERM_EXIT call 415
         MQACH structure 379
         MQAXC structure 382
         MQAXP structure 386
         MQXEP call 394
         reference information 377
      APICallerType field
         MQAXP structure 390
      ApiExitCommon stanza, mqs.ini 137
         ApiExitLocal stanza, qm.ini 141
      ApiExitTemplate stanza, mqs.ini 137
      application queues
         defining application queues for triggering 47
      application rules 201
      applications
         affects of browsing persistent messages in queue 93
         building C applications 458
   applications (continued)
      building C++ applications 459
      building COBOL applications 461
      building non-native applications 461
      common programming errors 223
      design considerations 230
      failure recovery 105
      managing concurrent transactions 181
      message length, effects on performance 230
      MQI local administration, support for 31
      native non-PIC
         building C applications 459
         building C++ applications 460
      native PIC
         building C applications 458
         building C++ applications 460
      persistent messages, effect on performance 230
      receiving messages 4
         restrictions using fastpath binding 97
      retrieving messages from queues 5
      running non-PIC 458
      running PIC 458
      samples supplied 463
      searching for messages, effect on performance 231
      sending messages 4
      threads, application design 231
      triggered 455
      trusted 96
      tuning 85
      user exits for Shared Resource Library (SRL) applications 471
         using OpenTMF 104
   APPLIDAT keyword, rules table 190
   ApplName field
      MQAXC structure 384
   APPLNAME keyword, rules table 190
   AppType field
      MQAXC structure 385
   APPLTYPE keyword, rules table 190
   architecture of WebSphere MQ for HP NonStop Server 77
   attributes
      agent 199
      channel attributes, displaying using the Monitoring Panels 56
      default process 198
      LIKE attribute, DEFINE command 42
      local queue attributes, changing 42
      object attributes and PCF commands 60
   PATHWAY attributes, changing 126
   queue attributes
      displaying using DISPLAY QUEUE 41
      displaying using the Monitoring Panels 53
attributes (continued)
queue manager attributes changing 40
displaying using DISPLAY QMGR 39
displaying using the Monitoring Panels 50
queues, examples of attributes 7
server class attributes changing 121
description of 116
audit trail size, TMF 185
audited message overflow files, queue server option 94
authority administration 143
alternate user 147
cntext 148
Authority field MQZAD structure 354
Authority parameter check authority call 324
get authority call 338
get explicit authority call 341
set authority call 349
AuthorityBuffer parameter enumerate authority data call 335
AuthorityBufferLength parameter enumerate authority data call 335
AuthorityDataLength parameter enumerate authority data call 335
authorization service defining to the queue manager 311
entry points check authority 323
copy all authority 328
delete authority 331
enumerate authority data 334
get authority 337
get explicit authority 340
initialize authorization service 343
refresh all authorizations 346
set authority 348
terminate authorization service 351
interface 313
introduction 12
overview 311
stanzas 312
structure data types MQZAD 353
MQZED 356
authorizations migrating authorization data from MQSeries 312
MQI calls 157
refreshing the OAM after changing a user's authorization 311
specification tables 157
automatic definition of channels 66
AutoSYNC 216
availability configuring for 111
definition 99

B
back out a unit of work 179
backing up a queue manager 214
BEGINTRANSACTION call 181
browse threshold of a local queue 93
browsing persistent messages 93
browsing queues 43
building C applications 458
C++ applications 459
COBOL applications 461
non-native applications 461
built in formats, data conversion 72

C
C applications, building 458
C++ applications, building 459
case sensitivity of control commands examples 23
reference information 239
csid.tbl, data conversion 72
cert.pem file 170
certificates See digital certificates certificate revocation list file, crl.pem 170
certificate store, cert.pem 170
ChainAreaLength field MQACH structure 380
changing
CCSID of a queue manager 73
configuration information in configuration files 130
in Pathway 115
local queue attributes 42
PATHWAY attributes 126
queue manager attributes 40
server class attributes 121
default queue manager 27
channel definitions file 105
channel events 434
channel exits 155
channel initiator failure recovery 105
function 83
starting 484
Channel Menu, Monitoring Panels 54
channel rules 202
channel server failure recovery 105
function 82
recovery and restart 215
channels administering a remote queue manager from a local one 63
attributes, displaying using the Monitoring Panels 56
auto-definition of 66
Channels stanza, qm.ini 139
commands for channel administration 442
defining channels for remote administration 64
escape command authorizations 160
introduction 10

channels (continued)
preparing channels for remote administration 64
role in remote queuing 61
security 153
SNA LU 6.2 configuring the calling end 483
configuring the responding end 483
configuring, introduction 482
starting 65
TCP/IP configuring the calling end 479
configuring the responding end 480
configuring, introduction 479
Channels stanza, qm.ini 139
character code sets, updating 72
checkpoint nonpersistent messages, queue server option 94
clearing a local queue 42
client channel definitions file 105
clients, WebSphere MQ 11
clusters, queue manager See queue manager clusters
COBOL applications building 461
code supplied with product 84
coded character sets, specifying 72
command files 35
command queue command server mandatory for remote administration 64
description of 9
command server commands for command server administration 441
displaying the status 66
failure recovery 105
function 83
remote administration 66
starting the command server 66
stopping the command server 67
command sets comparison of sets 441
control commands 23
MQSC commands 32
PCF commands 59
commands commands for queue manager administration 441
comparison of command sets 441
control commands See control commands
dmqmqaui 152
dsmqaut 153
for channel administration 442
for command server administration 441
for process administration 442
for queue administration 441
issuing MQSC commands from a text file 35
other commands 443
PCF commands 59
configuration files (continued)
ExitProperties stanza, mqs.ini 136
mq.ini, description of 142
priorities 131
proc.ini, description of 132
qm.ini, description of 134
qmproc.ini, description of 197
RestrictedMode stanza, mqs.ini 143
Service stanza, qm.ini 143
ServiceComponent stanza,
qm.ini 143
TCP stanza, qm.ini 140
configuration information
in configuration files 130
in Pathway 115
introduction 115
configuring
API exits 373
communications 479
for availability 111
for data integrity 111
installable services and service
components 306
SNA LU 6.2 channels
calling end 483
introduction 482
responding end 483
SNA node 482
TCP/IP channels
calling end 479
introduction 479
responding end 480
the entropy daemon 165
TMF 184
ConnectionName field
MQAXC structure 384
constants, values of 417
API exit caller type
(MQXACT_*) 419
API exit chain header length
(MQACH_*) 417
API exit chain header structure
identifier (MQACH_*) 417
API exit chain header version
(MQACH_*) 418
API exit context structure identifier
(MQAXC_*) 418
API exit context version
(MQAXC_*) 418
API exit environment (MQXE_*) 420
API exit function identifier
(MQXF_*) 420
API exit parameter structure identifier
(MQAXP_*) 418
API exit parameter version
(MQAXP_*) 418
API exit problem determination area
(MQXPDA_*) 420
authority data structure identifier
(MQZAD_*) 421
authority data version
(MQZAD_*) 421
authority service authorization type
(MQZAO_*) 421
authority service entity type
(MQZAET_*) 421
constants, values of (continued)
availability service version
(MQZAS_*) 422
collection codes (MQCC_*) 418
continuation indicator
(MQZCI_*) 422
descriptor structure identifier
(MQQED_*) 422
descriptor version
(MQQED_*) 422
exit identifier (MQXT_*) 421
exit reason (MQXR_*) 420
exit response (MQXCC_*) 420
exit user area (MQXUA_*) 421
feedback (MQFBL_*) 418
function identifier, all services
(MQZID_*) 422
function identifier, authority service
(MMQZID_*) 423
function identifier, name service
(MMQZID_*) 423
function identifier, user id service
(MMQZID_*) 423
initialization options (MQZIO_*) 423
lengths of character string and byte
fields (MQ_*) 417
name service version
(MMQZNS_*) 423
object type (MQOT_*) 419
reason codes (MQRC_*) 419
secondary exit response
(MMQXR_*) 421
security identifier (MQSID_*) 419
start-enumen?ation indicator
(MMQZE_*) 423
termination options (MQZTO_*) 423
user id service version
(MMQZUS_*) 423
context authority 148
Continuation parameter
check authority call 326
copy all authority call 329
delete authority call 332
enumerate authority data call 335
get authority call 338
get explicit authority call 341
initialize authorization service
call 344
initialize name service call 361
insert name call 364
lookup name call 366
MQ_GET_EXIT call 403
MQZ_DELETE_NAME call 359
MQZEP call 321
set authority call 349
terminate authorization service
call 352
terminate name service call 369
ComponentData parameter
check authority call 326
copy all authority call 329
delete authority call 332
enumerate authority data call 335
get authority call 338
get explicit authority call 341
initialize authorization service
call 344
initialize name service call 361
insert name call 364
lookup name call 366
MQZ_DELETE_NAME call 358
set authority call 349
terminate authorization service
call 351
terminate name service call 368
ComponentDataLength parameter
initialize authorization service
call 343
initialize name service call 360
configuration files
AllQueueManagers stanza,
mqs.ini 135
ApiExitCommon stanza, mqs.ini 137
ApiExitLocal stanza, qm.ini 141
ApiExitTemplate stanza, mqs.ini 137
backing up of 28
Channels stanza, qm.ini 139
DefaultQueueManager stanza,
mqs.ini 136
editing 130
example mqs.ini file 131
example proc.ini file 132
example qm.ini file 134
example qmproc.ini file 205
ExitPath stanza, qm.ini 141
control commands (continued)
definitions (continued)
dspmqfls 264
dspmqtr 267
dspmqusr 268
endmqcsv 270
endmqsr 271
endmqm 272
endmqtr 274
runmqcli 272
runmqch 276
runmqd 278
runmqsl 279
runmqsc 281
runmqsm 284
setmqaut 285
strmqcsv 292
strmqm 293
strmqtr 294
entering
at an OSS shell command
prompt 239
case sensitivity, examples 23
case sensitivity, reference
information 239
help with syntax 242
runmqsc, using interactively 33
starting a queue manager 29
stopping a queue manager 29
using
introduction 23
reference information 239
using with Pathway 83
controlled shutdown of a queue
manager 29
CorreId, performance
considerations 231
CPU
configuring for availability 111
high usage 96
queue server 89
tuning 85
create code for data conversion
exit command (crtmqcvx)
See crtmqcvx (create code for data
conversion exit) command
create queue manager command
(crtmqm)
See crtmqm (create queue manager)
command
creating
a default queue manager 27
a dynamic queue 4
a local definition of a remote
queue 68
a local queue 40
a model queue 46
a predefined queue 4
a process definition 48
a queue manager 24
a service component 308
a transmission queue 70
an alias queue 45
the queue manager’s SSL files 171
crl.pem file 170
crtmqcvx (create code for data conversion
exit) command
definition 251
crtmqm (create queue manager)
command
definition 253
current queue depth, determining 41

D
data conversion
built in formats 72
ccsid.tbl, uses for 72
ConvEBCDICNewline attribute, AllQueueManagers stanza 135
converting user defined message
formats 73
default data conversion 73
EBCDIC NL character conversion to
ASCII 135
exit 73
introduction 72
mechanism for supporting all
exits 453
updating coded character sets 72
data integrity
configuring for 111
definition 99
data types
elementary
MQHCONFIG 322
PMQFUNC 322
MQI structure data types 447
structure
MQACH 379
MQAXC 382
MQAXP 386
MQLXP 472
MQZAD 353
MQZED 356
databases
consistency 102
critical database files 105
external consistency 103
protected by TMF 102
dead letter header, MQDLH 187
dead letter queue handler
ACTION keyword, rules table 191
action keywords, rules table 191
APPLDAT keyword, rules table 190
APPLNAME keyword, rules
table 190
APPTYPE keyword, rules table 190
control data 188
DESTQ keyword, rules table 190
DESTQM keyword, rules table 190
elementary arguments 190
example of a rules table 195
FEEDBK keyword, rules table 190
FORMAT keyword, rules table 190
FWDQ keyword, rules table 191
FWDQM keyword, rules table 191
HEADER keyword, rules table 192
INPUTQ keyword, rules table 188
INPUTQM keyword, rules table 189
invoking the DLQ handler 187
MSGTYPE keyword, rules table 190
dead letter queue handler (continued)
pattern-matching keywords, rules
table 190
patterns and actions (rules) 189
PERSIST keyword, rules table 190
processing all DLQ messages 195
processing rules, rules table 194
PUTAUT keyword, rules table 192
REASON keyword, rules table 190
REPLYQ keyword, rules table 191
REPLYQM keyword, rules table 191
RETRY keyword, rules table 192
RETRYINT keyword, rules table 189
rule table conventions 192
rules table, description of 188
sample, amqslq 188
syntax rules, rules table 193
USERID keyword, rules table 191
WAIT keyword, rules table 189
dead letter queues
defining a dead letter queue 41
description of 8
MQDLH, dead letter header 187
specifying 25
debugging
command syntax errors 224
common command errors 224
common programming errors 223
further checks 225
preliminary checks 221
default data conversion 73
default process attributes 198
default process management rules
configuration file (proc.ini)
See proc.ini (default process
management rules configuration file)
default transmission queues 71
DefaultQueueManager stanza,
proc.ini 136
defaults
changing the default queue
manager 27
creating a default queue manager 27
queue manager 25
reverting to the original default queue
manager 27
system and default objects
introduction 11
list of 427
transmission queue 26
deferred message server 83
defining
a local definition of a remote
queue 68
a local queue 40
a model queue 46
a TCP/IP channel
calling end 479
responding end 480
an alias queue 45
an initiation queue 48
an SNA LU 6.2 channel
calling end 483
responding end 483
queues 7
the authorization service to the queue
manager 311
delete queue manager command (dlmqmq)

See dlmqm (delete queue manager) command

deleting
  a local queue 43
  a queue manager 439
  using dlmqm 30
  a server class 126

DESTQ keyword, rules table 190
DESTQQM keyword, rules table 190

digital certificates
  generating a request 168
  importing 170
  working with 167

directory of a queue manager 26
directory structure 429

disaster recovery 216

disk volume, partitioning a queue file or queue overflow file 92

display authority command (dspmqaut)

See dspmq (display authority) command

display command server command (dspmqcsv)

See dspmqcsv (display command server) command

display formatted trace command (dspmqtrc)

See dspmqtrc (display formatted trace) command

display queue managers command (dspmq)

See dspmq (display queue managers) command

display WebSphere MQ object attributes command (dspmqfls)

See dspmqfls (display WebSphere MQ object attributes) command

display WebSphere MQ user information command (dspmqusr)

See dspmqusr (display WebSphere MQ user information) command

displaying
  channel attributes using the Monitoring Panels 56
  process definitions 49
  queue attributes
    using DISPLAY QUEUE 41
    using the Monitoring Panels 53
  queue manager attributes
    using DISPLAY QMGRC 39
    using the Monitoring Panels 50

distributed queuing, incorrect output 227

DILLs
  building C applications 458
  building C++ applications 460
dlmqm (delete queue manager) command

definition 256
dspmqaut (dump authority) command

definition 257
dspmq (display queue managers) command

definition 259
dspmqaut (display authority) command

definition 260
dspmqcsv (display command server) command

definition 263
dspmqfls (display WebSphere MQ object attributes) command

definition 264
dspmqtrc (display formatted trace) command

definition 267
dspmqusr (display WebSphere MQ user information) command

definition 268
dump authority command (dpmqaut)

See dspmqaut (dump authority) command
dynamic definition of channels 66
dynamic queues 4

E

EBCDIC NL character conversion to ASCII 135
elementary data types
MQCONF 322
PMQFUNC 322
EMS (Event Management Service) events
  alternative collector, specifying 438
  default collector 438
  introduction 435
  setting the MQEMSEVENTS environment variable 437
  writing programs to process 438
  end command server command (endmqcsv)

See endmqcsv (end command server) command
dynamic definition of channels 35
endmqcsv (end command server) command

definition 270
endmqslr (end listener) command

definition 271
endmng (end queue manager) command

definition 272
endmqtrc (end trace) command

definition 274

ENDTRANSACTION call 181
entering control commands
at an OSS shell command prompt 239

entering control commands

See instruction events examples
creating a transmission queue 70
mq.ini file 131
proc.ini file 132
programming errors 223
qm.ini file 134
qmproc.ini 205

execution controller
  failure recovery 105
  function 79
  recovery and restart 215
  role in managing processes 197
  exit, installing 476

ExitChainAreaPtr field
MQAXP structure 392

ExitData field
MQAXP structure 387

ExitId field
MQAXP structure 387

ExitInfoName field
MQACH structure 381
ExitInfoName field (continued)  
MQAXP structure 391  
ExitPath stanza, qm.ini 141  
ExitPDArea field  
MQAXP structure 391  
ExitProperties stanza, mqs.ini 136  
ExitReason field  
MQAXP structure 387  
ExitReason parameter  
MQXEP call 394  
ExitResponse field  
MQAXP structure 388  
MQLPX structure 473  
ExitResponse2 field  
MQAXP structure 389  
exits  
API exits 371  
channel exits 155  
cluster workload exit 12  
data conversion exit 73  
loading user exits 472  
types of user exit 12  
user exits for Shared Resource Library (SRL) applications 471  
user exits in the non-PIC environment 472  
user exits in the PIC environment 471  
ExitUserArea field  
MQAXP structure 390  
extending queue manager facilities 12  
files (continued)  
principal database (continued)  
role in access control 146  
process management rules configuration file, qmproc.ini 197  
product files introduction 84  
locating 456  
queue files critical database files 105  
introduction 88  
moving 90  
partitioning 92  
queue manager configuration file 134  
queue overflow files critical database files 105  
introduction 88  
moving 90  
partitioning 92  
touch files for queues critical database files 105  
introduction 88  
moving 90  
understanding files names 18  
WebSphere MQ configuration file 131  
Filter parameter enumerate authority data call 334  
first failure support technology (FFST) See FFST (first failure support technology)  
fix command features 34  
floating point 455  
FORMAT keyword, rules table 190  
Function field  
MQAXP structure 392  
Function parameter  
MQXEP call 394  
MQZEP call 321  
FWDQ keyword, rules table 191  
FWDQM keyword, rules table 191  
G  
generic profiles 151  
global units of work coordinating with TMF 181  
definition of 180  
introduction 14  
grant or revoke authority command (setmquat)  
See setmquat (grant or revoke authority) command  
grants creating and managing 148  
nobody user group 147  
role in access control 146  
guidelines for creating queue managers 24  
H  
Hconfig field  
MQAXP structure 392  
Hconfig parameter  
initialize authorization service call 343  
initialize name service call 360  
MQXEP call 394  
MQZEP call 321  
terminate authorization service call 351  
terminate name service call 368  
HEADER keyword, rules table 192  
help with control command syntax 242  
home terminal of a queue manager 26  
I  
immediate shutdown of a queue manager 29  
indirect mode, runmqsc command 67  
inclusion rules defining 48  
description of 8  
INPUTQ keyword, rules table 188  
INPUTQM keyword, rules table 189  
installable services  
add component entry point function 321  
authorization service interface 313  
introduction 12  
overview 311  
authorization service entry points check authority 323  
copy all authority 328  
delete authority 331  
enumerate authority data 334  
get authority 337  
get explicit authority 340  
initialize authorization service 343  
refresh all authorizations 346  
set authority 348  
terminate authorization service 351  
authorization service structure data types  
MQZAD 353  
MQZED 356  
configuring services and components 306  
external data types  
MQHCONFIG 322  
PMQFUNC 322  
interface reference 319  
introduction 12  
list of 12  
name service  
interface 316  
introduction 13  
overview 315  
name service entry points delete name 358  
initialize name service 360  
insert name 363  
lookup name 365  
terminate name service 368  
overview 303
installable services (continued)
  service components
    component data 305
    creating your own 308
    initializing 306
    introduction 12
    multiple 308
    overview 304
    return codes 305
  service entry points 305
instrumentation events
  channel events 434
  description 433
  enabling 434
  event messages 435
  event queues
    introduction 9
    what happens when an instrumentation event occurs 434
  performance events 433
  queue manager events 433
  types of 433
interleaved application transactions 104
issuing MQSC commands
  from a text file 35
  to a remote queue manager 67
  using the runmqsc command 32

local queues (continued)
  clearing 42
  copying a local queue 42
  creating 40
  defining application queues for triggering 47
  definition of 9
  deleting 43
  increasing the capacity 91
  putting into maintenance mode 95
  specific queues used by WebSphere MQ
    8
    working with 40
  local units of work
    coordinating with TMF 182
    definition of 180
    introduction 13
lock in memory, queue server option 94
  logs, error log files 231
long running TMF transactions 182
LongMCAUserldLength field
  MQAXC structure 384
LongMCAUserldPtr field
  MQAXC structure 384
LongRemoteUserldLength field
  MQAXC structure 384
LongRemoteUserldPtr field
  MQAXC structure 384
messages (continued)
  message length, effects on performance 230
  message lengths 4
  nonpersistent 87
  not appearing on queues 225
  persistent 87
  persistent messages, effect on performance 230
  retrieval algorithms 5
  retrieving messages from queues 5
  sending and receiving 4
  undelivered 232
  variable length 231
migrating authorization data from MQSeries 312
model queues
  attributes, information specific to WebSphere MQ for HP NonStop Server 452
  creating 46
  DEFINE QMODEL command 46
  role in creating dynamic queues 4
  working with 46
monitoring
  current channels 57
  local queues 54
  TMF status 185
Monitoring Panels
  Channel Menu 54
  introduction 49
  Monitoring Panels server 83
  Queue Manager Menu 50
  Queue Menu 51
monitoring queue managers 433
MQ.* values 417
MQ.BACK_EXIT call 397
MQ.CLOSE_EXIT call 398
MQ.CMIT_EXIT call 399
MQ.CONNX_EXIT call 400
MQ.DISC_EXIT call 402
MQ.GET_EXIT call 403
MQ.INIT_EXIT call 405
MQ.INQ_EXIT call 406
MQ.LOAD_ENTRY_POINT_EXIT 472
MQ.OPEN_EXIT call 408
MQ.PUT_EXIT call 409
MQ.PUT1_EXIT call 411
MQ.SET_EXIT call 413
MQ.TERM_EXIT call 415
MQ.ACH structure 379
MQ.ACH.* values 379
MQ.AI (WebSphere MQ Administrative Interface) 60
MQAXC structure 382
MQAXC.* values 382
MQAXP structure 386
MQAXP.* values 386
MQBACK call
  implementing a local unit of work 182
  support for 450
MQBEGIN call 450
MQBO structure 447
MQCH structure 447
MQCLOSE call
  additional notes 451
MQCLOSE call (continued)
authorizations 158
support for 450
MQCMIT call
implementing a local unit of work 182
support for 450
MQCNIO structure
additional notes 448
support for 447
MQCNIO_FASTPATH_BINDING option 97
MQCONN call
authorizations 158
fastpath binding 97
support for 450
MQCONNNX call
fastpath binding 97
support for 450
MQDH structure 447
MQDISC call
additional notes 451
support for 450
MQDLH structure 447
MQDLHI, dead letter header 187
MQEMSEVENTS environment variable definition 445
enabling WebSphere MQ EMS events 437
MQGET call 450
MQGMO structure
additional notes 448
support for 447
MQGMO_SET_SIGNAL option 448
MQHCONFIG elementary data type 322
MQI (Message Queue Interface) calls
authorization specification tables 157
authorizations 157
support for 450
definition of 3
local administration support 31
queue manager calls 10
receiving messages 4
sending messages 4
MQH structure 447
MQINQ call
additional notes 451
support for 450
MQLPX structure 472
MQM group 144
MQMD structure
additional notes 449
support for 447
MQMDE structure 447
MQNSKOPPATH environment variable 445
MQNSKVARPATH environment variable 445
MQOD structure 447
MQOPEN call
additional notes 451
MQOPEN call (continued)
authorizations 158
support for 450
MQOR structure 447
MQOT_* values 354
MQPMO structure
additional notes 449
support for 447
MQPMR structure 447
MQPUT call 450
MQPUT1 call
authorizations 158
performance considerations 231
support for 450
MQQSH KEEPINT environment variable 446
MQQSMAXBATCHEXPIRE environment variable 446
MQQSMAXMSGSEXPIRE environment variable 446
MQQSI SITIMEOUT environment variable 446
MQRMH structure 447
MQRR structure 447
msq.ini (WebSphere MQ configuration file)
AllQueueManagers stanza 135
ApiExitCommon stanza 137
ApiExitTemplate stanza 137
DefaultQueueManager stanza 136
definition of 130
editing 130
ExitProperties stanza 136
path to 37
priorities 131
QueueManager stanza 137
MQSC (WebSphere MQ Script) commands
authorization 160
command files for the sample programs 37
input 35
output reports 36
running 35
samples for the WebSphere MQ SSL support 173
ending interactive input 35
escape PCFs 60
issuing commands interactively 33
to a remote queue manager 67
to a z/OS queue manager 68
maximum line length 36
object attribute names 6
overview 16
performing local administration 32
problems using MQSC commands remotely 68
problems, list 37
problems, resolving 37
redirecting input and output 35
runningmqsc control command, modes introduction 16
performing local administration 32
syntax errors 34
timed out command responses 67
MQSC (WebSphere MQ Script) commands (continued)
using 35
verifying 37
MQSC command files for the sample programs 37
input 35
output reports 36
running 35
samples for the WebSphere MQ SSL support 173
MQSET call
additional notes 451
support for 450
MQSIXD_* values 383
MQSNOAUTH environment variable 445
MQSYNC call 450
MQTM structure 447
MQTMIC2 structure 447
MQUHI structure 447
MQXACT_* values 388
MQXEP call 394
MQXPD_* values 391
MQXQH structure 447
MQXR_* values 387
MQXR2_* values 389
MQXUA_* values 390
MQZ_CHECK_AUTHORITY call 323
MQZ_COPY_ALL_AUTHORITY call 328
MQZ_DELETE_AUTHORITY call 331
MQZ_DELETE_NAME call 358
MQZ_ENUMERATE_AUTHORITY_DATA call 334
MQZ_GET_AUTHORITY call 337
MQZ_GET_EXPLICIT_AUTHORITY call 340
MQZ_INIT_AUTHORITY call 343
MQZ_INIT_NAME call 360
MQZ_INSERT_NAME call 363
MQZ_LOOKUP_NAME call 365
MQZ_REFRESH_CACHE call 346
MQZ_SET_AUTHORITY call 348
MQZ_TERM_AUTHORITY call 351
MQZ_TERM_NAME call 368
MQZAD structure 353
MQZAD_* values 353
MQZAET_* values 355
MQZAO_* authority constants 157
MQZAO_* values 354
MQZED structure 356
MQZED_* values 356
MQZEP call 321
MQZSE_* values 334
MsgId, performance considerations when using 231
MSGTYPE keyword, rules table 190
multiple points of failure 98

N
name service
entry points
delete name 358
initialize name service 360
insert name 363
name service (continued)
entry points (continued)
lookup name 365
terminate name service 368
interface 316
introduction 13
overview 315
name transformations 18
namelists
description of 10
namelist files 105
naming objects
introduction 5
rules 239
naming server processes 26
national language support
data conversion 72
EBCDIC NL character conversion to ASCII 135
error log files 231
native non-PIC applications and SRLs, building C++ applications 460
native PIC applications and DLLs
building C applications 458
building C++ applications 460
NextChainAreaPtr field
MQACH structure 381
NL character, EBCDIC conversion to ASCII 135
nobody principal 146
nobody user group 147
non-native applications
building 461
non-native TNS library, installing an exit in WebSphere MQ 476
non-PIC applications
native
building C applications 459
building C++ applications 460
running 458
non-PIC environment, user exits 472
non-PIC exit, installing 475
nonpersistent messages
availability 102
checkpointing, queue server option 94
configuring for data integrity 111
put or got within a unit of work 183
storage 89
synchronization logging 87
tuning 87
NonStop Tuxedo 104
NMPMSPEED channel attribute 87
NSI (name service interface) 315

O
OAM (Object Authority Manager)
authorization service component 311
generic profiles 151
introduction 12
OAM database 105
overview 13
refreshing after changing a user’s authorization 311
using the OAM to control access to objects 149
Object Authority Manager (OAM)
See OAM (Object Authority Manager)
object catalog 105
object name transformation 19
object touch files 105
ObjectName parameter
check authority call 323
copy all authority call 328
delete authority call 331
get authority call 337
get explicit authority call 340
set authority call 348
objects
access to 143
administration of 15
attributes of 6
automation of administration tasks 16
channels 10
local queues 9
managing objects for triggering 47
multiple queues 10
name transformation 19
namelists 10
naming objects
introduction 5
rules 239
object name transformation 19
process definitions 10
queue manager objects used by MQI calls 10
queue managers 9
remote administration 61
remote queue objects 71
remote queues 9
system and default objects
introduction 11
list of 427
types of 5
using MQSC commands to administer 16
ObjectType field
MQZAD structure 354
ObjectType parameter
check authority call 324
copy all authority call 328
delete authority call 331
get authority call 338
get explicit authority call 341
set authority call 349
OpenSSL 163
openssl command
introduction 164
where installed 165
openssl pcikcs12 command
importing digital certificates 170
use in the sample shell scripts 174
openssl req command
deciding how to specify the configuration file 167
generating public and private keys, and a request for a personal certificate 168
use in the sample shell scripts 173
openssl x509 command, use in the sample shell scripts 173
OpenTMF 104
Options parameter
initialize authorization service call 343
initialize name service call 360
terminate authorization service call 351
terminate name service call 368
output, standard 33

P
parameters, queue server tuning 446
partitioning a queue file or queue overflow file 92
pass phrase stash file, Stash.sh 170
PATHCOM (Pathway control program)
introduction as an administration interface 17
Pathway
configuring for availability 111
server class for queue server 89
using control commands 83
PATHWAY attributes, changing 126
Pathway control program (PATHCOM)
See PATHCOM (Pathway control program)
pBufferLength parameter
MQ_GET_EXIT call 403
MQ_PUT_EXIT call 409
MQ_PUTI_EXIT call 411
PCF (programmable command format) commands
administration tasks 16
authorization specification tables 157
automating administrative tasks using PCF 59
escape PCFs 60
MQAI, using to simplify use of 60
no response from a PCF command 228
object attribute names 6
object attributes 60
pCharAttrLength parameter
MQ_INQ_EXIT call 406
MQ_SET_EXIT call 413
pCompCode parameter
MQ_BACK_EXIT call 397
MQ_CLOSE_EXIT call 398
MQ_CONNEXIT call 400
MQ_DISC_EXIT call 402
MQ_INIT_EXIT call 405
MQ_INQ_EXIT call 406
MQ_OPEN_EXIT call 408
MQ_PUT_EXIT call 409
MQ_PUTI_EXIT call 411
MQ_SET_EXIT call 413
MQ_TERM_EXIT call 415
MQXEP call 395
PEM (Privacy Enhanced Mail) format 164
performance
application design, impact on 230
Correlld, effect of searching on 231
introduction 85
message length, effect of 230
MQPUT1 call 231
MsgId, effect of searching on 231
PMQFUNC elementary data type 322
point
floating 455
pOptions parameter
MQ_CLOSE_EXIT call 398
MQ_OPEN_EXIT call 408
ppBuffer parameter
MQ_GET_EXIT call 403
MQ_PUT_EXIT call 409
MQ_PUT1_EXIT call 411
ppCharAttrs parameter
MQ_INQ_EXIT call 406
MQ_SET_EXIT call 413
ppConnectOpts parameter 400
ppDataLength parameter
MQ_GET_EXIT call 403
ppGetMsgOpts parameter 403
pHconn parameter
MQ_CONNEXIT call 400
MQ_DISC_EXIT call 402
MQ_CONNEXIT call 400
MQ.DISC_EXIT call 402
ppHobj parameter
MQ_CONNEXIT call 398
MQ_OPEN_EXIT call 408
ppIntAttrs parameter
MQ_INQ_EXIT call 406
MQ_SET_EXIT call 413
ppMsgDesc parameter
MQ_CONNEXIT call 400
MQ.PUT_EXIT call 409
MQ.PUT1_EXIT call 411
ppObjDesc parameter
MQ_OPEN_EXIT call 408
MQ.PUT1_EXIT call 411
ppPutMsgOpts parameter
MQ.PUT_EXIT call 409
MQ.PUT1_EXIT call 411
ppSelectors parameter
MQ_INQ_EXIT call 406
MQ_SET_EXIT call 413
pQMgrName parameter
MQ_CONNEXIT call 400
pReason parameter
MQ_BACK_EXIT call 397
MQ_CLOSE_EXIT call 398
MQ.CMIT_EXIT call 399
MQ.PCMQ_EXIT call 400
MQ_CONNEXIT call 398
MQ.DISC_EXIT call 402
MQ_GET_EXIT call 403
MQ.INIT_EXIT call 405
MQ_INQ_EXIT call 406
MQ_OPEN_EXIT call 408
MQ.PUT_EXIT call 409
MQ.PUT1_EXIT call 411
MQ.SET_EXIT call 413
MQ.TERM_EXIT call 415
pFX format 164
pHconn parameter
MQ_BACK_EXIT call 397
MQ_CLOSE_EXIT call 398
MQ.CMIT_EXIT call 399
MQ_CONNEXIT call 400
MQ.DISC_EXIT call 402
MQ_GET_EXIT call 403
MQ.INIT_EXIT call 405
MQ_INQ_EXIT call 406
MQ_OPEN_EXIT call 408
MQ.PUT_EXIT call 409
MQ.PUT1_EXIT call 411
MQ_SET_EXIT call 413
MQ.TERM_EXIT call 415
pHobj parameter
MQ_GET_EXIT call 403
MQ.INQ_EXIT call 406
MQ.PUT_EXIT call 409
MQ.PUT1_EXIT call 411
MQ_SET_EXIT call 413
pIntAttrCount parameter
MQ_INQ_EXIT call 406
MQ_SET_EXIT call 413
PKCS#12 format 164
pQMgrName parameter
MQ_CONNEXIT call 400
pReason parameter
MQ_BACK_EXIT call 397
MQ_CLOSE_EXIT call 398
MQ.CMIT_EXIT call 399
MQ_CONNEXIT call 400
MQ.DISC_EXIT call 402
MQ_GET_EXIT call 403
MQ.INIT_EXIT call 405
MQ_INQ_EXIT call 406
MQ_OPEN_EXIT call 408
MQ.PUT_EXIT call 409
MQ.PUT1_EXIT call 411
MQ.SET_EXIT call 413
MQ.TERM_EXIT call 415
MQXEP call 396
preemptive queues 4
preemptive shutdown of a queue
manager 30
prevent access control checks 153
principal database
critical database files 105
role in access control 146
principals
nobody principal 146
role in access control 146
Privatity Enhanced Mail (PEM) format
See PEM (Privacy Enhanced Mail) format
problem determination
application design considerations 230
applications or systems running slowly 230
command errors 224
common programming errors 223
configuration files 233
error log files 231
has the application run successfully before? 222
incorrect output, definition of 225
incorrect output, distributed queuing 227
intermittent problems 224
introduction 221
PCF command, no response from 228
preliminary checks 221
problems affecting parts of a network 224
problems caused by service updates 225
problems that occur at specific times in the day 224
problems with shutdown 29
questions to ask 221
queue failures, problems caused by 229
remote queues, problems affecting 229
reproducing the problem 222
return codes checking all return codes 223
MQI calls 222
searching for messages, performance effects 231
things to check first 221
tracing
introduction 233
sample trace data 233
selecting components to trace 233
trace files 235
undelivered messages 232
WebSphere MQ error messages 222
what is different since the last successful run? 222
proc.ini (default process management rules configuration file) definition of 130
introduction 80
role in managing processes 197
process definitions commands for process administration 442
creating 48
description of 10
displaying 49
process management rules configuration file (qmproc.ini)
See qmproc.ini (process management rules configuration file)
ProcessId field
MQAXC structure 385
Performance (continued)
persistent messages, effect of 230
syncpoints, effect of 231
threads, effect of 231
performance events 433
PERSIST keyword, rules table 190
persistent messages availability 100
effect on performance 230
put or got outside of syncpoint control 183
storage 88
tuning 87
pExitContext parameter
MQ_BACK_EXIT call 397
MQ_CLOSE_EXIT call 398
MQ.CMIT_EXIT call 399
MQ_CONNEXIT call 400
MQ.DISC_EXIT call 402
MQ.GET_EXIT call 403
MQ.INIT_EXIT call 405
MQ_INQ_EXIT call 406
MQ_OPEN_EXIT call 408
MQ.PUT_EXIT call 409
MQ.PUT1_EXIT call 411
MQ.SET_EXIT call 413
MQ.TERM_EXIT call 415
pExitParms parameter
MQ_BACK_EXIT call 397
MQ_CLOSE_EXIT call 398
MQ.CMIT_EXIT call 399
MQ_CONNEXIT call 400
MQ.DISC_EXIT call 402
MQ.GET_EXIT call 403
MQ.INIT_EXIT call 405
MQ_INQ_EXIT call 406
MQ_OPEN_EXIT call 408
MQ.PUT_EXIT call 409
MQ.PUT1_EXIT call 411
MQ.SET_EXIT call 413
MQ.TERM_EXIT call 415
pFX format 164
pHconn parameter
MQ_BACK_EXIT call 397
MQ_CLOSE_EXIT call 398
MQ.CMIT_EXIT call 399
MQ_CONNEXIT call 400
MQ.DISC_EXIT call 402
MQ.GET_EXIT call 403
MQ.INIT_EXIT call 405
MQ_INQ_EXIT call 406
MQ_OPEN_EXIT call 408
MQ.PUT_EXIT call 409
MQ.PUT1_EXIT call 411
MQ.SET_EXIT call 413
MQ.TERM_EXIT call 415
pHobj parameter
MQ_GET_EXIT call 403
MQ_INQ_EXIT call 406
MQ_PUT_EXIT call 409
MQ_SET_EXIT call 413
PIC applications
native
building C applications 458
building C++ applications 460
running 458
PIC environment, user exits 471
pIntAttrCount parameter
MQ_INQ_EXIT call 406
MQ_SET_EXIT call 413
PKCS#12 format 164
primary points of interest 184
principal database
critical database files 105
role in access control 146
principals
nobody principal 146
role in access control 146
privacy enhanced mail (PEM) format
see PEM (privacy enhanced mail) format
Problem determination
application design considerations 230
applications or systems running slowly 230
command errors 224
common programming errors 223
configuration files 233
error log files 231
has the application run successfully before? 222
incorrect output, definition of 225
incorrect output, distributed queuing 227
intermittent problems 224
introduction 221
PCF command, no response from 228
preliminary checks 221
problems affecting parts of a network 224
problems caused by service updates 225
problems that occur at specific times in the day 224
problems with shutdown 29
questions to ask 221
queue failures, problems caused by 229
remote queues, problems affecting 229
reproducing the problem 222
return codes checking all return codes 223
MQI calls 222
searching for messages, performance effects 231
things to check first 221
tracing
introduction 233
sample trace data 233
selecting components to trace 233
trace files 235
undelivered messages 232
WebSphere MQ error messages 222
what is different since the last successful run? 222
proc.ini (default process management rules configuration file) definition of 130
introduction 80
role in managing processes 197
process definitions commands for process administration 442
creating 48
description of 10
displaying 49
process management rules configuration file (qmproc.ini)
See qmproc.ini (process management rules configuration file)
ProcessId field
MQAXC structure 385
product code 84
product files 84
locating 456
ProfileName field
MQZAD structure 354
profiles, generic 151
programmable command format (PCF) commands
See PCF (programmable command format) commands
programming errors
common errors 223
further checks 225
programs, samples supplied 463
pSelectorCount parameter
MQ_INQ_EXIT call 406
MQ_SET_EXIT call 413
PUTAUT keyword, rules table 192

Q
qm.ini (queue manager configuration file)
ApExitLocal stanza 141
Channels stanza 139
definition of 130
editing 130
ExitPath stanza 141
priorities 131
 RestrictedMode stanza 138
Service stanza 138
ServiceComponent stanza 138
TCP stanza 140
QMgrName field
MQAXP structure 391
MQLXP structure 473
QMgrName parameter
check authority call 323
copy all authority call 328
delete authority call 331
enumerate authority data call 334
get authority call 337
get explicit authority call 340
initialize authorization service call 343
initialize name service call 360
insert name call 363
lookup name call 365
MQZ_DELETE_NAME call 358
set authority call 348
terminate authorization service call 351
terminate name service call 368
qmproc.ini (process management rules configuration file)
definition of 130
editing 130
introduction 17
keyword definitions 203
role in managing processes 197
stanzas names 203
use by the execution controller 80
QName parameter
insert name call 363
lookup name call 365
MQZ_DELETE_NAME call 358
queue browser, sample 43
queue files
critical database files 105
introduction 88
moving 90
partitioning 92
queue manager clusters
class transmission queues
introduction 8
performance 90
class workload exit 12
ExitProperties stanza, mq.ini 136
introduction 10
network availability 110
using for remote administration 62
queue manager configuration file (qm.ini)
See qm.ini (queue manager configuration file)
queue manager events 433
Queue Manager Menu, Monitoring Panels 50
queue manager server
failure recovery 105
function 83
queue manager's directory 26
queue manager's SSL files
changing the location of 171
creating 171
location 171
preparing 170
when changes become effective 172
queue manager's subvolume 26
queue managers
accidentally making another queue manager the default 253
attributes
changing 40
displaying using DISPLAY
QMGX 39
displaying using the Monitoring Panels 50
information specific to WebSphere
MQ for HP NonStop Server 452
availability 99
backing up 214
changing
CCSID of a queue manager 73
default queue manager 27
command server 66
commands for queue manager administration 441
configuration files, backing up 28
configuration information 115
creating 24
creating a default queue manager 27
default for an installation 25
deleting
manually 439
using dtmqm 30
description of 9
extending queue manager facilities 12
failure recovery 105
guidelines for creating 24
home terminal 26
limiting the numbers of 25
monitoring 433
name transformation 18
queue managers (continued)
objects used in MQI calls 10
preparing for remote administration 63
queue manager aliases 71
remote administration 61
recovering 214
reverting to the original default 27
server classes 116
server processes 77
specifying unique names for 24
starting 29
stopping
manually 439
using endmqm 29
Queue Menu, Monitoring Panels 51
queue overflow files
critical database files 105
introduction 88
moving 90
partitioning 92
queue server
CPU distribution 89
failure recovery 105
function 81
maintaining a Measure counter for a queue 93
managing nonpersistent data 102
options
audited message overflow files 94
checkpoint nonpersistent messages 94
load at startup 94
lock in memory 94
reassigning a WebSphere MQ object 89
recovery and restart 215
storing messages in memory 93
 tuning 88
QueueManager stanza, mq.ini 137
queues
alias 44
application queues 47
attributes
displaying using DISPLAY
QUEUE 41
displaying using the Monitoring Panels 53
examples 7
information specific to WebSphere
MQ for HP NonStop Server 452
browsing 43
clearing local queues 42
cluster transmission queue 90
commands for queue administration 441
current queue depth, determining 41
dead letter, defining 41
defaults, transmission queues 26
defining queues 7
deletion of 4
deleting a local queue 43
distributed, incorrect output from 227
dynamic queues 4
event queues
introduction 9

Index 499
Remote administration

administering a remote queue manager from a local one 63
command server 66
defining channels and transmission queues 64
definition of remote administration 15
initial problems 68
of objects 61
preparing channels for 64
preparing queue managers for 63
preparing transmission queues for 64
Remote Database Facility (RDF)
See RDF (Remote Database Facility)
remote queue objects 71
remote queues
as reply-to queue aliases 71
creating a local definition of a remote queue 68
recommendations for remote queuing 68
remote queuing 63
removing
a queue manager
manually 439
using dltrimq 30
a server class 126
reply-to queue aliases 71
reply-to queues
description of 9
reply-to queue aliases 71
REPLYQ keyword, rules table 191
REPLYQM keyword, rules table 191
repository manager
attributes 203
failure recovery 105
managed by execution controller 197
Reserved parameter
MQXEP call 395
ResolvedQMgrName parameter
insert name call 363
lookup name call 365
resource manager 179
restoring a queue manager 214
RestrictedMode stanza, qm.ini 138
restrictions
access to WebSphere MQ objects 143
on object names 239
retrieval algorithms for messages 5
RETRY keyword, rules table 192
RETRYINT keyword, rules tables 189
return codes, problem determination 222
roll back a unit of work 179
rules
application 201
channel 202
rules table (DLQ handler) (continued)
ACTION keyword 191
action keywords 191
APPLIDAT keyword 190
APPLNAME keyword 190
APPLTYPE keyword 190
definition of 189
control-data entry 188
corelating 182
description of 188
rules table (DLQ handler) (continued)
DESTQ keyword 190
DESTQM keyword 190
event of 195
example of a rules table 195
FEEDBACK keyword 190
FORMAT keyword 190
FWDQ keyword 191
FWDOM keyword 191
HEADER keyword 192
INPUTQ keyword 188
INPUTQM keyword 189
MSGTYP0 keyword 190
patterns and actions 189
PERSIST keyword 190
processing rules 194
PUTAUT keyword 192
REASON keyword 190
REPLYQ keyword 191
REPLYQM keyword 191
RETRY keyword 192
RETRYINT keyword 189
syntax rules 193
USERID keyword 191
WAIT keyword 189
run channel command (runmqch1)
See runmqch1 (run channel command)
run channel initiator command
(runmqchi)
See runmqchi (run channel initiator) command
run dead letter queue handler command
(runmqdlq)
See runmqdlq (run dead letter queue handler) command
run listener command (runmqlsr)
See runmqlsr (run listener) command
run MQSC commands command
(runmqsc)
See runmqsc (run MQSC commands) command
runmqchi (run channel initiator)
definition 276
runmqch1 (run channel) command
definition 277
runmqdlq (run dead letter queue handler) command
definition 278
runmqlsr (run listener) command
definition 279
runmqsc (run MQSC commands) command
definition 281
definition of 35
feedback 34
fix command features 34
indirect mode 67
problems, resolving 37
redirecting input and output 35
using 35
using interactively 33
verifying 37
runmqtrm (start trigger monitor)
call 322
command
definition 284
running the entropy daemon 165
start command server command
  (strmqcsv)
    See strmqcsv (start command server) command
start queue manager command (strmqm)
  See strmqm (start queue manager) command
start trace command (strmqtrc)
  See strmqtrc (start trace) command
start trigger monitor command (runmqtrm)
  See runmqtrm (start trigger monitor) command
StartEnumeration parameter
e numerate authority data call 334

starting
  a channel 65
  a channel initiator 484
  a listener
    for remote administration 65
    for TCP/IP channels 480
  a queue manager 29
  the command server 66
  the entropy daemon 165
Strash, sth file 170
stdin, on runmqsc 35
stdout, on runmqsc 35

stopping
  a queue manager
    manually 439
    using endmqm 29
  listeners 481
  the command server 67
  the entropy daemon 166
strmqcsv (start command server) command
definition 292
strmqm (start queue manager) command
definition 293
strmqtrc (start trace) command
definition 294

StrucLength field
  MQACH structure 379
  MQAXC structure 382
  MQAXP structure 386
  MQLXP structure 472
  MQZAD structure 353
  MQZED structure 356

StrucId field
  MQACH structure 379
  MQAXC structure 382
  MQAXP structure 386
  MQI 447
  MQLXP 472
  MQZAD structure 353
  MQZED structure 356

Subsystem Control Facility (SCF)
  See SCF (Subsystem Control Facility)
subvolume of a queue manager 26
syncpoint
cordinating changes to external
databases 103
definition 179
limits 182
performance considerations 231
syncpoint coordinator 179
syntax of control commands, help
with 242
system and default objects
  introduction 11
  list of 427

T

TCP stanza, qm.ini 140
TCP/IP
  channels
    configuring the calling end 479
    configuring the responding end 480
    configuring, introduction 479
    keep alive function 482
    specifying the TCP/IP process 481
    TCP/IP listener
    failure recovery 105
    function 83
    templates, EMS event 435
Thread field
  MQAXC structure 385
timed out responses from MQSC commands 67

TMF
  audit files 87
  TMF (Transaction Management Facility)
    audit files 87
    audit trail size 185
    auditing database files 102
    configuring for WebSphere MQ 184
    coordinating changes to external databases 103
    coordinating global units of work, introduction 14
    coordinating local and global units of work 180
    monitoring the status of 185
    role as a transaction manager 180
    transactions
      implementing global units of work 181
      long running 182
      managing multiple 181
      maximum number for an application 183
      troubleshooting 185
    TMFCOM (TMF configuration utility) 18
    TNS-R
      building C applications 459
      building C++ applications 460
    touch files for queues
      critical database files 105
      introduction 88
      moving 90
    tracing
      introduction 233
      sample trace data 233
      selecting components to trace 233
      trace files 235
    Transaction Management Facility (TMF)
      See TMF (Transaction Management Facility)
    transaction manager 179

transactional support
deals 179
overview 13
transactions, TMF
  implementing global units of work 181
  long running 182
  managing multiple 181
  maximum number for an application 183
transmission queues
  cluster transmission queues
    introduction 8
    performance 90
    creating 70
    default 26
    default transmission queues 71
    defining transmission queues remote administration 64
    description of 8
    preparing transmission queues for remote administration 64
    security 155

triggering
  defining an application queue for triggering 47
  managing objects for triggering 47
  message driven processing 3
  trigger monitor 83
  triggered applications 455
  troubleshooting, TMF 185
  tuning parameters, queue server 446
  tuning WebSphere MQ for HP NonStop Server 85
  Tuxedo, NonStop 104

U

undelivered message queue
  See dead letter queues
units of work
  back out 179
  commit 179
  global
    coordinating with TMF 181
    definition of 180
    introduction 179
  local
    coordinating with TMF 182
    definition of 180
  updating coded character sets 72
  user defined message formats 73
  user exits
    API exits 471
    channel exits 155
    cluster workload exit 12
    data conversion exit 73
    loading 472
    non-PIC environment 472
    PIC environment 471
    Shared Resource Library (SRL) applications 471
types of user exit 12
  user groups
    See groups
  UserId field
    MQAXC structure 383
V
variables, environment  445
verifying MQSC commands  37
Version field
   MQACH structure  380
   MQAXC structure  382
   MQAXP structure  386
   MQLXP structure  473
   MQZAD structure  353
   MQZED structure  356
Version parameter
   initialize authorization service
call            344
   initialize name service call  361

W
WAIT keyword, rules table  189
WebSphere MQ Administrative Interface
   (MQAI)       60
WebSphere MQ clients  11
WebSphere MQ configuration file
   (mqs.ini)
      See mqs.ini (WebSphere MQ configuration file)
WebSphere MQ Script (MQSC) commands
      See MQSC (WebSphere MQ Script) commands
WebSphere MQ SRLs
   installing a non-PIC exit  475
workload balancing  85

X
XA interface  455

Z
z/OS queue manager, issuing MQSC commands to  68
ZMQSTMPL, EMS event template file  435
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.