

WebSphere MQ



Intercommunication

Version 7.0

WebSphere MQ



Intercommunication

Version 7.0

Note

Before using this information and the product it supports, be sure to read the general information under notices at the back of this book.

First edition (April 2008)

This edition of the book applies to the following products:

- IBM WebSphere MQ, Version 7.0
- IBM WebSphere MQ for z/OS, Version 7.0

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

© Copyright International Business Machines Corporation 1994, 2008. All rights reserved.

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

Contents

Figures **vii**

Tables **ix**

Chapter 1. Introduction **1**

Concepts of intercommunication.	1
What is intercommunication?	1
Distributed queuing components	5
Dead-letter queues	12
Remote queue definitions.	12
How to get to the remote queue manager	13
Security	15
Making your applications communicate	15
How to send a message to another queue manager	15
Triggering channels.	18
Safety of messages	20
More about intercommunication	21
Addressing information	21
What are aliases?	21
Queue manager alias definitions	22
Reply-to queue alias definitions	24
Networks	25

Chapter 2. How intercommunication works **29**

WebSphere MQ distributed-messaging techniques	29
Message flow control	29
Putting messages on remote queues	31
Choosing the transmission queue	33
Receiving messages.	33
Passing messages through your system	35
Separating message flows	36
Concentrating messages to diverse locations	38
Diverting message flows to another destination	39
Sending messages to a distribution list	40
Reply-to queue	41
Networking considerations	46
Return routing	47
Managing queue name translations	47
Channel message sequence numbering	49
Loopback testing	49
Route tracing and activity recording	50
DQM implementation	50
Functions of DQM	50
Message sending and receiving.	51
Channel control function	53
What happens when a message cannot be delivered?	66
Initialization and configuration files	68
Data conversion	70
Writing your own message channel agents	70
Channel attributes	71
Channel attributes and channel types.	71
Channel attributes in alphabetical order	73

Example configuration chapters in this book	101
Network infrastructure	102
Communications software	102
How to use the communication examples	103

Chapter 3. DQM in WebSphere MQ for Windows and Unix platforms **105**

Monitoring and controlling channels on Windows and Unix platforms	105
DQM channel control.	105
Functions available	106
Getting started with objects.	107
Channel attributes and channel types	111
Channel functions	111
Preparing WebSphere MQ for distributed platforms	115
Transmission queues and triggering	115
Channel programs.	116
Other things to consider.	117
What next?	120
Setting up communication for Windows	120
Deciding on a connection	120
Defining a TCP connection	121
Defining an LU 6.2 connection	122
Defining a NetBIOS connection	124
Defining an SPX connection	126
Example configuration - IBM WebSphere MQ for Windows	129
Configuration parameters for an LU 6.2 connection	129
Establishing an LU 6.2 connection	134
Establishing a TCP connection.	142
Establishing a NetBIOS connection	142
Establishing an SPX connection	143
WebSphere MQ for Windows configuration	145
Setting up communication on UNIX systems	150
Deciding on a connection	150
Defining a TCP connection	151
Defining an LU 6.2 connection	154
Example configuration - IBM WebSphere MQ for AIX	155
Configuration parameters for an LU 6.2 connection	156
Establishing a session using Communications Server for AIX	160
Establishing a TCP connection.	166
WebSphere MQ for AIX configuration	167
Example configuration - IBM WebSphere MQ for HP-UX	172
Configuration parameters for an LU 6.2 connection	172
Establishing a session using HP SNAplus2	176
Establishing a TCP connection.	189
WebSphere MQ for HP-UX configuration	190
Example configuration - IBM WebSphere MQ for Solaris.	194

Configuration parameters for an LU 6.2 connection using SNAP-IX	195
Establishing a session using SNAP-IX	199
Establishing a TCP connection.	210
WebSphere MQ for Solaris configuration	211
Example configuration - IBM WebSphere MQ for Linux	215
Configuration parameters for an LU 6.2 connection	215
Establishing a session using Communications Server for Linux	219
Establishing a TCP connection.	232
WebSphere MQ for Linux configuration	234
Message channel planning example for distributed platforms.	238
What the example shows	238
Running the example.	241

Chapter 4. DQM in WebSphere MQ for z/OS 243

Monitoring and controlling channels on z/OS	243
The DQM channel control function	243
Using the panels and the commands	244
Managing your channels	245
Preparing WebSphere MQ for z/OS	259
Defining DQM requirements to WebSphere MQ	259
Defining WebSphere MQ objects	260
Other things to consider.	261
z/OS Automatic Restart Management (ARM)	262
Setting up communication for z/OS.	263
Deciding on a connection	263
Defining a TCP connection	263
Defining an LU6.2 connection	265
Example configuration - IBM WebSphere MQ for z/OS	266
Configuration parameters for an LU 6.2 connection	267
Establishing an LU 6.2 connection	270
Establishing a TCP connection.	272
WebSphere MQ for z/OS configuration.	272
Message channel planning example for z/OS.	275
What the example shows	276
Running the example.	279
Preparing WebSphere MQ for z/OS for DQM with queue-sharing groups	280
Concepts	280
Components.	280
Benefits	283
Clusters and queue-sharing groups	284
Channels and serialization	284
Intra-group queuing	284
Setting up communication for WebSphere MQ for z/OS using queue-sharing groups	284
Deciding on a connection	285
Defining a TCP connection	285
Defining an LU6.2 connection	285
Example configuration - IBM WebSphere MQ for z/OS using queue-sharing groups	286
Configuration parameters for an LU 6.2 connection	286

Establishing an LU 6.2 connection into a queue-sharing group	288
Establishing a TCP connection into a queue-sharing group	290
WebSphere MQ for z/OS shared channel configuration	291
Message channel planning example for z/OS using queue-sharing groups	294
What this example shows	294
Intra-group queuing	298
Concepts	298
Terminology.	300
Benefits	300
Limitations	302
Getting started	302
Configurations	303
Messages.	308
Security	310
Specific properties.	310
Example configuration — WebSphere MQ for z/OS using intra-group queuing	312
Configuration 1	313
Configuration 2	314
Configuration 3	314
Running the example.	318

Chapter 5. DQM in WebSphere MQ for i5/OS 321

Monitoring and controlling channels on i5/OS	321
DQM channel control.	321
Operator commands	321
Getting started	323
Creating objects	324
Creating a channel	324
Starting a channel	326
Selecting a channel	327
Browsing a channel	327
Renaming a channel	329
Work with channel status	329
Work-with-channel choices	331
Panel choices	332
Preparing WebSphere MQ for i5/OS.	336
Creating a transmission queue.	337
Triggering channels in WebSphere MQ for i5/OS	339
Channel programs.	339
Channel states on i5/OS.	340
Other things to consider.	340
Setting up communication for WebSphere MQ for i5/OS	342
Deciding on a connection	342
Defining a TCP connection	342
Defining an LU 6.2 connection	344
Example configuration - IBM WebSphere MQ for i5/OS	350
Configuration parameters for an LU 6.2 connection	351
Establishing an LU 6.2 connection	355
Establishing a TCP connection.	360
WebSphere MQ for i5/OS configuration	362

Message channel planning example for WebSphere	
MQ for i5/OS	368
What the example shows	368
Running the example.	372

Chapter 6. Further intercommunication considerations. . . 375

Channel-exit programs	375
What are channel-exit programs?.	375
Writing and compiling channel-exit programs	394
SSPI security exit	403
Implications of sharing conversations	404
Channel-exit calls and data structures	405
Data definition files	406
MQ_CHANNEL_EXIT – Channel exit	406
MQ_CHANNEL_AUTO_DEF_EXIT – Channel auto-definition exit	410
MQXWAIT – Wait in exit	412
MQCD – Channel definition	413
MQCXP – Channel exit parameter	458
MQXWD – Exit wait descriptor	475
Problem determination in DQM	476
Error message from channel control	477
Ping	477
Dead-letter queue considerations	477
Validation checks	478

In-doubt relationship	478
Channel startup negotiation errors	478
When a channel refuses to run	479
Retrying the link	481
Data structures	482
User exit problems	482
Disaster recovery	482
Channel switching.	483
Connection switching.	483
Client problems	483
Error logs	484
Message monitoring	485

Chapter 7. Queue name resolution 487

What is queue name resolution?	488
How queue name resolution works	489

Chapter 8. Configuration file stanzas for distributed queuing 491

Notices 493

Index 497

Sending your comments to IBM . . . 509

Figures

1. Overview of the components of distributed queuing	2	44. Listing cluster channels	259
2. Sending messages	3	45. Channel Initiator APPL definition	271
3. Sending messages in both directions	4	46. The first example for WebSphere MQ for z/OS	276
4. A cluster of queue managers	5	47. Message channel planning example for WebSphere MQ for z/OS using queue-sharing groups	295
5. A sender-receiver channel	7	48. An example of intra-group queuing	299
6. A requester-server channel	7	49. An example of migration support	301
7. A requester-sender channel	8	50. An example configuration	304
8. A cluster-sender channel	8	51. An example of clustering with intra-group queuing	306
9. Channel initiators and listeners	10	52. Configuration 2	312
10. Sequence in which channel exit programs are called	12	53. Configuration 1: z/OS using intra-group queuing	313
11. Passing through intermediate queue managers	13	54. Configuration 3	314
12. Sharing a transmission queue	14	55. Create channel (1)	325
13. Using multiple channels	14	56. Create channel (2)	325
14. The concepts of triggering	19	57. Create channel (3)	326
15. Queue manager alias	23	58. Create channel (4)	326
16. Reply-to queue alias used for changing reply location	24	59. Work with channels	327
17. Network diagram showing all channels	26	60. Display a TCP/IP channel (1)	328
18. Network diagram showing QM-concentrators	28	61. Display a TCP/IP channel (2)	328
19. A remote queue definition is used to resolve a queue name to a transmission queue to an adjacent queue manager	32	62. Display a TCP/IP channel (3)	329
20. The remote queue definition allows a different transmission queue to be used	33	63. Channel status (1)	330
21. Receiving messages directly, and resolving alias queue manager name	34	64. Channel status (2)	330
22. Three methods of passing messages through your system	35	65. Channel status (3)	331
23. Separating messages flows	37	66. Create a queue (1)	337
24. Combining message flows on to a channel	38	67. Create a queue (2)	338
25. Diverting message streams to another destination	39	68. Create a queue (3)	338
26. Reply-to queue name substitution during PUT call	41	69. Create a queue (4)	339
27. Reply-to queue alias example	43	70. LU 6.2 communication setup panel - initiating end	345
28. Distributed queue management model	52	71. LU 6.2 communication setup panel - initiated end	348
29. Channel states and substates	56	72. LU 6.2 communication setup panel - initiated end	349
30. Flows between channel states	57	73. The message channel example for WebSphere MQ for i5/OS	369
31. What happens when a message cannot be delivered	67	74. Security exit loop	376
32. WebSphere MQ channel to be set up in the example configuration chapters in this book	101	75. Example of a send exit at the sender end of message channel	377
33. Local LU window	164	76. Example of a receive exit at the receiver end of message channel	377
34. Mode window	165	77. Sender-initiated exchange with agreement	379
35. The message channel example for Windows, and UNIX systems	239	78. Sender-initiated exchange with no agreement	380
36. The operations and controls initial panel	244	79. Receiver-initiated exchange with agreement	381
37. Listing channels	245	80. Receiver-initiated exchange with no agreement	382
38. Starting a system function	250	81. Client connection-initiated exchange with agreement for client connection using security parameters	383
39. Stopping a function control	251	82. Security exit skeleton code	384
40. Starting a channel	253	83. Sample source code for a channel exit on Windows	398
41. Testing a channel	254	84. Sample DEF file for Windows	399
42. Stopping a channel	256		
43. Listing channel connections	258		

85.	Sample source code for a channel exit on AIX	400	92.	Sample source code for a channel exit on Linux	403
86.	Sample compiler and linker commands for channel exits on AIX	400	93.	Sample compiler and linker commands for channel-exits on Linux platforms where the queue manager is 64-bit	403
87.	Sample export file for AIX	400	94.	Sample compiler and linker commands for channel-exits on Linux platforms where the queue manager is 32-bit	403
88.	Sample source code for a channel exit on HP-UX	401	95.	Name resolution	487
89.	Sample compiler and linker commands for channel exits on HP-UX	401	96.	qm.ini stanzas for distributed queuing	491
90.	Sample source code for a channel exit on Solaris	402			
91.	Sample compiler and linker commands for channel exits on Solaris	402			

Tables

1. Example of channel names	26	19. Configuration worksheet for HP SNAplus2	172
2. Three ways of using the remote queue definition object	31	20. Configuration worksheet for WebSphere MQ for HP-UX	191
3. Reply-to queue alias	45	21. Configuration worksheet for SNAP-IX	195
4. Queue name resolution at queue manager QMA	48	22. Configuration worksheet for WebSphere MQ for Solaris.	212
5. Queue name resolution at queue manager QMB.	48	23. Configuration worksheet for Communications Server for Linux	216
6. Reply-to queue name translation at queue manager QMA	48	24. Configuration worksheet for WebSphere MQ for Linux	235
7. Channel attributes for the channel types	71	25. Channel tasks	245
8. Negotiated HBINT value and the corresponding KAINTE value	85	26. Settings on the local z/OS system for a remote queue manager platform	265
9. Functions available in Windows systems and UNIX systems	106	27. Configuration worksheet for z/OS using LU 6.2	267
10. Channel programs for Windows and UNIX systems	117	28. Configuration worksheet for WebSphere MQ for z/OS	272
11. Settings on the local Windows system for a remote queue manager platform	122	29. Configuration worksheet for z/OS using LU 6.2	287
12. Default outstanding connection requests on Windows	128	30. Configuration worksheet for WebSphere MQ for z/OS using queue-sharing groups	292
13. Configuration worksheet for IBM Communications Server for Windows systems	130	31. Channel attribute fields per message channel type.	332
14. Configuration worksheet for WebSphere MQ for Windows.	146	32. Program and transaction names	340
15. Maximum outstanding connection requests queued at a TCP/IP port	152	33. Channel states on i5/OS	340
16. Settings on the local UNIX system for a remote queue manager platform	154	34. Settings on the local i5/OS system for a remote queue manager platform	344
17. Configuration worksheet for Communications Server for AIX	156	35. Configuration worksheet for SNA on an i5/OS system	351
18. Configuration worksheet for WebSphere MQ for AIX	168	36. Configuration worksheet for WebSphere MQ for i5/OS	364
		37. Channel exits available for each channel type	375
		38. Identifying API calls	388

Chapter 1. Introduction

Concepts of intercommunication

This chapter introduces the concepts of intercommunication in WebSphere® MQ.

- The basic concepts of intercommunication are explained in “What is intercommunication?”
- The objects that you need for intercommunication are described in “Distributed queuing components” on page 5.

This chapter goes on to introduce:

- “Dead-letter queues” on page 12
- “Remote queue definitions” on page 12
- “How to get to the remote queue manager” on page 13

What is intercommunication?

In WebSphere MQ, intercommunication means sending messages from one queue manager to another. The receiving queue manager could be on the same machine or another; nearby or on the other side of the world. It could be running on the same platform as the local queue manager, or could be on any of the platforms supported by WebSphere MQ. This is called a *distributed* environment. WebSphere MQ handles communication in a distributed environment such as this using Distributed Queue Management (DQM).

The local queue manager is sometimes called the *source queue manager* and the remote queue manager is sometimes called the *target queue manager* or the *partner queue manager*.

How does distributed queuing work?

Figure 1 on page 2 shows an overview of the components of distributed queuing.

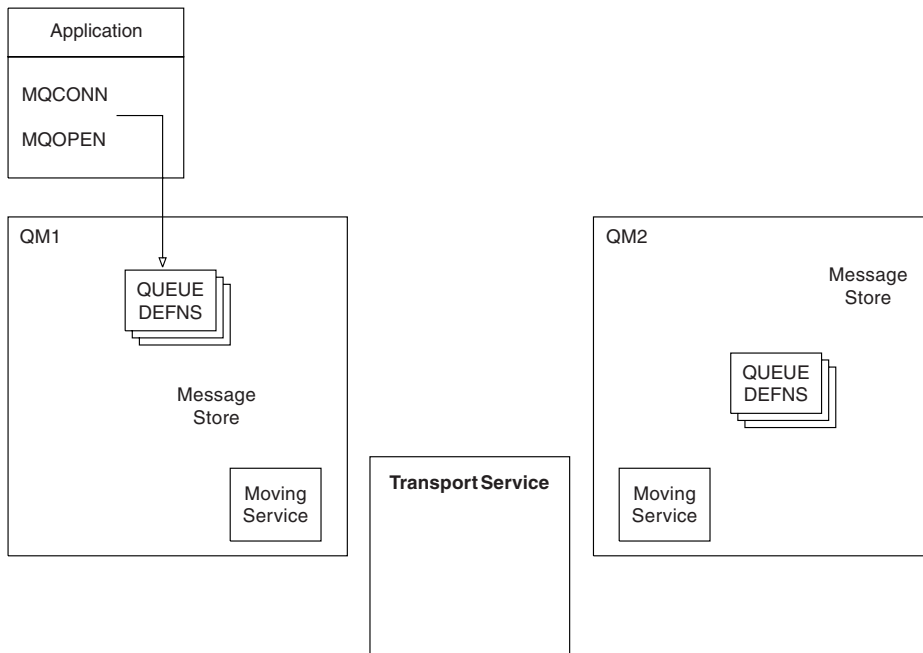


Figure 1. Overview of the components of distributed queuing

1. An application uses the MQCONN call to connect to a queue manager.
2. The application then uses the MQOPEN call to open a queue so that it can put messages on it.
3. A queue manager has a definition for each of its queues, specifying information such as the maximum number of messages allowed on the queue.
4. If the messages are destined for a queue on a remote system, the local queue manager holds them in a message store until it is ready to forward them to the remote queue manager. This can be transparent to the application.
5. Each queue manager contains communications software called the *moving service* component; through this, the queue manager can communicate with other queue managers.
6. The *transport service* is independent of the queue manager and can be any one of the following (depending on the platform):
 - Systems Network Architecture Advanced Program-to Program Communication (SNA APPC)
 - Transmission Control Protocol/Internet Protocol (TCP/IP)
 - Network Basic Input/Output System (NetBIOS)
 - Sequenced Packet Exchange (SPX)

What do we call the components?:

1. WebSphere MQ applications can put messages onto a local queue, that is, a queue on the queue manager the application is connected to.
2. A queue manager has a definition for each of its queues. It can also have definitions for queues that are owned by other queue managers. These are called *remote queue definitions*. WebSphere MQ applications can also put messages targeted at these remote queues.
3. If the messages are destined for a remote queue manager, the local queue manager stores them on a *transmission queue* until it is ready to send them to the remote queue manager. A transmission queue is a special type of local

queue on which messages are stored until they can be successfully transmitted and stored at the remote queue manager.

4. The software that handles the sending and receiving of messages is called the *Message Channel Agent (MCA)*.
5. Messages are transmitted between queue managers on a *channel*. A channel is a one-way communication link between two queue managers. It can carry messages destined for any number of queues at the remote queue manager.

Components needed to send a message:

If a message is to be sent to a remote queue manager, the local queue manager needs definitions for a transmission queue and a channel.

Each end of a channel has a separate definition, defining it, for example, as the sending end or the receiving end. A simple channel consists of a *sender* channel definition at the local queue manager and a *receiver* channel definition at the remote queue manager. These two definitions must have the same name, and together constitute one channel.

There is also a *message channel agent (MCA)* at each end of a channel.

Each queue manager should have a *dead-letter queue* (also known as the *undelivered message queue*). Messages are put on this queue if they cannot be delivered to their destination.

Figure 2 shows the relationship between queue managers, transmission queues, channels, and MCAs.

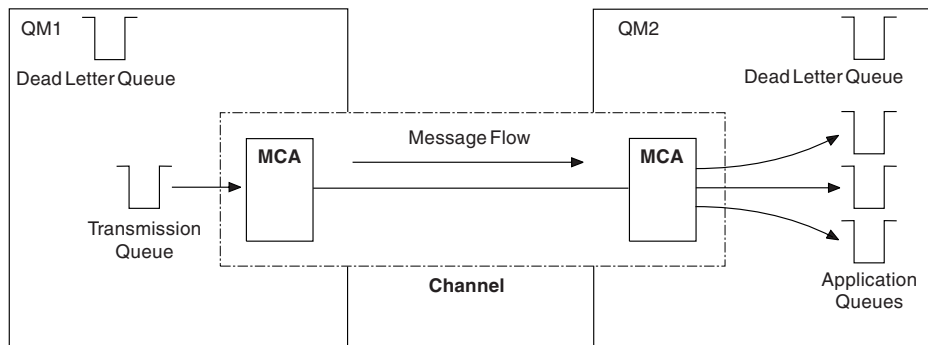


Figure 2. Sending messages

Components needed to return a message:

If your application requires messages to be returned from the remote queue manager, you need to define another channel, to run in the opposite direction between the queue managers, as shown in Figure 3 on page 4.

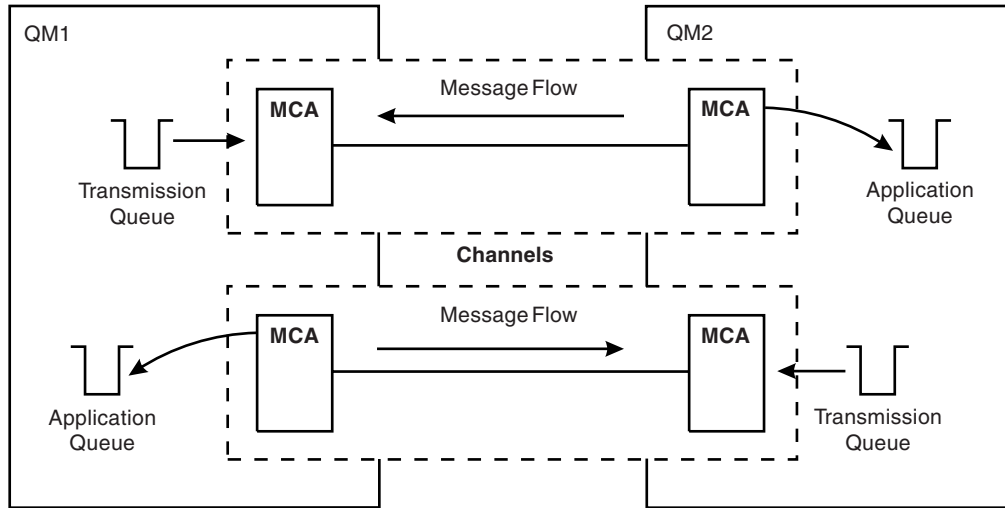


Figure 3. Sending messages in both directions

Cluster components:

An alternative to the traditional WebSphere MQ network is the use of clusters.

A cluster is a network of queue managers that are logically associated in some way. You can group queue managers in a cluster so that queue managers can make the queues that they host available to every other queue manager in the cluster. Assuming you have the necessary network infrastructure in place, any queue manager can send a message to any other queue manager in the same cluster without the need for explicit channel definitions, remote-queue definitions, or transmission queues for each destination. Every queue manager in a cluster has a single transmission queue that transmits messages to any other queue manager in the cluster. Each queue manager needs to define only one cluster-receiver channel and one cluster-sender channel.

Figure 4 on page 5 shows the components of a cluster called CLUSTER:

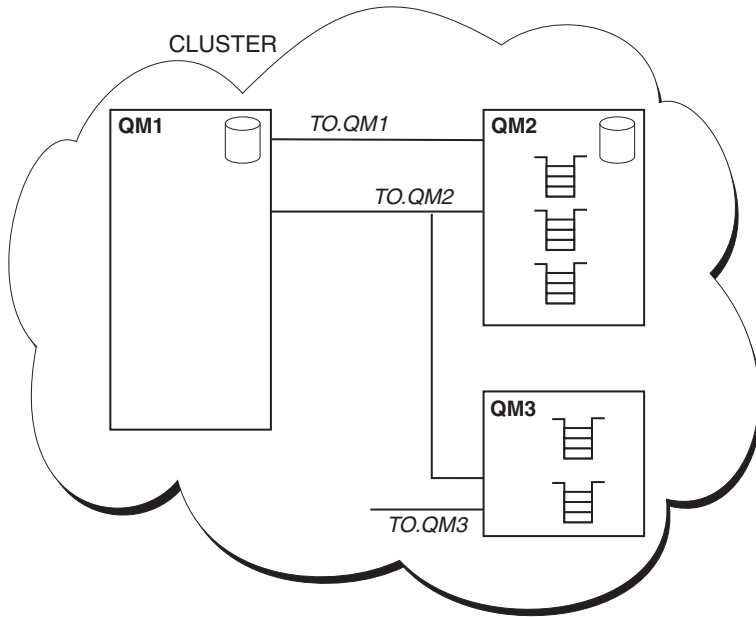


Figure 4. A cluster of queue managers

- CLUSTER contains three queue managers, QM1, QM2, and QM3.
- QM1 and QM2 host full repositories of information about the queue managers and queues in the cluster.
- QM2 and QM3 host some cluster queues, that is, queues that are accessible to any other queue manager in the cluster.
- Each queue manager has a cluster-receiver channel called TO.qmgr on which it can receive messages.
- Each queue manager also has a cluster-sender channel on which it can send information to one of the repository queue managers.
- QM1 and QM3 send to the repository at QM2 and QM2 sends to the repository at QM1.

As with distributed queuing, you use the MQPUT call to put a message to a queue at any queue manager. You use the MQGET call to retrieve messages from a local queue.

For further information about clusters, see the WebSphere MQ Queue Manager Clusters book.

Distributed queuing components

This section describes the components of distributed queuing. These are:

- Message channels
- Message channel agents
- Transmission queues
- Channel initiators and listeners
- Channel-exit programs

Message channels

Message channels are the channels that carry messages from one queue manager to another.

Do not confuse message channels with MQI channels. There are two types of MQI channel, server-connection and client-connection. These are discussed in WebSphere MQ Clients.

The definition of each end of a message channel can be one of the following types:

- Sender
- Receiver
- Server
- Requester
- Cluster sender
- Cluster receiver

A message channel is defined using one of these types defined at one end, and a compatible type at the other end. Possible combinations are:

- Sender-receiver
- Requester-server
- Requester-sender (callback)
- Server-receiver
- Cluster sender-cluster receiver

Detailed instructions for creating a sender-receiver channel are included in the *Quick Beginnings* book for your platform (not applicable to z/OS®). Examples of the parameters needed to set up sender-receiver channels can be found in this book, in the Example Configuration chapter applicable to your platform. For the parameters needed to define a channel of any type, see the **DEFINE CHANNEL** command in WebSphere MQ Script (MQSC) Command Reference.

Sender-receiver channels:

A sender in one system starts the channel so that it can send messages to the other system. The sender requests the receiver at the other end of the channel to start. The sender sends messages from its transmission queue to the receiver. The receiver puts the messages on the destination queue. Figure 5 on page 7 illustrates this.

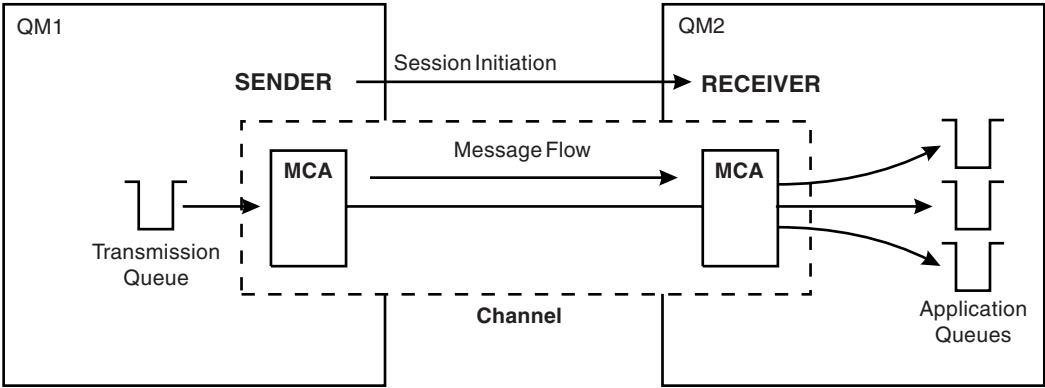


Figure 5. A sender-receiver channel

Requester-server channel:

A requester in one system starts the channel so that it can receive messages from the other system. The requester requests the server at the other end of the channel to start. The server sends messages to the requester from the transmission queue defined in its channel definition.

A server channel can also initiate the communication and send messages to a requester, but this applies only to *fully qualified* servers, that is server channels that have the connection name of the partner specified in the channel definition. A fully qualified server can either be started by a requester, or can initiate a communication with a requester.

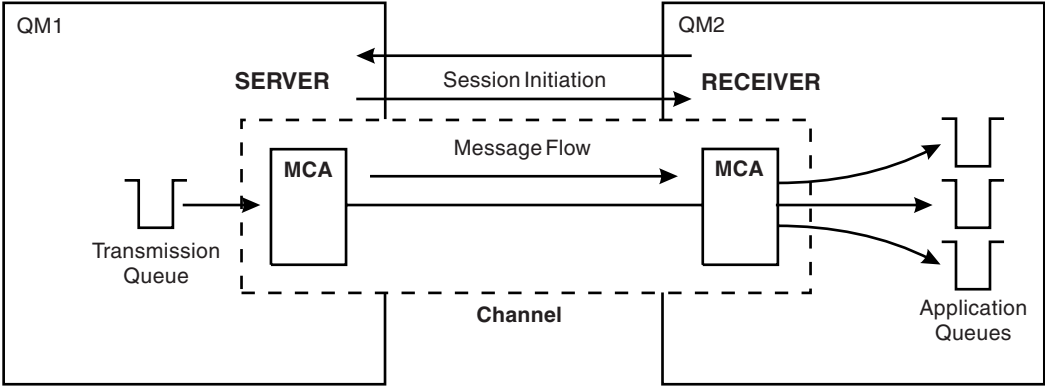


Figure 6. A requester-server channel

Requester-sender channel:

The requester starts the channel and the sender terminates the call. The sender then restarts the communication according to information in its channel definition (this is known as *callback*). It sends messages from the transmission queue to the requester.

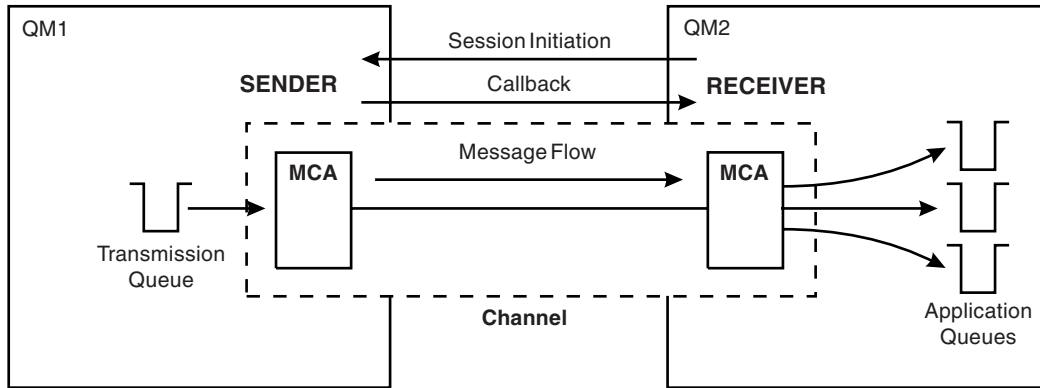


Figure 7. A requester-sender channel

Server-receiver channel:

This is similar to sender-receiver but applies only to *fully qualified* servers, that is server channels that have the connection name of the partner specified in the channel definition. Channel startup must be initiated at the server end of the link. The illustration of this is similar to the illustration in Figure 5 on page 7.

Cluster-sender channels:

In a cluster, each queue manager has a cluster-sender channel on which it can send cluster information to one of the full repository queue managers. Queue managers can also send messages to other queue managers on cluster-sender channels.

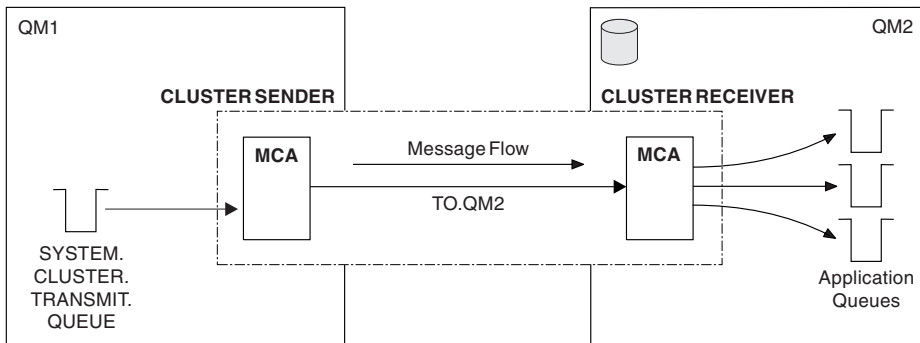


Figure 8. A cluster-sender channel

Cluster-receiver channels:

In a cluster, each queue manager has a cluster-receiver channel on which it can receive messages and information about the cluster. The illustration of this is similar to the illustration in Figure 8.

Message channel agents

A *message channel agent* (MCA) is a program that controls the sending and receiving of messages. There is one message channel agent at each end of a channel. One

MCA takes messages from the transmission queue and puts them on the communication link. The other MCA receives messages and delivers them onto a queue on the remote queue manager.

A message channel agent is called a *caller MCA* if it initiated the communication, otherwise it is called a *responder MCA*. A caller MCA may be associated with a sender, cluster-sender, server (fully qualified), or requester channel. A responder MCA may be associated with any type of message channel, except a cluster sender.

Transmission queues

A *transmission queue* is a special type of local queue used to store messages before they are transmitted by the MCA to the remote queue manager. In a distributed-queuing environment, you need to define one transmission queue for each sending MCA, unless you are using WebSphere MQ Queue Manager clusters.

You specify the name of the transmission queue in a *remote queue definition*, (see “Remote queue definitions” on page 12). If you do not specify the name, the queue manager looks for a transmission queue with the same name as the remote queue manager.

You can specify the name of a default transmission queue for the queue manager. This is used if you do not specify the name of the transmission queue, and a transmission queue with the same name as the remote queue manager does not exist.

Channel initiators and listeners

A *channel initiator* acts as a *trigger monitor* for sender channels, because a transmission queue may be defined as a triggered queue. When a message arrives on a transmission queue that satisfies the triggering criteria for that queue, a message is sent to the initiation queue, triggering the channel initiator to start the appropriate sender channel. You can also start server channels in this way if you specified the connection name of the partner in the channel definition. This means that channels can be started automatically, based upon messages arriving on the appropriate transmission queue.

You need a *channel listener* program to start receiving (responder) MCAs. Responder MCAs are started in response to a startup request from the caller MCA; the channel listener detects incoming network requests and starts the associated channel.

Figure 9 on page 10 shows how channel initiators and channel listeners are used.

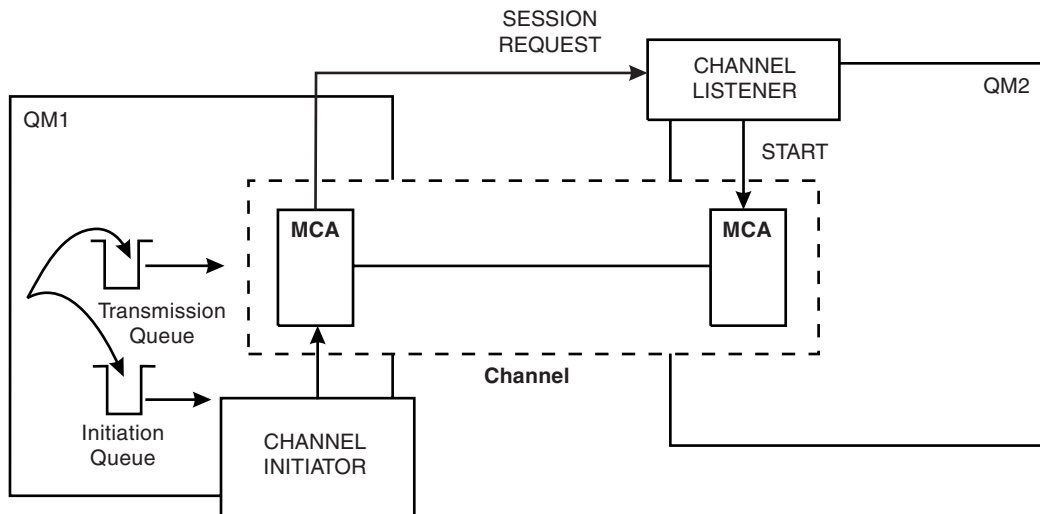


Figure 9. Channel initiators and listeners

The implementation of channel initiators is platform specific.

- In z/OS, there is one channel initiator for each queue manager and it runs as a separate address space. You start it using the WebSphere MQ command `START CHINIT`, which you would normally issue as part of your queue manager startup. It monitors the system-defined queue `SYSTEM.CHANNEL.INITQ`, which is the initiation queue that is recommended for all the transmission queues.
- On other platforms, you can start as many channel initiators as you like, specifying a name for the initiation queue for each one. Normally you need only one initiator. WebSphere MQ for AIX®, i5/OS®, HP-UX, Solaris, and Windows® systems allow you to start up to three channel initiators (the default value), but you can change this value. The default behavior is to start a channel initiator when a queue manager is automatically started. You can change this by setting the `SCHINIT` attribute of the queue manager to `MANUAL`.

The channel initiator is also required for other functions. These are discussed later in this book.

The implementation of channel listeners is platform specific.

- Use the channel listener programs provided by WebSphere MQ if you are using z/OS.

Note: On z/OS, The TCP/IP listener can be started many times with different combinations of port number and address to listen on. For more information, see “Listeners” on page 280.

- On i5/OS, use the channel listener program provided by WebSphere MQ if you are using TCP/IP. If you are using SNA, you do not need a listener program. SNA starts the channel by invoking the receiver program on the remote system.
- On Windows systems, you can use either the channel listener program provided by WebSphere MQ, or the facilities provided by the operating system. If performance is important in your environment and if the environment is stable, you can choose to run the WebSphere MQ listener as a trusted application as

described in “Running channels and listeners as trusted applications” on page 119. See the WebSphere MQ Application Programming Guide for information about trusted applications.

- On UNIX[®] systems, use the channel listener program provided by WebSphere MQ or the facilities provided by the ‘operating system’ (for example, inetd for TCP/IP communications).

By default, threaded channels started by the channel initiator or a listener do not run under that process but under a process called AMQRMPPA, otherwise known as a pool process. To revert to the MQSeries[®] Version 5.2 behavior and have channels run under the originating process you can define an environment variable MQNOREMPOOL. The existence of this variable, set to any value, will cause the channel threads to run as part of the listener or channel initiator process. This can be useful when trying to isolate one or more channels from the rest of the configuration, for example when testing channel exits.

Channel-exit programs

If you want to do some additional processing (for example, encryption or data compression) you can write your own channel-exit programs, or sometimes use SupportPacs. The Transaction Processing SupportPacs library for WebSphere MQ is available on the Internet at: <http://www.ibm.com/software/integration/support/supportpacs/product.html#wmq>

WebSphere MQ calls channel-exit programs at defined places in the processing carried out by the MCA. There are six types of channel exit:

Security exit

Used for security checking, such as authentication of the partner.

Message exit

Used for operations on the message, for example, encryption prior to transmission.

Send and receive exits

Used for operations on split messages, for example, data compression and decompression.

Message-retry exit

Used when there is a problem putting the message to the destination.

Channel auto-definition exit

Used to modify the supplied default definition for an automatically defined receiver or server-connection channel.

The sequence of processing is as follows:

1. The security exits are called after the initial data negotiation between both ends of the channel. These must end successfully for the startup phase to complete and to allow messages to be transferred.
2. The message exit is called by the sending MCA, and then the send exit is called for each part of the message that is transmitted to the receiving MCA.
3. The receiving MCA calls the receive exit when it receives each part of the message, and then calls the message exit when the whole message has been received.

This is illustrated in Figure 10 on page 12.

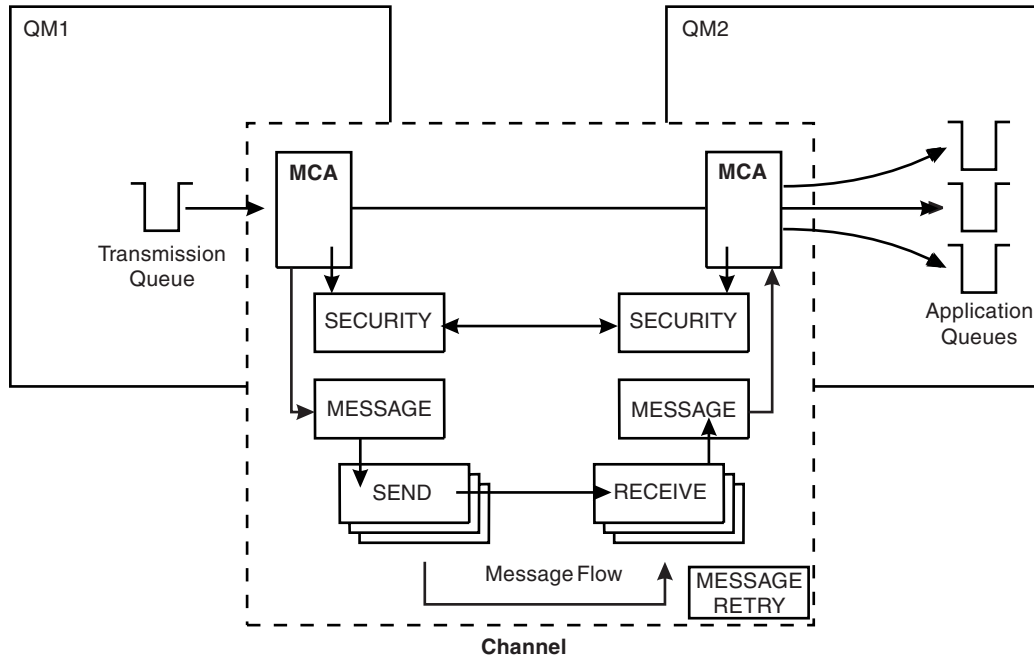


Figure 10. Sequence in which channel exit programs are called

The *message-retry* exit is used to determine how many times the receiving MCA will attempt to put a message to the destination queue before taking alternative action.

For more information about channel exits, see “Channel-exit programs” on page 375.

Dead-letter queues

The *dead-letter queue* (or *undelivered-message queue*) is the queue to which messages are sent if they cannot be routed to their correct destination. Messages are put on this queue when they cannot be put on the destination queue for some reason (for example, because the queue does not exist, or because it is full). Dead-letter queues are also used at the sending end of a channel, for data-conversion errors.

We recommend that you define a dead-letter queue for each queue manager. If you do not, and the MCA is unable to put a message, it is left on the transmission queue and the channel is stopped.

Also, if fast, non-persistent messages (see “Fast, nonpersistent messages” on page 20) cannot be delivered and no DLQ exists on the target system, these messages are discarded.

However, using dead-letter queues can affect the sequence in which messages are delivered, and so you may choose not to use them.

Remote queue definitions

Whereas applications can retrieve messages only from local queues, they can put messages on local queues or remote queues. Therefore, as well as a definition for each of its local queues, a queue manager may have *remote queue definitions*. These

are definitions for queues that are owned by another queue manager. The advantage of remote queue definitions is that they enable an application to put a message to a remote queue without having to specify the name of the remote queue or the remote queue manager, or the name of the transmission queue. This gives you location independence.

There are other uses for remote queue definitions, which will be described later.

How to get to the remote queue manager

You may not always have one channel between each source and target queue manager. Consider these alternative possibilities.

Multi-hopping

If there is no direct communication link between the source queue manager and the target queue manager, it is possible to pass through one or more *intermediate queue managers* on the way to the target queue manager. This is known as a *multi-hop*.

You need to define channels between all the queue managers, and transmission queues on the intermediate queue managers. This is shown in Figure 11.

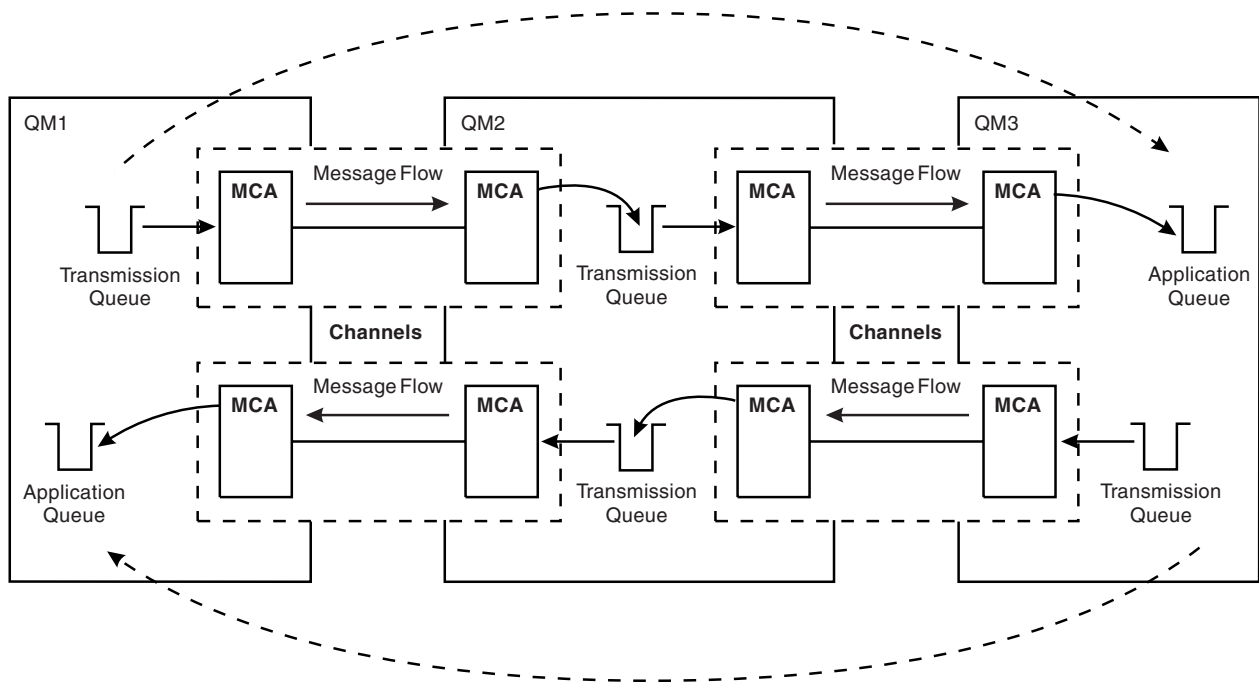


Figure 11. Passing through intermediate queue managers

Sharing channels

As an application designer, you have the choice of forcing your applications to specify the remote queue manager name along with the queue name, or creating a *remote queue definition* for each remote queue. This definition holds the remote queue manager name, the queue name, and the name of the transmission queue.

Either way, all messages from all applications addressing queues at the same remote location have their messages sent through the same transmission queue. This is shown in Figure 12.

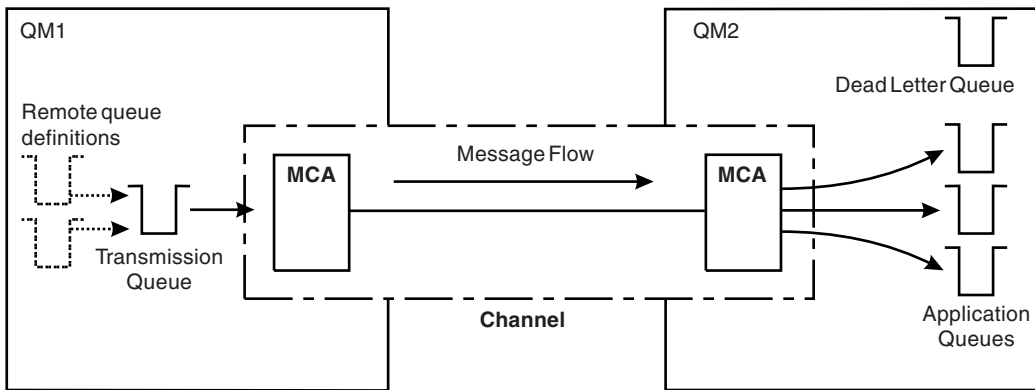


Figure 12. Sharing a transmission queue

Figure 12 illustrates that messages from multiple applications to multiple remote queues can use the same channel.

Using different channels

If you have messages of different types to send between two queue managers, you can define more than one channel between the two. There are times when you need alternative channels, perhaps for security purposes, or to trade off delivery speed against sheer bulk of message traffic.

To set up a second channel you need to define another channel and another transmission queue, and create a remote queue definition specifying the location and the transmission queue name. Your applications can then use either channel but the messages will still be delivered to the same target queues. This is shown in Figure 13.

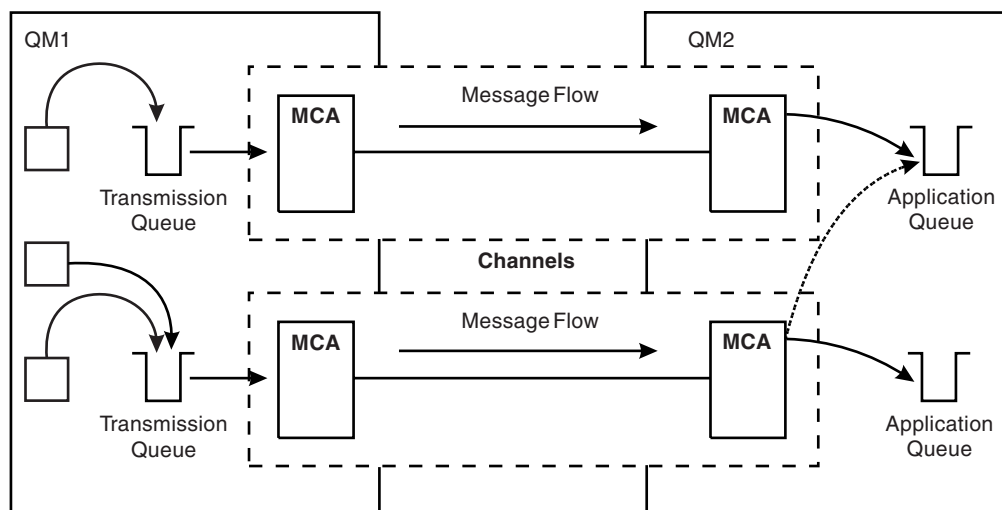


Figure 13. Using multiple channels

When you use remote queue definitions to specify a transmission queue, your applications must *not* specify the location (that is, the destination queue manager) themselves. If they do, the queue manager will not make use of the remote queue definitions. Remote queue definitions make the location of queues and the transmission queue transparent to applications. Applications can put messages to a *logical* queue without knowing where the queue is located and you can alter the *physical* queue without having to change your applications.

Using clustering

Every queue manager within a cluster defines a cluster-receiver channel. When another queue manager wants to send a message to that queue manager, it defines the corresponding cluster-sender channel automatically. For example, if there is more than one instance of a queue in a cluster, the cluster-sender channel could be defined to any of the queue managers that host the queue. WebSphere MQ uses a workload management algorithm that uses a round-robin routine to select an available queue manager to route a message to. For more information see WebSphere MQ Queue Manager Clusters.

Security

Security exits

You can write channel exit programs to perform specific tasks at defined places in the processing carried out by MCA programs. Security exits are a type of channel exit used to verify that the partner at the other end of the channel is genuine. See “Channel security exit programs” on page 378 for more information about channel security exit programs.

Secure sockets layer

WebSphere MQ uses the Secure Sockets Layer (SSL) to provide authentication, confidentiality and integrity at the link level. For more information about SSL in WebSphere MQ, see WebSphere MQ Security.

Making your applications communicate

This chapter provides more detailed information about intercommunication between WebSphere MQ installations. Before reading this chapter it is helpful to have an understanding of channels, queues, and the other concepts introduced in “Concepts of intercommunication” on page 1.

This chapter covers the following topics:

- “How to send a message to another queue manager”
- “Triggering channels” on page 18
- “Safety of messages” on page 20

How to send a message to another queue manager

This section describes the simplest way to send a message from one queue manager to another.

Before you do this you need to do the following:

1. Check that your chosen communication protocol is available.

2. Start the queue managers.
3. Start the channel initiators.
4. Start the listeners.

You also need to have the correct WebSphere MQ security authorization to create the objects required.

To send messages from one queue manager to another:

- Define the following objects on the source queue manager:
 - Sender channel
 - Remote queue definition
 - Initiation queue (required on z/OS, otherwise optional)
 - Transmission queue
 - Dead-letter queue (recommended)
- Define the following objects on the target queue manager:
 - Receiver channel
 - Target queue
 - Dead-letter queue (recommended)

You can use several different methods to define these objects, depending on your WebSphere MQ platform:

- On all platforms, you can use the WebSphere MQ script commands (MQSC) described in *WebSphere MQ Script (MQSC) Command Reference*, the programmable command format (PCF) commands described in *WebSphere MQ Programmable Command Formats and Administration Interface*, or the WebSphere MQ explorer.
- On z/OS you can also use the Operation and Control panels described in *WebSphere MQ for z/OS System Administration Guide*.
- On i5/OS you can also use the panel interface.

The different methods are described in more detail in the platform-specific parts of this book.

Defining the channels

To send messages from one queue manager to another, you need to define two channels; one on the source queue manager and one on the target queue manager.

On the source queue manager

Define a channel with a channel type of SENDER. You need to specify the following:

- The name of the transmission queue to be used (the XMITQ attribute).
- The connection name of the partner system (the CONNAME attribute).
- The name of the communication protocol you are using (the TRPTYPE attribute). On WebSphere MQ for z/OS, the protocol must be TCP or LU6.2. On other platforms, you do not have to specify this. You can leave it to pick up the value from your default channel definition.

Details of all the channel attributes are given in “Channel attributes” on page 71.

On the target queue manager

Define a channel with a channel type of RECEIVER, and the **same** name as the sender channel.

Specify the name of the communication protocol you are using (the TRPTYPE attribute). On WebSphere MQ for z/OS, the protocol must be TCP or LU6.2. On other platforms, you do not have to specify this. You can leave it to pick up the value from your default channel definition.

Note that receiver channel definitions can be generic. This means that if you have several queue managers communicating with the same receiver, the sending channels can all specify the same name for the receiver, and one receiver definition will apply to them all.

When you have defined the channel, you can test it using the PING CHANNEL command. This command sends a special message from the sender channel to the receiver channel and checks that it is returned.

Defining the queues

To send messages from one queue manager to another, you need to define up to six queues; four on the source queue manager and two on the target queue manager.

On the source queue manager

- Remote queue definition

In this definition you specify the following:

Remote queue manager name

The name of the target queue manager.

Remote queue name

The name of the target queue on the target queue manager.

Transmission queue name

The name of the transmission queue. You do not have to specify this. If you do not, a transmission queue with the same name as the target queue manager is used, or, if this does not exist, the default transmission queue is used. You are advised to give the transmission queue the same name as the target queue manager so that the queue is found by default.

- Initiation queue definition

Required on z/OS, and optional on other platforms. On z/OS you must use the initiation queue called SYSTEM.CHANNEL.INITQ and you are recommended to do so on other platforms also.

- Transmission queue definition

A local queue with the USAGE attribute set to XMITQ. If you are using the WebSphere MQ for i5/OS native interface, the USAGE attribute is *TMQ.

- Dead-letter queue definition—recommended

Define a dead-letter queue to which undelivered messages can be written.

On the target queue manager

- Local queue definition

The target queue. The name of this queue must be the same as that specified in the remote queue name field of the remote queue definition on the source queue manager.

- Dead-letter queue definition—recommended
Define a dead-letter queue to which undelivered messages can be written.

Sending the messages

When you put messages on the remote queue defined at the source queue manager, they are stored on the transmission queue until the channel is started. When the channel has been started, the messages are delivered to the target queue on the remote queue manager.

Starting the channel

Start the channel on the sending queue manager using the START CHANNEL command. When you start the sending channel, the receiving channel is started automatically (by the listener) and the messages are sent to the target queue. Both ends of the message channel must be running for messages to be transferred.

Because the two ends of the channel are on different queue managers, they could have been defined with different attributes. To resolve any differences, there is an initial data negotiation between the two ends when the channel starts. In general, the two ends of the channel agree to operate with the attributes needing the fewer resources, thus enabling larger systems to accommodate the lesser resources of smaller systems at the other end of the message channel.

The sending MCA splits large messages before sending them across the channel. They are reassembled at the remote queue manager. This is transparent to the user.

An MCA can transfer messages using multiple threads. This process, called *pipelining* enables the MCA to transfer messages more efficiently, with fewer wait states. This improves channel performance.

Triggering channels

This explanation is intended as an overview of triggering concepts. You can find a complete description in the WebSphere MQ Application Programming Guide.

For platform-specific information see the following:

- For Windows systems and UNIX systems, “Triggering channels” on page 115
- For z/OS, “Transmission queues and triggering channels” on page 260
- For i5/OS, “Triggering channels in WebSphere MQ for i5/OS” on page 339

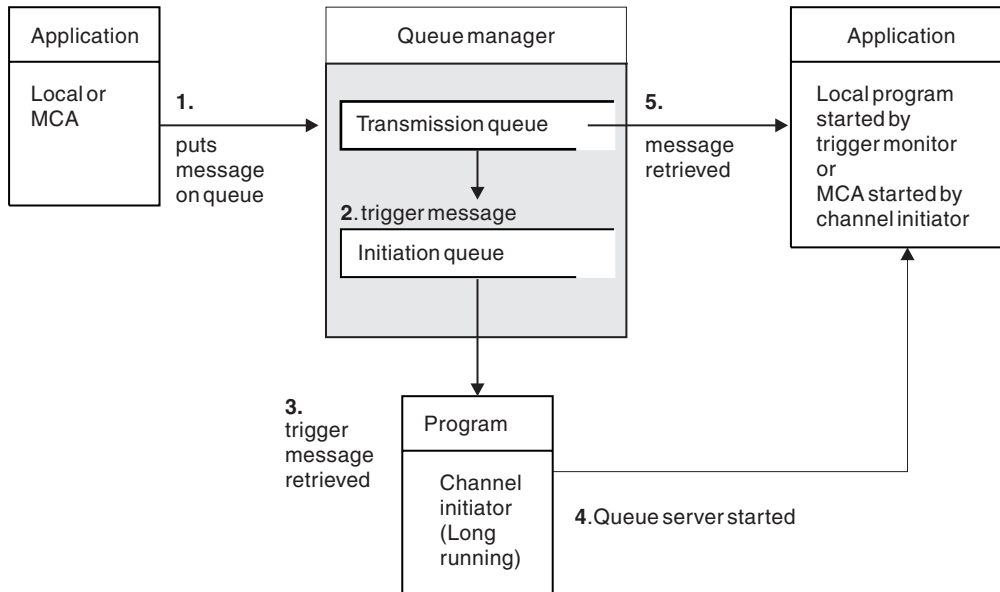


Figure 14. The concepts of triggering

The objects required for triggering are shown in Figure 14. It shows the following sequence of events:

1. The local queue manager places a message from an application or from a message channel agent (MCA) on the transmission queue.
2. When the triggering conditions are fulfilled, the local queue manager places a trigger message on the initiation queue.
3. The long-running channel initiator program monitors the initiation queue, and retrieves messages as they appear.
4. The channel initiator processes the trigger messages according to information contained in them. This information may include the channel name, in which case the corresponding MCA is started.
5. The local application or the MCA, having been triggered, retrieves the messages from the transmission queue.

To set up this scenario, you need to:

- Create the transmission queue with the name of the initiation queue (that is, `SYSTEM.CHANNEL.INITQ`) in the corresponding attribute.
- Ensure that the initiation queue (`SYSTEM.CHANNEL.INITQ`) exists.
- Ensure that the channel initiator program is available and running. The channel initiator program must be provided with the name of the initiation queue in its start command. On z/OS, the name of the initiation queue is fixed, so is not used on the start command.
- Optionally, create the process definition for the triggering, if it does not exist, and ensure that its *UserData* field contains the name of the channel it serves. Instead of creating a process definition, you can specify the channel name in the *TriggerData* attribute of the transmission queue. WebSphere MQ for i5/OS, UNIX systems, and Windows systems, allow the channel name to be specified as blank, in which case the first available channel definition with this transmission queue is used.
- Ensure that the transmission queue definition contains the name of the process definition to serve it, (if applicable), the initiation queue name, and the

triggering characteristics you feel are most suitable. The trigger control attribute allows triggering to be enabled, or not, as necessary.

Note:

1. The channel initiator program acts as a 'trigger monitor' monitoring the initiation queue used to start channels.
2. An initiation queue and trigger process can be used to trigger any number of channels.
3. Any number of initiation queues and trigger processes can be defined.
4. A trigger type of FIRST is recommended, to avoid flooding the system with channel starts.

Safety of messages

In addition to the usual recovery features of WebSphere MQ, distributed queue management ensures that messages are delivered properly by using a syncpoint procedure coordinated between the two ends of the message channel. If this procedure detects an error, it closes the channel to allow you to investigate the problem, and keeps the messages safely in the transmission queue until the channel is restarted.

The syncpoint procedure has an added benefit in that it attempts to recover an *in-doubt* situation when the channel starts up. (*In-doubt* is the status of a unit of recovery for which a syncpoint has been requested but the outcome of the request is not yet known.) Also associated with this facility are the two functions:

1. Resolve with commit or backout
2. Reset the sequence number

The use of these functions occurs only in exceptional circumstances because the channel recovers automatically in most cases.

Fast, nonpersistent messages

The nonpersistent message speed (NPMSPEED) channel attribute can be used to specify that any nonpersistent messages on the channel are to be delivered more quickly. For more information about this attribute, see "Nonpersistent message speed (NPMSPEED)" on page 93.

If a channel terminates while fast, nonpersistent messages are in transit, the messages may be lost and it is up to the application to arrange for their recovery if required.

If the receiving channel cannot put the message to its destination queue then it is placed on the dead letter queue, if one has been defined. If not, the message is discarded.

Note: If the other end of the channel does not support the option, the channel runs at normal speed.

Undelivered messages

For information about what happens when a message cannot be delivered, see "What happens when a message cannot be delivered?" on page 66.

More about intercommunication

This chapter mentions three aliases:

- Remote queue definition
- Queue manager alias definition
- Reply-to queue alias definition

These are all based on the *remote queue definition* object introduced in “Remote queue definitions” on page 12.

This discussion does not apply to *alias queues*. These are described in the WebSphere MQ Application Programming Guide.

This chapter also discusses “Networks” on page 25.

Addressing information

In a single-queue-manager environment, the address of a destination queue is established when an application opens a queue for putting messages to. Because the destination queue is on the same queue manager, there is no need for any addressing information.

In a distributed environment the queue manager needs to know not only the destination queue name, but also the location of that queue (that is, the queue manager name), and the route to that remote location (that is, the transmission queue). When an application puts messages that are destined for a remote queue manager, the local queue manager adds a transmission header to them before placing them on the transmission queue. The transmission header contains the name of the destination queue and queue manager, that is, the *addressing information*. The receiving channel removes the transmission header and uses the information in it to locate the destination queue.

You can avoid the need for your applications to specify the name of the destination queue manager if you use a remote queue definition. This definition specifies the name of the remote queue, the name of the remote queue manager to which messages are destined, and the name of the transmission queue used to transport the messages.

What are aliases?

Aliases are used to provide a quality of service for messages. The queue manager alias enables a system administrator to alter the name of a target queue manager without causing you to have to change your applications. It also enables the system administrator to alter the route to a destination queue manager, or to set up a route that involves passing through a number of other queue managers (multi-hopping). The reply-to queue alias provides a quality of service for replies.

Queue manager aliases and reply-to queue aliases are created using a remote-queue definition that has a blank RNAME. These definitions do not define real queues; they are used by the queue manager to resolve physical queue names, queue manager names, and transmission queues.

Alias definitions are characterized by having a blank RNAME.

Queue name resolution

Queue name resolution occurs at every queue manager each time a queue is opened. Its purpose is to identify the target queue, the target queue manager (which may be local), and the route to that queue manager (which may be null). The resolved name has three parts: the queue manager name, the queue name, and, if the queue manager is remote, the transmission queue.

When a remote queue definition exists, no alias definitions are referenced. The queue name supplied by the application is resolved to the name of the destination queue, the remote queue manager, and the transmission queue specified in the remote queue definition. For more detailed information about queue name resolution, see Chapter 7, “Queue name resolution,” on page 487.

If there is no remote queue definition and a queue manager name is specified, or resolved by the name service, the queue manager looks to see if there is a queue manager alias definition that matches the supplied queue manager name. If there is, the information in it is used to resolve the queue manager name to the name of the destination queue manager. The queue manager alias definition can also be used to determine the transmission queue to the destination queue manager.

If the resolved queue name is not a local queue, both the queue manager name and the queue name are included in the transmission header of each message put by the application to the transmission queue.

The transmission queue used usually has the same name as the resolved queue manager, unless changed by a remote queue definition or a queue manager alias definition. If you have not defined such a transmission queue but you have defined a default transmission queue, then this is used.

Note: Names of queue managers running on z/OS are limited to four characters.

Queue manager alias definitions

Queue manager alias definitions apply when an application that opens a queue to put a message, specifies the queue name **and** the queue manager name.

Queue manager alias definitions have three uses:

- When sending messages, remapping the queue manager name
- When sending messages, altering or specifying the transmission queue
- When receiving messages, determining whether the local queue manager is the intended destination for those messages

Outbound messages - remapping the queue manager name

Queue manager alias definitions can be used to remap the queue manager name specified in an MQOPEN call. For example, an MQOPEN call specifies a queue name of THISQ and a queue manager name of YOURQM. At the local queue manager there is a queue manager alias definition like this:

```
DEFINE QREMOTE (YOURQM) RQMNAME(REALQM)
```

This shows that the real queue manager to be used, when an application puts messages to queue manager YOURQM, is REALQM. If the local queue manager is REALQM, it puts the messages to the queue THISQ, which is a local queue. If the local queue manager is not called REALQM, it routes the message to a

transmission queue called REALQM. The queue manager changes the transmission header to say REALQM instead of YOURQM.

Outbound messages - altering or specifying the transmission queue

Figure 15 shows a scenario where messages arrive at queue manager 'QM1' with transmission headers showing queue names at queue manager 'QM3'. In this scenario, 'QM3' is reachable by multi-hopping through 'QM2'.

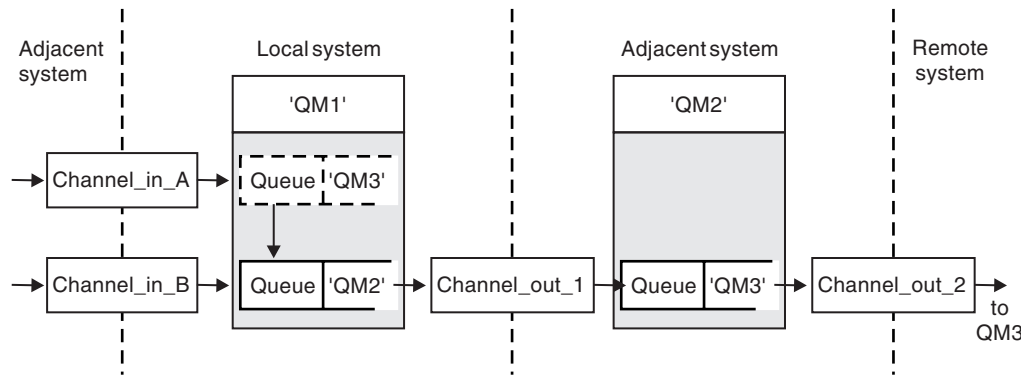


Figure 15. Queue manager alias

All messages for 'QM3' are captured at 'QM1' with a queue manager alias. The queue manager alias is named 'QM3' and contains the definition 'QM3' via transmission queue QM2'. The definition looks like this:

```
DEFINE QREMOTE (QM3) RNAME(' ') RQMNAME(QM3) XMITQ(QM2)
```

The queue manager puts the messages on transmission queue 'QM2' but does not make any alteration to the transmission queue header because the name of the destination queue manager, 'QM3', does not alter.

All messages arriving at 'QM1' and showing a transmission header containing a queue name at 'QM2' are also put on the 'QM2' transmission queue. In this way, messages with different destinations are collected onto a common transmission queue to an appropriate adjacent system, for onward transmission to their destinations.

Inbound messages - determining the destination

A receiving MCA opens the queue referenced in the transmission header. If a queue manager alias definition exists with the same name as the queue manager referenced, then the queue manager name received in the transmission header is replaced with the RQMNAME from that definition.

This has two uses:

- Directing messages to another queue manager
- Altering the queue manager name to be the same as the local queue manager

Reply-to queue alias definitions

When an application needs to reply to a message it may look at the data in the *message descriptor* of the message it received to find out the name of the queue to which it should reply. It is up to the sending application to suggest where replies should be sent and to attach this information to its messages. This has to be coordinated as part of your application design.

What is a reply-to queue alias definition?

A reply-to queue alias definition specifies alternative names for the reply information in the message descriptor. The advantage of this is that you can alter the name of a queue or queue manager without having to alter your applications. Queue name resolution takes place at the sending end, before the message is put to a queue.

Note: This is an unusual use of queue-name resolution. It is the only situation in which name resolution takes place at a time when a queue is not being opened.

Normally an application specifies a reply-to queue and leaves the reply-to queue manager name blank. The queue manager fills in its own name at put time. This works well except when you want an alternate channel to be used for replies, for example, a channel that uses transmission queue 'QM1_relief' instead of the default return channel which uses transmission queue 'QM1'. In this situation, the queue manager names specified in transmission-queue headers do not match "real" queue manager names but are re-specified using queue manager alias definitions. In order to return replies along alternate routes, it is necessary to map reply-to queue data as well, using reply-to queue alias definitions.

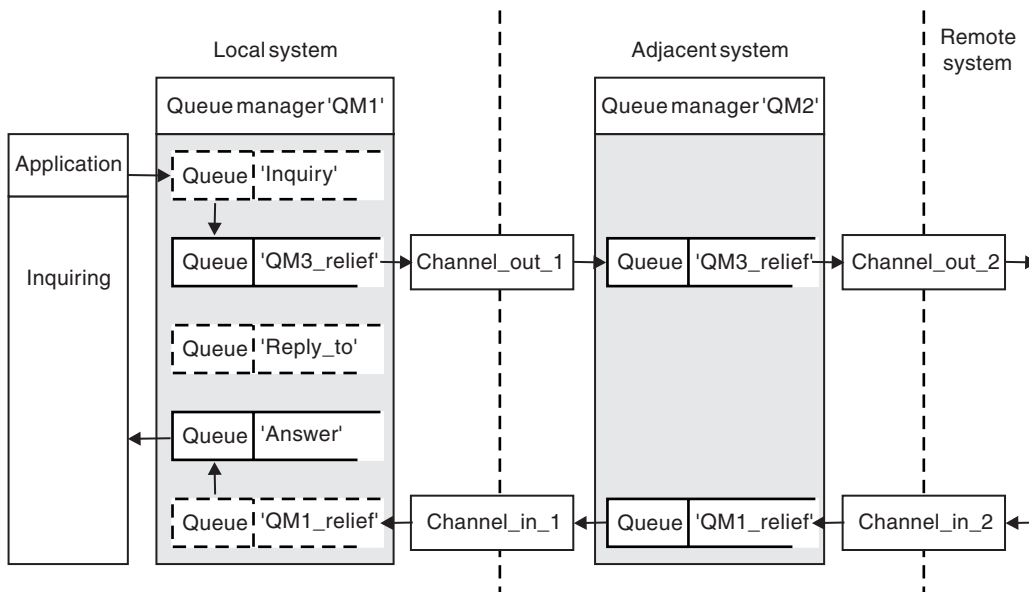


Figure 16. Reply-to queue alias used for changing reply location

In the example in Figure 16:

1. The application puts a message using the MQPUT call and specifying the following in the message descriptor:

```
ReplyToQ='Reply_to'  
ReplyToQMgr=''
```

Note that ReplyToQMgr must be blank in order for the reply-to queue alias to be used.

2. You create a reply-to queue alias definition called 'Reply_to', which contains the name 'Answer', and the queue manager name 'QM1_relief'.

```
DEFINE QREMOTE ('Reply_to') RNAME ('Answer')  
          RQMNAME ('QM1_relief')
```
3. The messages are sent with a message descriptor showing ReplyToQ='Answer' and ReplyToQMgr='QM1_relief'.
4. The application specification must include the information that replies are to be found in queue 'Answer' rather than 'Reply_to'.

To prepare for the replies you have to create the parallel return channel. This involves defining:

- At QM2, the transmission queue named 'QM1_relief'

```
DEFINE QLOCAL ('QM1_relief') USAGE(XMITQ)
```

- At QM1, the queue manager alias QM1_relief'

```
DEFINE QREMOTE ('QM1_relief') RNAME() RQMNAME(QM1)
```

This queue manager alias terminates the chain of parallel return channels and captures the messages for QM1.

If you think you might want to do this at sometime in the future, arrange for your applications to use the alias name from the start. For now this is a normal queue alias to the reply-to queue, but later it can be changed to a queue manager alias.

Reply-to queue name

Care is needed with naming reply-to queues. The reason that an application puts a reply-to queue name in the message is that it can specify the queue to which its replies will be sent. But when you create a reply-to queue alias definition with this name, you cannot have the actual reply-to queue (that is, a local queue definition) with the same name. Therefore, the reply-to queue alias definition must contain a new queue name as well as the queue manager name, and the application specification must include the information that its replies will be found in this other queue.

The applications now have to retrieve the messages from a different queue from the one they named as the reply-to queue when they put the original message.

Networks

So far this book has covered creating channels between your system and any other system with which you need to have communications, and creating multi-hop channels to systems where you have no direct connections. The message channel connections described in the scenarios are shown as a network diagram in Figure 17 on page 26.

Channel and transmission queue names

You can give transmission queues any name you like, but to avoid confusion, you can give them the same names as the destination queue manager names, or queue

manager alias names, as appropriate, to associate them with the route they use. This gives a clear overview of parallel routes that you create through intermediate (multi-hopped) queue managers.

This is not quite so clear-cut for channel names. The channel names in Figure 17 for QM2, for example, must be different for incoming and outgoing channels. All channel names may still contain their transmission queue names, but they must be qualified to make them unique.

For example, at QM2, there is a QM3 channel coming from QM1, and a QM3 channel going to QM3. To make the names unique, the first one may be named 'QM3_from_QM1', and the second may be named 'QM3_from_QM2'. In this way, the channel names show the transmission queue name in the first part of the name, and the direction and adjacent queue manager name in the second part of the name.

A table of suggested channel names for Figure 17 is given in Table 1.

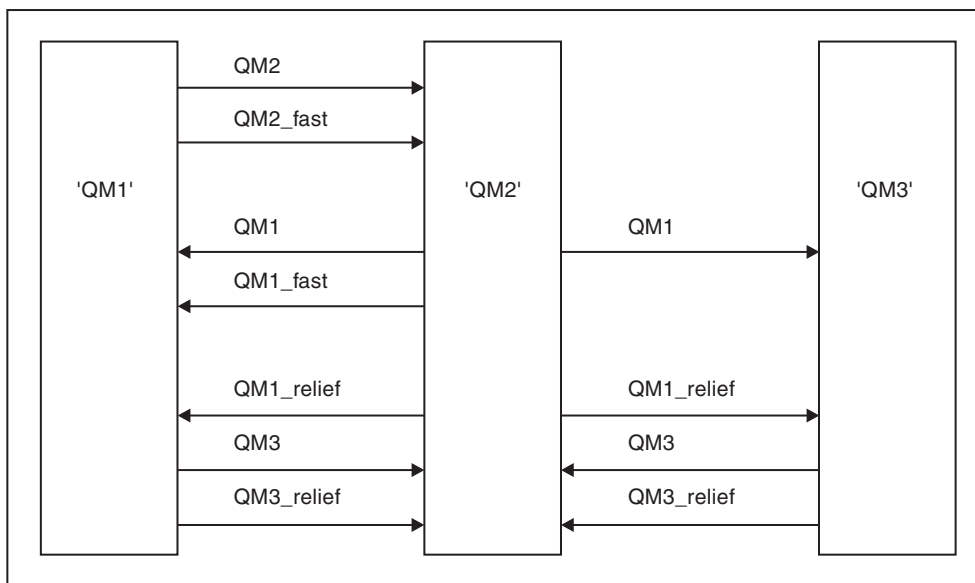


Figure 17. Network diagram showing all channels

Table 1. Example of channel names

Route name	Queue managers hosting channel	Transmission queue name	Suggested channel name
QM1	QM1 & QM2	QM1 (at QM2)	QM1.from.QM2
QM1	QM2 & QM3	QM1 (at QM3)	QM1.from.QM3
QM1_fast	QM1 & QM2	QM1_fast (at QM2)	QM1_fast.from.QM2
QM1_relief	QM1 & QM2	QM1_relief (at QM2)	QM1_relief.from.QM2
QM1_relief	QM2 & QM3	QM1_relief (at QM3)	QM1_relief.from.QM3
QM2	QM1 & QM2	QM2 (at QM1)	QM2.from.QM1
QM2_fast	QM1 & QM2	QM2_fast (at QM1)	QM2_fast.from.QM1
QM3	QM1 & QM2	QM3 (at QM1)	QM3.from.QM1
QM3	QM2 & QM3	QM3 (at QM2)	QM3.from.QM2

Table 1. Example of channel names (continued)

Route name	Queue managers hosting channel	Transmission queue name	Suggested channel name
QM3_relief	QM1 & QM2	QM3_relief (at QM1)	QM3_relief.from.QM1
QM3_relief	QM2 & QM3	QM3_relief (at QM2)	QM3_relief.from.QM2

Note:

1. On WebSphere MQ for z/OS, queue manager names are limited to 4 characters.
2. You are strongly recommended to name all the channels in your network uniquely. As shown in Table 1 on page 26, including the source and target queue manager names in the channel name is a good way to do this.

Network planner

This chapter has discussed application designer, systems administrator, and channel planner functions. Creating a network assumes that there is another, higher level function of *network planner* whose plans are implemented by the other members of the team.

If an application is used widely, it is more economical to think in terms of local access sites for the concentration of message traffic, using wide-band links between the local access sites, as shown in Figure 18 on page 28.

In this example there are two main systems and a number of satellite systems (The actual configuration would depend on business considerations.) There are two concentrator queue managers located at convenient centers. Each QM-concentrator has message channels to the local queue managers:

- QM-concentrator 1 has message channels to each of the three local queue managers, QM1, QM2, and QM3. The applications using these queue managers can communicate with each other through the QM-concentrators.
- QM-concentrator 2 has message channels to each of the three local queue managers, QM4, QM5, and QM6. The applications using these queue managers can communicate with each other through the QM-concentrators.
- The QM-concentrators have message channels between themselves thus allowing any application at a queue manager to exchange messages with any other application at another queue manager.

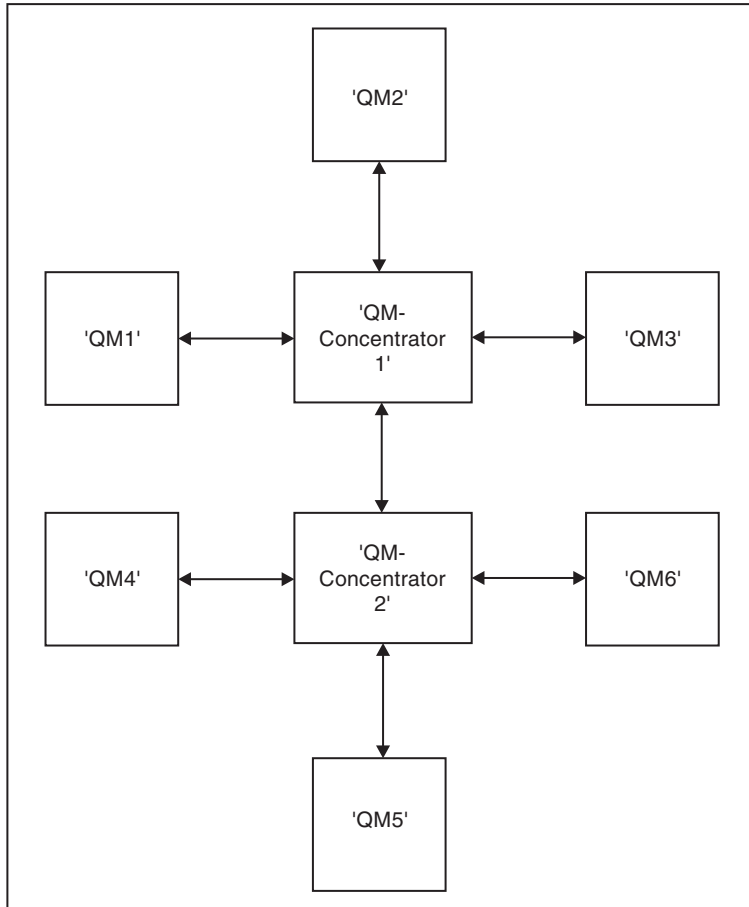


Figure 18. Network diagram showing QM-concentrators

Chapter 2. How intercommunication works

WebSphere MQ distributed-messaging techniques

This chapter describes techniques that are of use when planning channels. It introduces the concept of message flow control and explains how this is arranged in distributed queue management (DQM). It gives more detailed information about the concepts introduced in the preceding chapters and starts to show how you might use distributed queue management. This chapter covers the following topics:

- “Message flow control”
- “Putting messages on remote queues” on page 31
- “Choosing the transmission queue” on page 33
- “Receiving messages” on page 33
- “Passing messages through your system” on page 35
- “Separating message flows” on page 36
- “Concentrating messages to diverse locations” on page 38
- “Diverting message flows to another destination” on page 39
- “Sending messages to a distribution list” on page 40
- “Reply-to queue” on page 41
- “Networking considerations” on page 46
- “Return routing” on page 47
- “Managing queue name translations” on page 47
- “Channel message sequence numbering” on page 49
- “Loopback testing” on page 49
- “Route tracing and activity recording” on page 50

Message flow control

Message flow control is a task that involves the setting up and maintenance of message routes between queue managers. This is very important for routes that multi-hop through many queue managers.

You control message flow using a number of techniques that were introduced in “Making your applications communicate” on page 15. If your queue manager is in a cluster, message flow is controlled using different techniques, as described in the WebSphere MQ Queue Manager Clusters book. If your queue managers are in a queue sharing group and intra-group queuing (IGQ) is enabled, then the message flow can be controlled by IGQ agents, which are described in “Intra-group queuing” on page 298.

This chapter describes how you use your system’s queues, alias queue definitions, and message channels to achieve message flow control.

You make use of the following objects:

- Transmission queues
- Message channels
- Remote queue definition

- Queue manager alias definition
- Reply-to queue alias definition

The queue manager and queue objects are described in the WebSphere MQ System Administration Guide book for WebSphere MQ for UNIX systems, and Windows systems, in the WebSphere MQ for i5/OS System Administration Guide book for WebSphere MQ for i5/OS, in the WebSphere MQ for z/OS Concepts and Planning Guide for WebSphere MQ for z/OS, or in the *MQSeries System Management Guide* for your platform. Message channels are described in “Message channels” on page 6. The following techniques use these objects to create message flows in your system:

- Putting messages to remote queues
- Routing via particular transmission queues
- Receiving messages
- Passing messages through your system
- Separating message flows
- Switching a message flow to another destination
- Resolving the reply-to queue name to an alias name

Note

All the concepts described in this chapter are relevant for all nodes in a network, and include sending and receiving ends of message channels. For this reason, only one node is illustrated in most examples, except where the example requires explicit cooperation by the administrator at the other end of a message channel.

Before proceeding to the individual techniques it is useful to recap on the concepts of name resolution and the three ways of using remote queue definitions. See “More about intercommunication” on page 21.

Queue names in transmission header

The queue name used by the application, the logical queue name, is resolved by the queue manager to the destination queue name, that is, the physical queue name. This destination queue name travels with the message in a separate data area, the transmission header, until the destination queue has been reached after which the transmission header is stripped off.

You will be changing the queue manager part of this queue name when you create parallel classes of service. Remember to return the queue manager name to the original name when the end of the class of service diversion has been reached.

How to create queue manager and reply-to aliases

As discussed above, the remote queue definition object is used in three different ways. Table 2 on page 31 explains how to define each of the three ways:

- Using a remote queue definition to redefine a local queue name.

The application provides only the queue name when opening a queue, and this queue name is the name of the remote queue definition.

The remote queue definition contains the names of the target queue and queue manager, and optionally, the definition can contain the name of the transmission queue to be used. If no transmission queue name is provided, the queue manager uses the queue manager name, taken from the remote queue definition,

for the transmission queue name. If a transmission queue of this name is not defined, but a default transmission queue is defined, the default transmission queue is used.

- Using a remote queue definition to redefine a queue manager name.

The application, or channel program, provides a queue name together with the remote queue manager name when opening the queue.

If you have provided a remote queue definition with the same name as the queue manager name, and you have left the queue name in the definition blank, then the queue manager will substitute the queue manager name in the open call with the queue manager name in the definition.

In addition, the definition can contain the name of the transmission queue to be used. If no transmission queue name is provided, the queue manager takes the queue manager name, taken from the remote queue definition, for the transmission queue name. If a transmission queue of this name is not defined, but a default transmission queue is defined, the default transmission queue is used.

- Using a remote queue definition to redefine a reply-to queue name.

Each time an application puts a message to a queue, it may provide the name of a reply-to queue for answer messages but with the queue manager name blank.

If you provide a remote queue definition with the same name as the reply-to queue then the local queue manager replaces the reply-to queue name with the queue name from your definition.

You may provide a queue manager name in the definition, but not a transmission queue name.

Table 2. Three ways of using the remote queue definition object

Usage	Queue manager name	Queue name	Transmission queue name
1. Remote queue definition (on OPEN call)			
Supplied in the call	blank or local QM	(*) required	-
Supplied in the definition	required	required	optional
2. Queue manager alias (on OPEN call)			
Supplied in the call	(*) required and not local QM	required	-
Supplied in the definition	required	blank	optional
3. Reply-to queue alias (on PUT call)			
Supplied in the call	blank	(*) required	-
Supplied in the definition	optional	optional	blank
Note: (*) means that this name is the name of the definition object			

For a formal description, see Chapter 7, “Queue name resolution,” on page 487.

Putting messages on remote queues

In a distributed-queuing environment, a transmission queue and channel are the focal point for all messages to a location whether the messages originate from applications in your local system, or arrive through channels from an adjacent system. This is shown in Figure 19 on page 32 where an application is placing messages on a logical queue named ‘QA_norm’. The name resolution uses the

remote queue definition 'QA_norm' to select the transmission queue 'QMB', and adds a transmission header to the messages stating 'QA_norm at QMB'.

Messages arriving from the adjacent system on 'Channel_back' have a transmission header with the physical queue name 'QA_norm at QMB', for example. These messages are placed unchanged on transmission queue QMB.

The channel moves the messages to an adjacent queue manager.

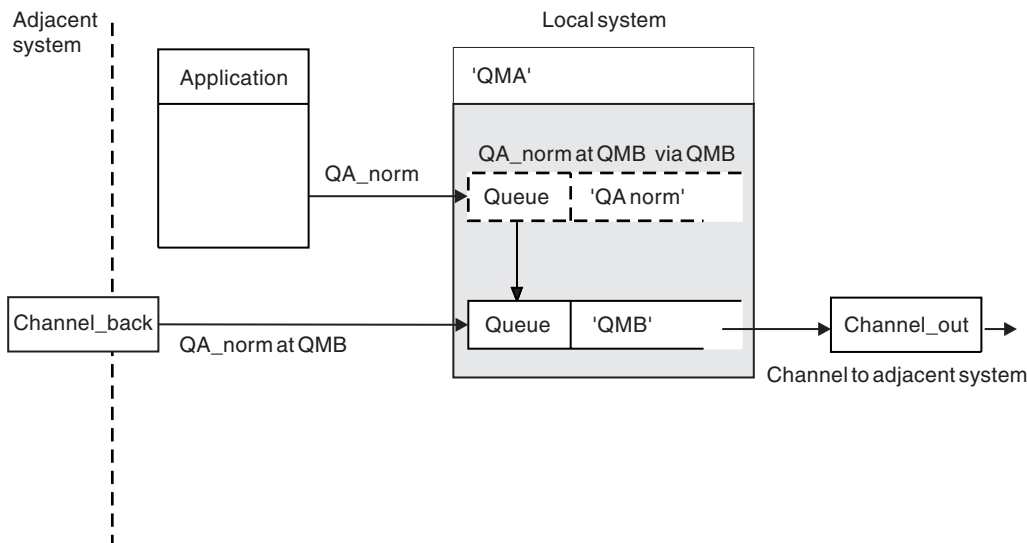


Figure 19. A remote queue definition is used to resolve a queue name to a transmission queue to an adjacent queue manager. Note: The dashed outline represents a remote queue definition. This is not a real queue, but a name alias that is controlled as though it were a real queue.

If you are the WebSphere MQ system administrator, you must:

- Define the message channel from the adjacent system
- Define the message channel to the adjacent system
- Create the transmission queue 'QMB'
- Define the remote queue object 'QA_norm' to resolve the queue name used by applications to the desired destination queue name, destination queue manager name, and transmission queue name

In a clustering environment, you only need to define a cluster-receiver channel at the local queue manager. You do not need to define a transmission queue or a remote queue object. For information about this, see the WebSphere MQ Queue Manager Clusters book.

More about name resolution

The effect of the remote queue definition is to define a physical destination queue name and queue manager name; these names are put in the transmission headers of messages.

Incoming messages from an adjacent system have already had this type of name resolution carried out by the original queue manager, and have the transmission header showing the physical destination queue name and queue manager name. These messages are unaffected by remote queue definitions.

Choosing the transmission queue

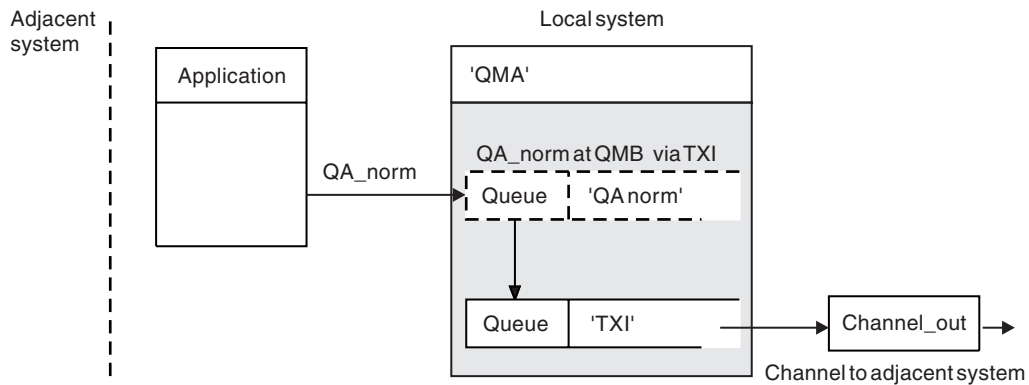


Figure 20. The remote queue definition allows a different transmission queue to be used

In a distributed-queuing environment, when you need to change a message flow from one channel to another, use the same system configuration as shown in Figure 19 on page 32. Figure 20 shows how you use the remote queue definition to send messages over a different transmission queue, and therefore over a different channel, to the same adjacent queue manager.

For the configuration shown in Figure 20 you must provide:

- The remote queue object 'QA_norm' to choose:
 - Queue 'QA_norm' at the remote queue manager
 - Transmission queue 'TXI'
 - Queue manager 'QMB_priority'
- The transmission queue 'TXI'. Specify this in the definition of the channel to the adjacent system

Messages are placed on transmission queue 'TXI' with a transmission header containing 'QA_norm at QMB_priority', and are sent over the channel to the adjacent system.

The channel_back has been left out of this illustration because it would need a queue manager alias; this is discussed in the following example.

In a clustering environment, you do not need to define a transmission queue or a remote queue definition. For more information about this, see the WebSphere MQ Queue Manager Clusters book.

Receiving messages

Configure the queue manager to receive messages from other queue managers. Ensure that unintentional name resolution does not occur.

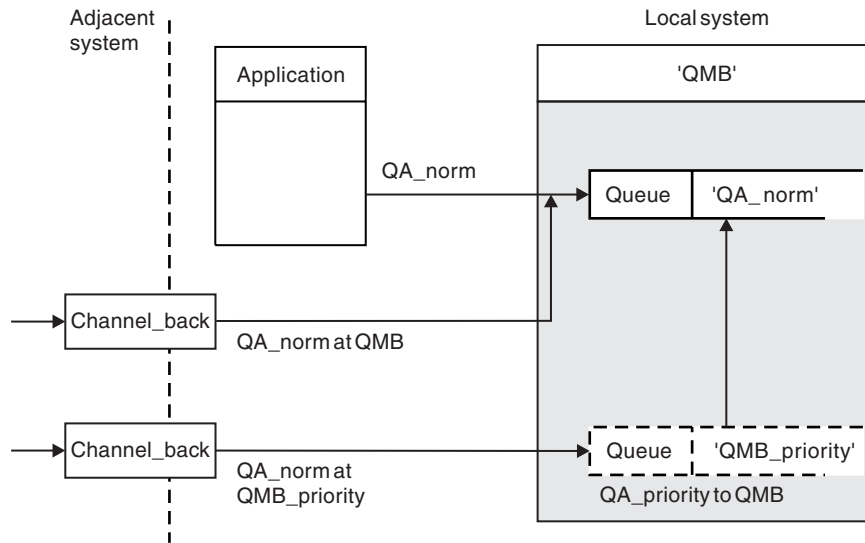


Figure 21. Receiving messages directly, and resolving alias queue manager name

As well as arranging for messages to be sent, the system administrator must also arrange for messages to be received from adjacent queue managers. Received messages contain the physical name of the destination queue manager and queue in the transmission header. They are treated exactly the same as messages from a local application that specifies both queue manager name and queue name. Because of this, you need to ensure that messages entering your system do not have an unintentional name resolution carried out. See Figure 21 for this scenario.

For this configuration, you must prepare:

- Message channels to receive messages from adjacent queue managers
- A queue manager alias definition to resolve an incoming message flow, 'QMB_priority', to the local queue manager name, 'QMB'
- The local queue, 'QA_norm', if it does not already exist

Receiving alias queue manager names

The use of the queue manager alias definition in this illustration has not selected a different destination queue manager. Messages passing through this local queue manager and addressed to 'QMB_priority' are intended for queue manager 'QMB'. The alias queue manager name is used to create the separate message flow.

Passing messages through your system

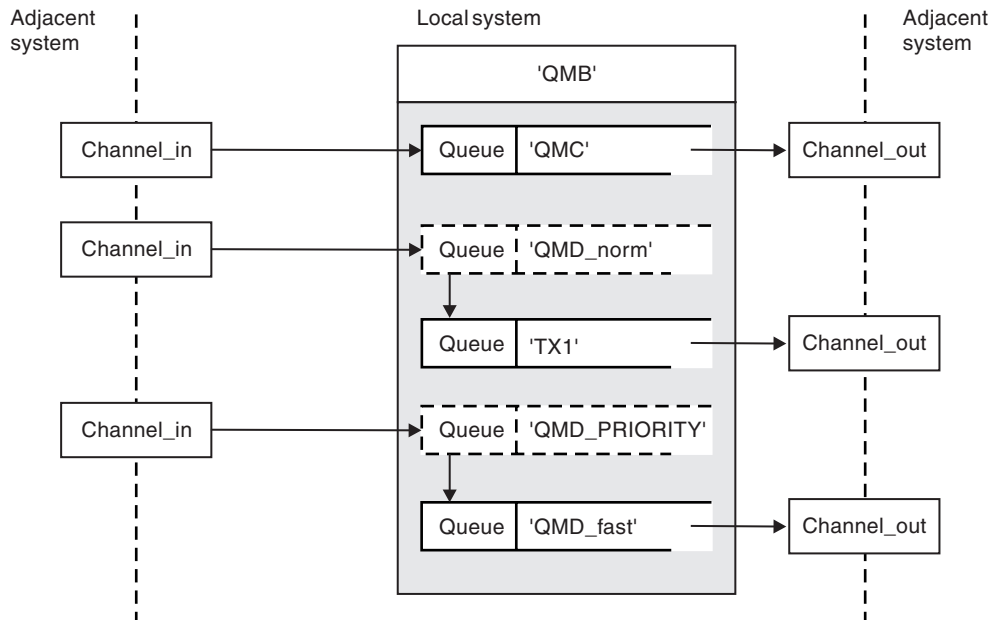


Figure 22. Three methods of passing messages through your system

Following on from the technique shown in Figure 21 on page 34, where you saw how an alias flow is captured, Figure 22 illustrates the ways networks are built up by bringing together the techniques we have discussed.

The configuration shows a channel delivering three messages with different destinations:

1. 'QB at QMC'
2. 'QB at QMD_norm'
3. 'QB at QMD_PRIORITY'

You must pass the first message flow through your system unchanged; the second message flow through a different transmission queue and channel, while reverting the messages from the alias queue manager name 'QMD_norm' to the physical location 'QMD'; and the third message flow simply chooses a different transmission queue without any other change.

In a clustering environment, all messages are passed through the cluster transmission queue, `SYSTEM.CLUSTER.TRANSMIT.QUEUE`. This is illustrated in Figure 4 on page 5.

The following methods describe techniques applicable to a distributed-queuing environment:

Method 1: Using the incoming location name

When you are going to receive messages with a transmission header containing another location name, the simplest preparation is to have a transmission queue

with that name, 'QMC' in this example, as a part of a channel to an adjacent queue manager. The messages are delivered unchanged.

Method 2: Using an alias for the queue manager

The second method is to use the queue manager alias object definition, but specify a new location name, 'QMD', as well as a particular transmission queue, 'TX1'.

This action:

- Terminates the alias message flow set up by the queue manager name alias 'QMD_norm'. That is the named class of service 'QMD_norm'.
- Changes the transmission headers on these messages from 'QMD_norm' to 'QMD'.

Method 3: Selecting a transmission queue

The third method is to have a queue manager alias object defined with the same name as the destination location, 'QMD_PRIORITY', and use the definition to select a particular transmission queue, 'QMD_fast', and therefore another channel. The transmission headers on these messages remain unchanged.

Using these methods

For these configurations, you must prepare the:

- Input channel definitions
- Output channel definitions
- Transmission queues:
 - QMC
 - TX1
 - QMD_fast
- Queue manager alias definitions:
 - QMD_norm with 'QMD_norm to QMD via TX1'
 - QMD_PRIORITY with 'QMD_PRIORITY to QMD_PRIORITY via QMD_fast'

Note

None of the message flows shown in the example changes the destination queue. The queue manager name aliases simply provide separation of message flows.

Separating message flows

In a distributed-queuing environment, the need to separate messages to the same queue manager into different message flows can arise for a number of reasons. For example:

- You may need to provide a separate flow for very large, medium, and small messages. This also applies in a clustering environment and, in this case, you may create clusters that overlap. There are a number of reasons you might do this, for example:
 - To allow different organizations to have their own administration.
 - To allow independent applications to be administered separately.
 - To create a class of service. For example you could have a cluster called STAFF that is a subset of the cluster called STUDENTS. When you put a message to a queue advertised in the STAFF cluster, a restricted channel is

used. When you put a message to a queue advertised in the STUDENTS cluster, either a general channel or a restricted channel may be used.

- To create test and production environments.
- It may be necessary to route incoming messages via different paths from the path of the locally generated messages.
- Your installation may require to schedule the movement of messages at certain times (for example, overnight) and the messages then need to be stored in reserved queues until scheduled.

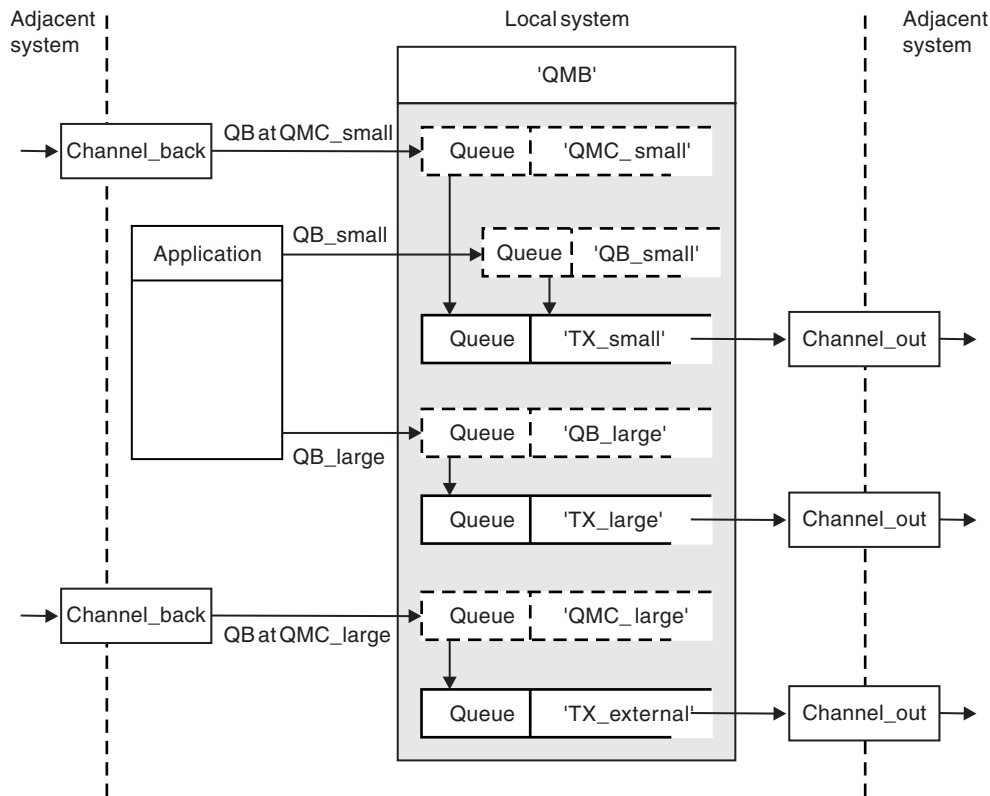


Figure 23. Separating messages flows

In the example shown in Figure 23, the two incoming flows are to alias queue manager names 'QMC_small' and 'QMC_large'. You provide these flows with a queue manager alias definition to capture these flows for the local queue manager. You have an application addressing two remote queues and you need these message flows to be kept separate. You provide two remote queue definitions that specify the same location, 'QMC', but specify different transmission queues. This keeps the flows separate, and nothing extra is needed at the far end as they have the same destination queue manager name in the transmission headers. You provide:

- The incoming channel definitions
- The two remote queue definitions QB_small and QB_large
- The two queue manager alias definitions QMC_small and QMC_large
- The three sending channel definitions
- Three transmission queues: TX_small, TX_large, and TX_external

Coordination with adjacent systems

When you use a queue manager alias to create a separate message flow, you need to coordinate this activity with the system administrator at the remote end of the message channel to ensure that the corresponding queue manager alias is available there.

Concentrating messages to diverse locations

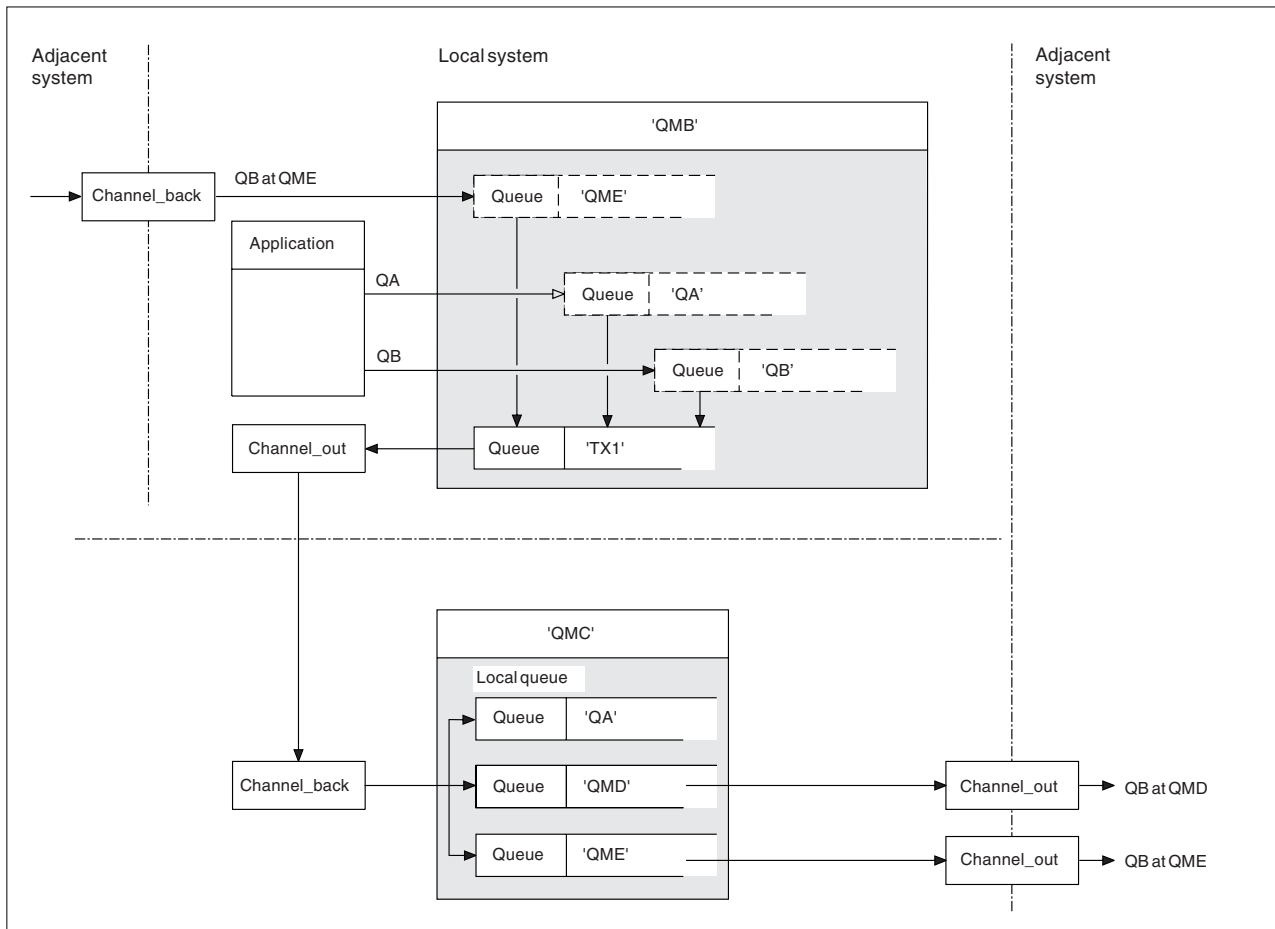


Figure 24. Combining message flows on to a channel

Figure 24 illustrates a distributed-queuing technique for concentrating messages that are destined for various locations on to one channel. Two possible uses would be:

- Concentrating message traffic through a gateway
- Using wide bandwidth highways between nodes

In this example, messages from different sources, local and adjacent, and having different destination queues and queue managers, are flowed via transmission queue 'TX1' to queue manager QMC. Queue manager QMC delivers the messages according to the destinations, one set to a transmission queue 'QMD' for onward transmission to queue manager QMD, another set to a transmission queue 'QME' for onward transmission to queue manager QME, while other messages are put on the local queue 'QA'.

You must provide:

- Channel definitions
- Transmission queue TX1
- Remote queue definitions:
 - QA with 'QA at QMC via TX1'
 - QB with 'QB at QMD via TX1'
- Queue manager alias definition:
 - QME with 'QME via TX1'

The complementary administrator who is configuring QMC must provide:

- Receiving channel definition with the same channel name
- Transmission queue QMD with associated sending channel definition
- Transmission queue QME with associated sending channel definition
- Local queue object QA.

Diverting message flows to another destination

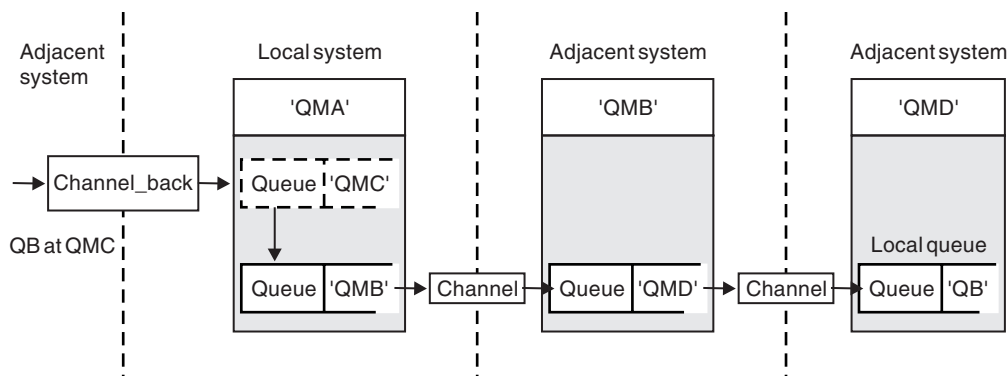


Figure 25. Diverting message streams to another destination

Figure 25 illustrates how you can redefine the destination of certain messages. Incoming messages to QMA are destined for 'QB at QMC'. They would normally arrive at QMA and be placed on a transmission queue called QMC which would have been part of a channel to QMC. QMA must divert the messages to QMD, but is able to reach QMD only over QMB. This method is useful when you need to move a service from one location to another, and allow subscribers to continue to send messages on a temporary basis until they have adjusted to the new address.

The method of rerouting incoming messages destined for a certain queue manager to a different queue manager uses:

- A queue manager alias to change the destination queue manager to another queue manager, and to select a transmission queue to the adjacent system
- A transmission queue to serve the adjacent queue manager
- A transmission queue at the adjacent queue manager for onward routing to the destination queue manager

You must provide:

- Channel_back definition

- Queue manager alias object definition QMC with QB at QMD via QMB
- Channel_out definition
- The associated transmission queue QMB

The complementary administrator who is configuring QMB must provide:

- The corresponding channel_back definition
- The transmission queue, QMD
- The associated channel definition to QMD

You can use aliases within a clustering environment. For information about this, see the WebSphere MQ Queue Manager Clusters book.

Sending messages to a distribution list

In WebSphere MQ on all platforms except z/OS, an application can send a message to several destinations with a single MQPUT call. This applies in both a distributed-queuing environment and a clustering environment. You have to define the destinations in a distribution list, as described in the WebSphere MQ Application Programming Guide.

Not all queue managers support distribution lists. When an MCA establishes a connection with a partner, it determines whether or not the partner supports distribution lists and sets a flag on the transmission queue accordingly. If an application tries to send a message that is destined for a distribution list but the partner does not support distribution lists, the sending MCA intercepts the message and puts it onto the transmission queue once for each intended destination.

A receiving MCA ensures that messages sent to a distribution list are safely received at all the intended destinations. If any destinations fail, the MCA establishes which ones have failed so that it can generate exception reports for them and can try to re-send the messages to them.

Reply-to queue

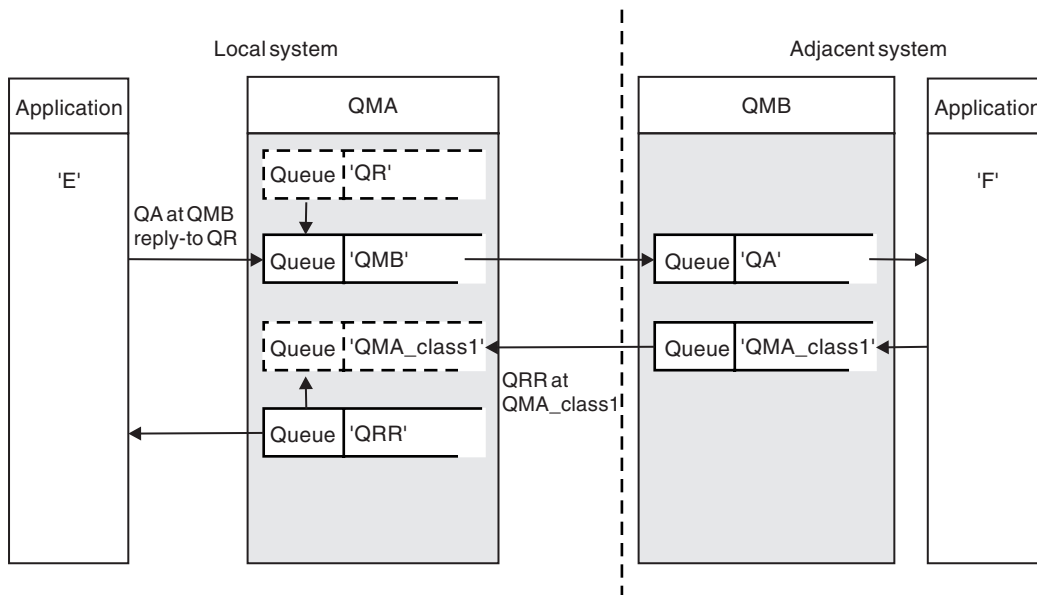


Figure 26. Reply-to queue name substitution during PUT call

A complete remote queue processing loop using a reply-to queue is shown in Figure 26. This applies in both a distributed-queuing environment and a clustering environment. The details are as shown in Table 6 on page 48.

The application opens QA at QMB and puts messages on that queue. The messages are given a reply-to queue name of QR, without the queue manager name being specified. Queue manager QMA finds the reply-to queue object QR and extracts from it the alias name of QRR and the queue manager name QMA_class1. These names are put into the reply-to fields of the messages.

Reply messages from applications at QMB are addressed to QRR at QMA_class1. The queue manager alias name definition QMA_class1 is used by the queue manager to flow the messages to itself, and to queue QRR.

This scenario depicts the way you give applications the facility to choose a class of service for reply messages, the class being implemented by the transmission queue QMA_class1 at QMB, together with the queue manager alias definition, QMA_class1 at QMA. In this way, you can change an application's reply-to queue so that the flows are segregated without involving the application. That is, the application always chooses QR for this particular class of service, and you have the opportunity to change the class of service with the reply-to queue definition QR.

You must create:

- Reply-to queue definition QR
- Transmission queue object QMB
- Channel_out definition
- Channel_back definition
- Queue manager alias definition QMA_class1
- Local queue object QRR, if it does not exist

The complementary administrator at the adjacent system must create:

- Receiving channel definition
- Transmission queue object QMA_class1
- Associated sending channel
- Local queue object QA.

Your application programs use:

- Reply-to queue name QR in put calls
- Queue name QRR in get calls

In this way, you may change the class of service as necessary, without involving the application, by changing the reply-to alias 'QR', together with the transmission queue 'QMA_class1' and queue manager alias 'QMA_class1'.

If no reply-to alias object is found when the message is put on the queue, the local queue manager name is inserted in the blank reply-to queue manager name field, and the reply-to queue name remains unchanged.

Name resolution restriction

Because the name resolution has been carried out for the reply-to queue at 'QMA' when the original message was put, no further name resolution is allowed at 'QMB', that is, the message is put with the physical name of the reply-to queue by the replying application.

Note that the applications must be aware of the naming convention that the name they use for the reply-to queue is different from the name of the actual queue where the return messages are to be found.

For example, when two classes of service are provided for the use of applications with reply-to queue alias names of 'C1_alias', and 'C2_alias', the applications use these names as reply-to queue names in the message put calls, but the applications will actually expect messages to appear in queues 'C1' and 'C2', respectively.

However, an application is able to make an inquiry call on the reply-to alias queue to check for itself the name of the real queue it must use to get the reply messages.

Reply-to queue alias example

This example illustrates the use of a reply-to alias to select a different route (transmission queue) for returned messages. The use of this facility requires the reply-to queue name to be changed in cooperation with the applications.

As shown in Figure 27 on page 43, the return route must be available for the reply messages, including the transmission queue, channel, and queue manager alias.

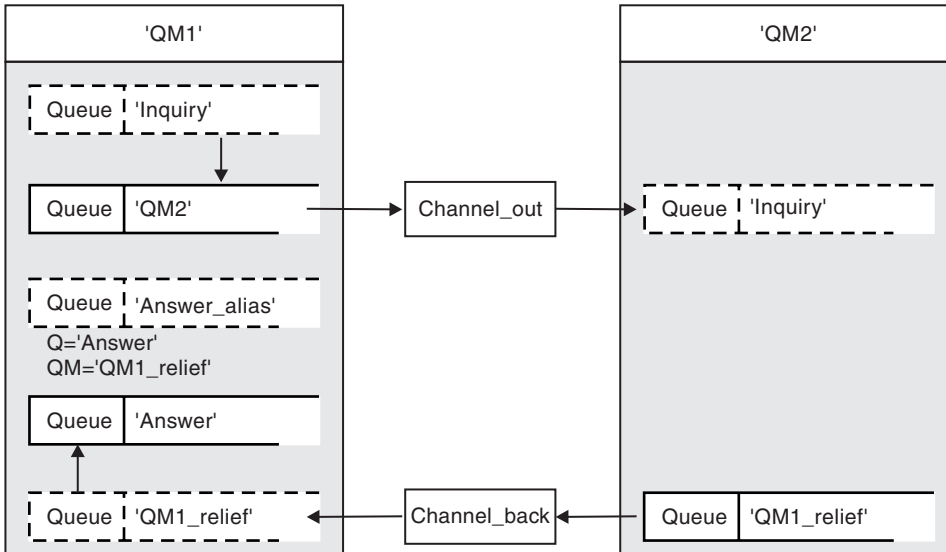


Figure 27. Reply-to queue alias example

This example is for requester applications at 'QM1' that send messages to server applications at 'QM2'. The servers' messages are to be returned through an alternative channel using transmission queue 'QM1_relief' (the default return channel would be served with a transmission queue 'QM1').

The reply-to queue alias is a particular use of the remote queue definition named 'Answer_alias'. Applications at QM1 include this name, 'Answer_alias', in the reply-to field of all messages that they put on queue 'Inquiry'.

Reply-to queue definition 'Answer_alias' is defined as 'Answer at QM1_relief'. Applications at QM1 expect their replies to appear in the local queue named 'Answer'.

Server applications at QM2 use the reply-to field of received messages to obtain the queue and queue manager names for the reply messages to the requester at QM1.

Definitions used in this example at QM1:

The WebSphere MQ system administrator at QM1 must ensure that the reply-to queue 'Answer' is created along with the other objects. The name of the queue manager alias, marked with a '*', must agree with the queue manager name in the reply-to queue alias definition, also marked with an '*'.

Object	Definition	
Local transmission queue	QM2	
Remote queue definition	Object name	Inquiry
	Remote queue manager name	QM2
	Remote queue name	Inquiry
	Transmission queue name	QM2 (DEFAULT)
Queue manager alias	Object name	QM1_relief *
	Queue manager name	QM1
	Queue name	(blank)
Reply-to queue alias	Object name	Answer_alias
	Remote queue manager name	QM1_relief *

Object	Definition	
	Remote queue name	Answer

Definitions used in this example at QM2:

The WebSphere MQ system administrator at QM2 must ensure that the local queue exists for the incoming messages, and that the correctly named transmission queue is available for the reply messages.

Object	Definition
Local queue	Inquiry
Transmission queue	QM1_relief

Put definition at QM1:

Applications fill the reply-to fields with the reply-to queue alias name, and leave the queue manager name field blank.

Field	Content
Queue name	Inquiry
Queue manager name	(blank)
Reply-to queue name	Answer_alias
Reply-to queue manager	(blank)

Put definition at QM2:

Applications at QM2 retrieve the reply-to queue name and queue manager name from the original message and use them when putting the reply message on the reply-to queue.

Field	Content
Queue name	Answer
Queue manager name	QM1_relief

How the example works

In this example, requester applications at QM1 always use 'Answer_alias' as their reply-to queue in the relevant field of the put call, and they always retrieve their messages from the queue named 'Answer'.

The reply-to queue alias definitions are available for use by the QM1 system administrator to change the name of the reply-to queue 'Answer', and of the return route 'QM1_relief'.

Changing the queue name 'Answer' is normally not useful because the QM1 applications are expecting their answers in this queue. However, the QM1 system administrator is able to change the return route (class of service), as necessary.

How the queue manager makes use of the reply-to queue alias

Queue manager QM1 retrieves the definitions from the reply-to queue alias when the reply-to queue name, included in the put call by the application, is the same as the reply-to queue alias, and the queue manager part is blank.

The queue manager replaces the reply-to queue name in the put call with the queue name from the definition. It replaces the blank queue manager name in the put call with the queue manager name from the definition.

These names are carried with the message in the message descriptor.

Table 3. Reply-to queue alias

Field name	Put call	Transmission header
Reply-to queue name	Answer_alias	Answer
Reply-to queue manager name	(blank)	QM1_relief

Reply-to queue alias walk-through

To complete this example, let us take a walk through the process, from an application putting a message on a remote queue at queue manager 'QM1', through to the same application removing the reply message from the alias reply-to queue.

1. The application opens a queue named 'Inquiry', and puts messages to it. The application sets the reply-to fields of the message descriptor to:

Reply-to queue name	Answer_alias
Reply-to queue manager name	(blank)

2. Queue manager 'QM1' responds to the blank queue manager name by checking for a remote queue definition with the name 'Answer_alias'. If none is found, the queue manager places its own name, 'QM1', in the reply-to queue manager field of the message descriptor.
3. If the queue manager finds a remote queue definition with the name 'Answer_alias', it extracts the queue name and queue manager names from the definition (queue name='Answer' and queue manager name='QM1_relief') and puts them into the reply-to fields of the message descriptor.
4. The queue manager 'QM1' uses the remote queue definition 'Inquiry' to determine that the intended destination queue is at queue manager 'QM2', and the message is placed on the transmission queue 'QM2'. 'QM2' is the default transmission queue name for messages destined for queues at queue manager 'QM2'.
5. When queue manager 'QM1' puts the message on the transmission queue, it adds a transmission header to the message. This header contains the name of the destination queue, 'Inquiry', and the destination queue manager, 'QM2'.
6. The message arrives at queue manager 'QM2', and is placed on the 'Inquiry' local queue.
7. An application gets the message from this queue and processes the message. The application prepares a reply message, and puts this reply message on the reply-to queue name from the message descriptor of the original message. This is:

Reply-to queue name	Answer
Reply-to queue manager name	QM1_relief

8. Queue manager 'QM2' carries out the put command, and finding that the queue manager name, 'QM1_relief', is a remote queue manager, it places the message on the transmission queue with the same name, 'QM1_relief'. The

message is given a transmission header containing the name of the destination queue, 'Answer', and the destination queue manager, 'QM1_relief'.

9. The message is transferred to queue manager 'QM1' where the queue manager, recognizing that the queue manager name 'QM1_relief' is an alias, extracts from the alias definition 'QM1_relief' the physical queue manager name 'QM1'.
10. Queue manager 'QM1' then puts the message on the queue name contained in the transmission header, 'Answer'.
11. The application extracts its reply message from the queue 'Answer'.

Networking considerations

In a distributed-queuing environment, because message destinations are addressed with just a queue name and a queue manager name, the following rules apply:

1. Where the queue manager name is given, and the name is different from the local queue manager's name:
 - A transmission queue must be available with the same name, and this transmission queue must be part of a message channel moving messages to another queue manager, or
 - A queue manager alias definition must exist to resolve the queue manager name to the same, or another queue manager name, and optional transmission queue, or
 - If the transmission queue name cannot be resolved, and a default transmission queue has been defined, the default transmission queue is used.
2. Where only the queue name is supplied, a queue of any type but with the same name must be available on the local queue manager. This queue may be a remote queue definition which resolves to: a transmission queue to an adjacent queue manager, a queue manager name, and an optional transmission queue.

To see how this works in a clustering environment, see the WebSphere MQ Queue Manager Clusters book.

If the queue managers are running in a queue-sharing group (QSG) and intra-group queuing (IGQ) is enabled, you can use the `SYSTEM.QSG.TRANSMIT.QUEUE`. For more information, see "Intra-group queuing" on page 298.

Consider the scenario of a message channel moving messages from one queue manager to another in a distributed-queuing environment.

The messages being moved have originated from any other queue manager in the network, and some messages may arrive that have an unknown queue manager name as destination. This can occur when a queue manager name has changed or has been removed from the system, for example.

The channel program recognizes this situation when it cannot find a transmission queue for these messages, and places the messages on your undelivered-message (dead-letter) queue. It is your responsibility to look for these messages and arrange for them to be forwarded to the correct destination, or to return them to the originator, where this can be ascertained.

Exception reports are generated in these circumstances, if report messages were requested in the original message.

Name resolution convention

It is strongly recommended that name resolution that changes the identity of the destination queue, (that is, logical to physical name changing), should only occur once, and only at the originating queue manager.

Subsequent use of the various alias possibilities should be used only when separating and combining message flows.

Return routing

Messages may contain a return address in the form of the name of a queue and queue manager. This applies in both a distributed-queuing environment and a clustering environment. This address is normally specified by the application that creates the message, but may be modified by any application that subsequently handles the message, including user exit applications.

Irrespective of the source of this address, any application handling the message may choose to use this address for returning answer, status, or report messages to the originating application.

The way these response messages are routed is not different from the way the original message is routed. You need to be aware that the message flows you create to other queue managers will need corresponding return flows.

Physical name conflicts

The destination reply-to queue name has been resolved to a physical queue name at the original queue manager, and must not be resolved again at the responding queue manager.

This is a likely possibility for name conflict problems that can only be prevented by a network-wide agreement on physical and logical queue names.

Managing queue name translations

This description is mainly provided for application designers and channel planners concerned with an individual system that has message channels to adjacent systems. It takes a local view of channel planning and control.

When you create a queue manager alias definition or a remote queue definition, the name resolution is carried out for every message carrying that name, regardless of the source of the message. To oversee this situation, which may involve large numbers of queues in a queue manager network, you keep tables of:

- The names of source queues and of source queue managers with respect to resolved queue names, resolved queue manager names, and resolved transmission queue names, with method of resolution
- The names of source queues with respect to:
 - Resolved destination queue names
 - Resolved destination queue manager names
 - Transmission queues
 - Message channel names
 - Adjacent system names

- Reply-to queue names

Note: The use of the term *source* in this context refers to the queue name or the queue manager name provided by the application, or a channel program when opening a queue for putting messages.

An example of each of these tables is shown in Table 4, Table 5, and Table 6.

The names in these tables are derived from the examples in this chapter, and this table is not intended as a practical example of queue name resolution in one node.

Table 4. Queue name resolution at queue manager QMA

Source queue specified when queue is opened	Source queue manager specified when queue is opened	Resolved queue name	Resolved queue manager name	Resolved transmission queue name	Resolution type
QA_norm	-	QA_norm	QMB	QMB	Remote queue
(any)	QMB	-	-	QMB	(none)
QA_norm	-	QA_norm	QMB	TX1	Remote queue
QB	QMC	QB	QMD	QMB	Queue manager alias

Table 5. Queue name resolution at queue manager QMB

Source queue specified when queue is opened	Source queue manager specified when queue is opened	Resolved queue name	Resolved queue manager name	Resolved transmission queue name	Resolution type
QA_norm	-	QA_norm	QMB	-	(none)
QA_norm	QMB	QA_norm	QMB	-	(none)
QA_norm	QMB_PRIORITY	QA_norm	QMB	-	Queue manager alias
(any)	QMC	(any)	QMC	QMC	(none)
(any)	QMD_norm	(any)	QMD_norm	TX1	Queue manager alias
(any)	QMD_PRIORITY	(any)	QMD_PRIORITY	QMD_fast	Queue manager alias
(any)	QMC_small	(any)	QMC_small	TX_small	Queue manager alias
(any)	QMC_large	(any)	QMC_large	TX_external	Queue manager alias
QB_small	QMC	QB_small	QMC	TX_small	Remote queue
QB_large	QMC	QB_large	QMC	TX_large	Remote queue
(any)	QME	(any)	QME	TX1	Queue manager alias
QA	QMC	QA	QMC	TX1	Remote queue
QB	QMD	QB	QMD	TX1	Remote queue

Table 6. Reply-to queue name translation at queue manager QMA

Application design		Reply-to alias definition	
Local QMGR QMA	Queue name for messages QRR	Reply-to queue alias name QR	Redefined to QRR at QMA_class1

Channel message sequence numbering

The channel uses sequence numbers to assure that messages are delivered, delivered without duplication, and stored in the same order as they were taken from the transmission queue. The sequence number is generated at the sending end of the channel and is incremented by one before being used, which means that the current sequence number is the number of the last message sent. This information can be displayed using DISPLAY CHSTATUS (see WebSphere MQ Script (MQSC) Command Reference). The sequence number and an identifier called the LUWID are stored in persistent storage for the last message transferred in a batch. These values are used during channel start-up to ensure that both ends of the link agree on which messages have been transferred successfully.

Sequential retrieval of messages

If an application puts a sequence of messages to the same destination queue, those messages can be retrieved in sequence by a *single* application with a sequence of MQGET operations, if the following conditions are met:

- All of the put requests were done from the same application.
- All of the put requests were either from the same unit of work, or all the put requests were made outside of a unit of work.
- The messages all have the same priority.
- The messages all have the same persistence.
- For remote queuing, the configuration is such that there can only be one path from the application making the put request, through its queue manager, through intercommunication, to the destination queue manager and the target queue.
- The messages are not put to a dead-letter queue (for example, if a queue is temporarily full).
- The application getting the message does not deliberately change the order of retrieval, for example by specifying a particular *MsgId* or *CorrelId* or by using message priorities.
- Only one application is doing get operations to retrieve the messages from the destination queue. If this is not the case, these applications must be designed to get all the messages in each sequence put by a sending application.

Note: Messages from other tasks and units of work may be interspersed with the sequence, even where the sequence was put from within a single unit of work.

If these conditions cannot be met, and the order of messages on the target queue is important, then the application can be coded to use its own message sequence number as part of the message to assure the order of the messages.

Sequence of retrieval of fast, nonpersistent messages

Nonpersistent messages on a fast channel may overtake persistent messages on the same channel and so arrive out of sequence. The receiving MCA puts the nonpersistent messages on the destination queue immediately and makes them visible. Persistent messages are not made visible until the next syncpoint.

Loopback testing

Loopback testing is a technique on non-z/OS platforms that allows you to test a communications link without actually linking to another machine. You set up a

connection between two queue managers as though they are on separate machines, but you test the connection by looping back to another process on the same machine. This means that you can test your communications code without requiring an active network.

The way you do this depends on which products and protocols you are using.

On Windows systems, you can use the “loopback” adapter.

Refer to the documentation for the products you are using for more information.

Route tracing and activity recording

To confirm the route a message will take through a series of queue managers, you can use the WebSphere MQ display route application, available through the control command `dspmqrte`, or you can use activity recording. Both of these topics are described in Monitoring WebSphere MQ.

DQM implementation

This chapter describes the implementation of the concepts introduced in “Making your applications communicate” on page 15.

Distributed queue management (DQM):

- Enables you to define and control communication channels between queue managers
- Provides you with a message channel service to move messages from a type of *local queue*, known as a transmission queue, to communication links on a local system, and from communication links to local queues at a destination queue manager
- Provides you with facilities for monitoring the operation of channels and diagnosing problems, using panels, commands, and programs

This chapter discusses:

- “Functions of DQM”
- “Message sending and receiving” on page 51
- “Channel control function” on page 53
- “What happens when a message cannot be delivered?” on page 66
- “Initialization and configuration files” on page 68
- “Data conversion” on page 70
- “Writing your own message channel agents” on page 70

Functions of DQM

Distributed queue management has these functions:

- Message sending and receiving
- Channel control
- Initialization file
- Data conversion
- Channel exits

Channel definitions associate channel names with transmission queues, communication link identifiers, and channel attributes. Channel definitions are implemented in different ways on different platforms. Message sending and receiving is controlled by programs known as *message channel agents* (MCAs), which use the channel definitions to start up and control communication.

The MCAs in turn are controlled by DQM itself. The structure is platform dependent, but typically includes listeners and trigger monitors, together with operator commands and panels.

A *message channel* is a one-way pipe for moving messages from one queue manager to another. Thus a message channel has two end-points, represented by a pair of MCAs. Each end-point has a definition of its end of the message channel. For example, one end would define a sender, the other end a receiver.

For details of how to define channels, see:

- “Monitoring and controlling channels on Windows and Unix platforms” on page 105
- “Monitoring and controlling channels on z/OS” on page 243
- “Monitoring and controlling channels on i5/OS” on page 321

For information about channel exits, see “Channel-exit programs” on page 375.

Message sending and receiving

Figure 28 on page 52 shows the relationships between entities when messages are transmitted, and shows the flow of control.

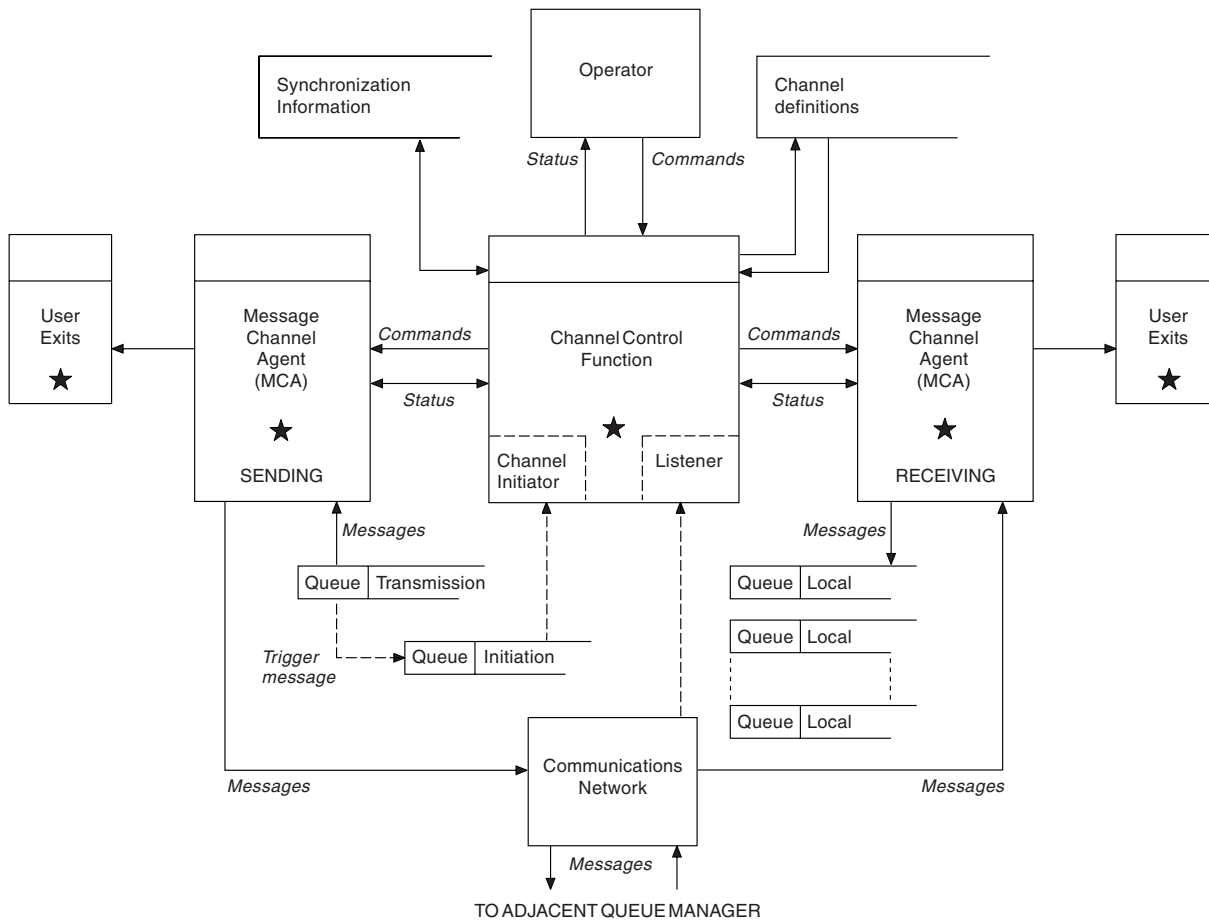


Figure 28. Distributed queue management model

Note:

1. There is one MCA per channel, depending on the platform. There may be one or more channel control functions for a given queue manager.
2. The implementation of MCAs and channel control functions is highly platform dependent; they may be programs or processes or threads, and they may be a single entity or many comprising several independent or linked parts.
3. All components marked with a star can use the MQI.

Channel parameters

An MCA receives its parameters in one of several ways:

- If started by a command, the channel name is passed in a data area. The MCA then reads the channel definition directly to obtain its attributes.
- For sender, and in some cases server channels, the MCA can be started automatically by the queue manager trigger. The channel name is retrieved from the trigger process definition, where applicable, and is passed to the MCA. The remaining processing is the same as that described above. Server channels should only be set up to trigger if they are fully qualified, that is, they specify a CONNAME to connect to.
- If started remotely by a sender, server, requester, or client-connection, the channel name is passed in the initial data from the partner message channel agent. The MCA reads the channel definition directly to obtain its attributes.

Certain attributes not defined in the channel definition are also negotiable:

Split messages

If one end does not support this, split messages will not be sent.

Conversion capability

If one end cannot perform the necessary code page conversion or numeric encoding conversion when needed, the other end must handle it. If neither end supports it, when needed, the channel cannot start.

Distribution list support

If one end does not support distribution lists, the partner MCA sets a flag in its transmission queue so that it will know to intercept messages intended for multiple destinations.

Channel status and sequence numbers

Message channel agent programs keep records of the current sequence number and logical unit of work number for each channel, and of the general status of the channel. Some platforms allow you to display this status information to help you control channels.

Channel control function

The channel control function provides facilities for you to define, monitor, and control channels. Commands are issued through panels, programs, or from a command line to the channel control function. The panel interface also displays channel status and channel definition data. You can use Programmable Command Formats or those WebSphere MQ commands (MQSC) and control commands that are detailed in “Monitoring and controlling channels on Windows and Unix platforms” on page 105.

The commands fall into the following groups:

- Channel administration
- Channel control
- Channel status monitoring

Channel administration commands deal with the definitions of the channels. They enable you to:

- Create a channel definition
- Copy a channel definition
- Alter a channel definition
- Delete a channel definition

Channel control commands manage the operation of the channels. They enable you to:

- Start a channel
- Stop a channel
- Re-synchronize with partner (in some implementations)
- Reset message sequence numbers
- Resolve an in-doubt batch of messages
- Ping; send a test communication across the channel

Channel monitoring displays the state of channels, for example:

- Current channel settings
- Whether the channel is active or inactive
- Whether the channel terminated in a synchronized state

Preparing channels

Before trying to start a message channel or MQI channel, you must make sure that all the attributes of the local and remote channel definitions are correct and compatible. “Channel attributes” on page 71 describes the channel definitions and attributes.

Although you set up explicit channel definitions, the channel negotiations carried out when a channel starts up may override one or other of the values defined. This is quite normal, and transparent, and has been arranged like this so that otherwise incompatible definitions can work together.

Auto-definition of receiver and server-connection channels:

In WebSphere MQ on all platforms except z/OS, if there is no appropriate channel definition, then for a receiver or server-connection channel that has auto-definition enabled, a definition is created automatically. The definition is created using:

1. The appropriate model channel definition, SYSTEM.AUTO.RECEIVER or SYSTEM.AUTO.SVRCONN. The model channel definitions for auto-definition are the same as the system defaults, SYSTEM.DEF.RECEIVER and SYSTEM.DEF.SVRCONN, except for the description field, which is “Auto-defined by” followed by 49 blanks. The systems administrator can choose to change any part of the supplied model channel definitions.
2. Information from the partner system. The partner’s values are used for the channel name and the sequence number wrap value.
3. A channel exit program, which you can use to alter the values created by the auto-definition. See “Channel auto-definition exit program” on page 393.

The description is then checked to determine whether it has been altered by an auto-definition exit or because the model definition has been changed. If the first 44 characters are still “Auto-defined by” followed by 29 blanks, the queue manager name is added. If the final 20 characters are still all blanks the local time and date are added.

Once the definition has been created and stored the channel start proceeds as though the definition had always existed. The batch size, transmission size, and message size are negotiated with the partner.

Defining other objects:

Before a message channel can be started, both ends must be defined (or enabled for auto-definition) at their respective queue managers. The transmission queue it is to serve must be defined to the queue manager at the sending end, and the communication link must be defined and available. In addition, it may be necessary for you to prepare other WebSphere MQ objects, such as remote queue definitions, queue manager alias definitions, and reply-to queue alias definitions, so as to implement the scenarios described in “Making your applications communicate” on page 15.

For information about MQI channels, see the WebSphere MQ Clients book.

Multiple message channels per transmission queue:

It is possible to define more than one channel per transmission queue, but only one of these channels can be active at any one time. This is recommended for the provision of alternative routes between queue managers for traffic balancing and link failure corrective action. A transmission queue cannot be used by another channel if the previous channel to use it terminated leaving a batch of messages in-doubt at the sending end. For more information, see “In-doubt channels” on page 65.

Starting a channel:

A channel can be caused to start transmitting messages in one of four ways. It can be:

- Started by an operator (not receiver, cluster-receiver or server-connection channels).
- Triggered from the transmission queue. This applies to sender channels and fully qualified server channels (those which specify a CONNAME) only. You will need to prepare the necessary objects for triggering channels.
- Started from an application program (not receiver, cluster-receiver or server-connection channels).
- Started remotely from the network by a sender, cluster-sender, requester, server, or client-connection channel. Receiver, cluster-receiver and possibly server and requester channel transmissions, are started this way; so are server-connection channels. The channels themselves must already be started (that is, enabled).

Note: Because a channel is ‘started’ it is not necessarily transmitting messages, but, rather, it is ‘enabled’ to start transmitting when one of the four events described above occurs. The enabling and disabling of a channel is achieved using the START and STOP operator commands.

Channel states

Figure 29 on page 56 shows the hierarchy of all possible channel states and the substates that apply to each of the channel states. The substates are shown inside boxes, below the states to which they apply. Figure 30 on page 57 shows the links between channel states. These apply to all types of message channel and server-connection channels.

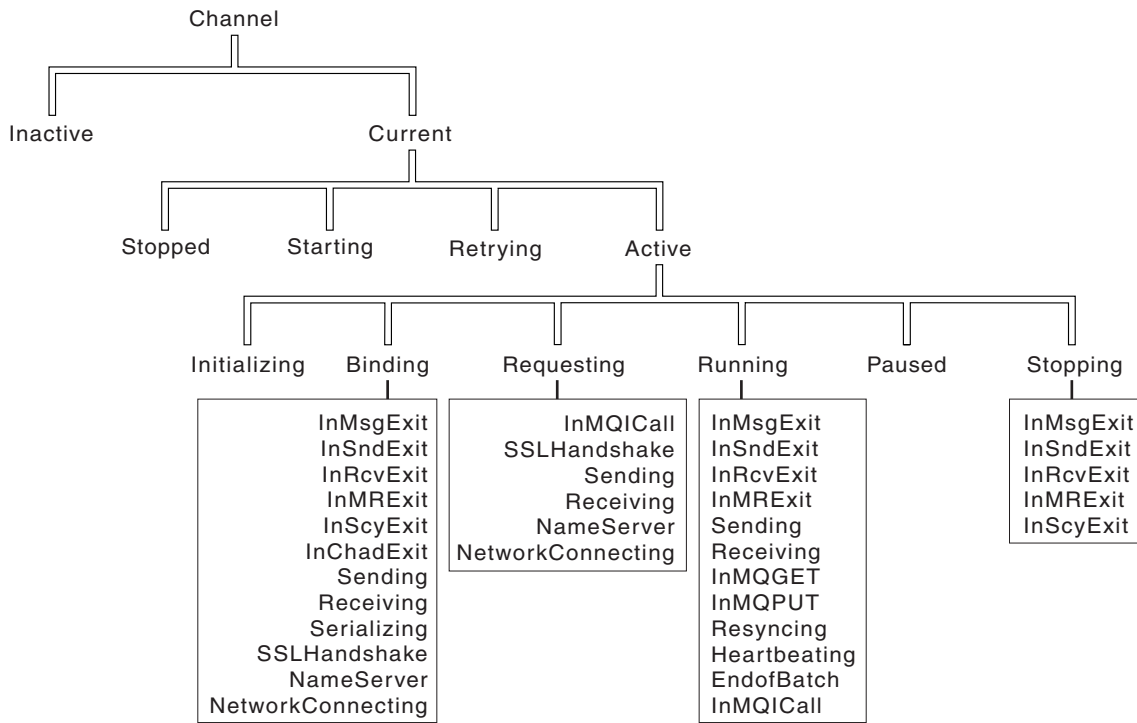


Figure 29. Channel states and substates

Current and active:

The channel is “current” if it is in any state other than inactive. A current channel is “active” unless it is in **RETRYING**, **STOPPED**, or **STARTING** state. If a channel is “active” it may also show a substate giving more detail of exactly what the channel is doing.

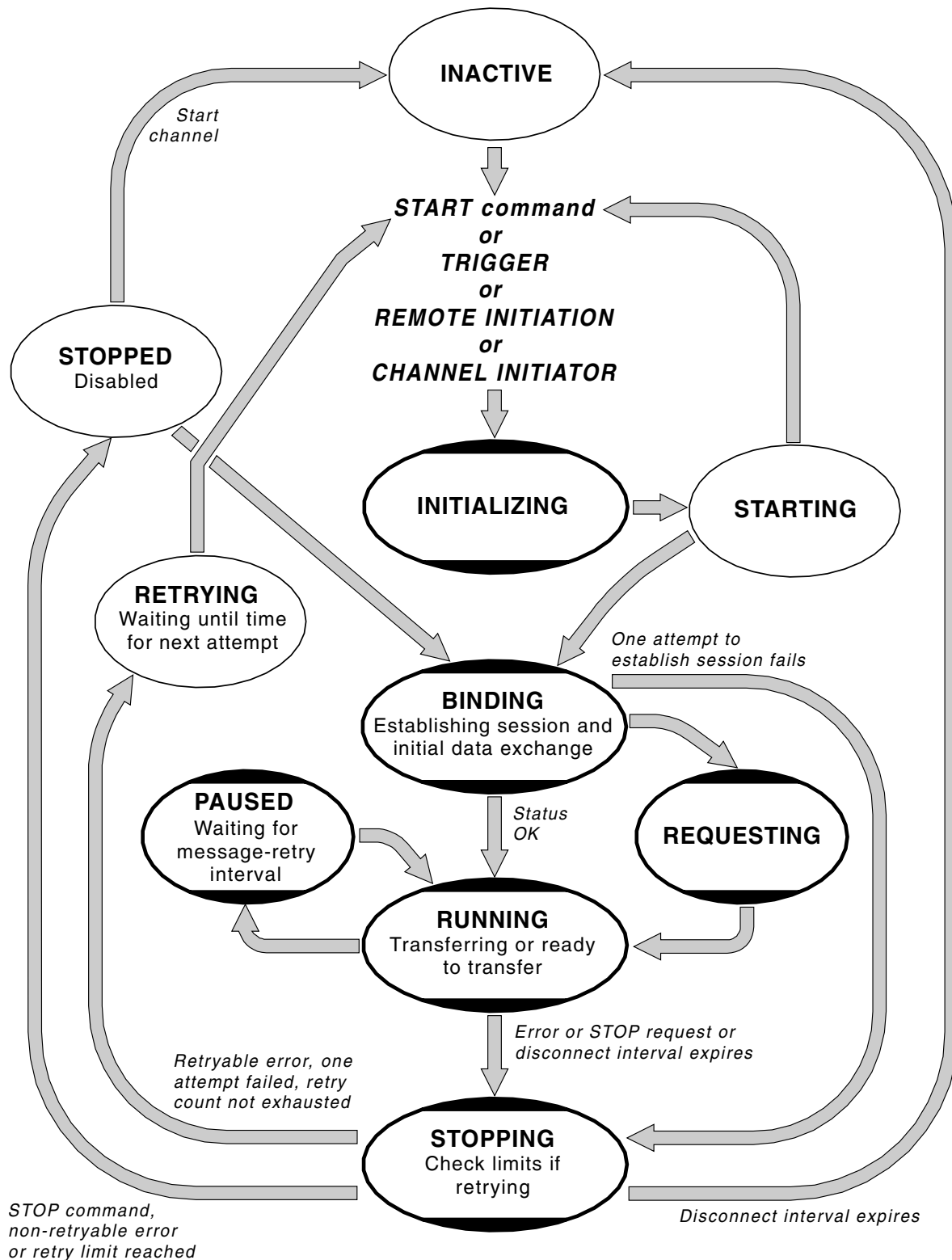


Figure 30. Flows between channel states

Note:

1. When a channel is in one of the six states highlighted in Figure 30 on page 57 (INITIALIZING, BINDING, REQUESTING, RUNNING, PAUSED, or STOPPING), it is consuming resource and a process or thread is running; the channel is *active*.
2. When a channel is in STOPPED state, the session may be active because the next state is not yet known.

Specifying the maximum number of current channels:

You can specify the maximum number of channels that can be current at one time. This is the number of channels that have entries in the channel status table, including channels that are retrying and channels that are stopped. Specify this using **ALTER QMGR MAXCHL** for z/OS, the queue manager initialization file for i5/OS, the queue manager configuration file for UNIX systems, or the WebSphere MQ Explorer. For more information about the values you set using the initialization or the configuration file see Chapter 8, “Configuration file stanzas for distributed queuing,” on page 491. For more information about specifying the maximum number of channels, see the WebSphere MQ System Administration Guide for WebSphere MQ for UNIX systems, and Windows systems, the WebSphere MQ for i5/OS System Administration Guide book for WebSphere MQ for i5/OS, or the WebSphere MQ Script (MQSC) Command Reference for WebSphere MQ for z/OS.

Note:

1. Server-connection channels are included in this number.
2. A channel must be current before it can become active. If a channel is started, but cannot become current, the start fails.

Specifying the maximum number of active channels:

You can also specify the maximum number of active channels. You can do this to prevent your system being overloaded by a large number of starting channels. If you use this method, you should set the disconnect interval attribute to a low value to allow waiting channels to start as soon as other channels terminate.

Each time a channel that is retrying attempts to establish connection with its partner, it must become an active channel. If the attempt fails, it remains a current channel that is not active, until it is time for the next attempt. The number of times that a channel will retry, and how often, is determined by the retry count and retry interval channel attributes. There are short and long values for both these attributes. See “Channel attributes” on page 71 for more information.

When a channel has to become an active channel (because a START command has been issued, or because it has been triggered, or because it is time for another retry attempt), but is unable to do so because the number of active channels is already at the maximum value, the channel waits until one of the active slots is freed by another channel instance ceasing to be active. If, however, a channel is starting because it is being initiated remotely, and there are no active slots available for it at that time, the remote initiation is rejected.

Whenever a channel, other than a requester channel, is attempting to become active, it goes into the STARTING state. This is true even if there is an active slot immediately available, although in this case it will only be in STARTING state for a very short time. However, if the channel has to wait for an active slot, it is in STARTING state while it is waiting.

Requester channels do not go into STARTING state. If a requester channel cannot start because the number of active channels is already at the limit, the channel ends abnormally.

Whenever a channel, other than a requester channel, is unable to get an active slot, and so waits for one, a message is written to the log or the z/OS console, and an event is generated. When a slot is subsequently freed and the channel is able to acquire it, another message and event are generated. Neither of these events and messages are generated if the channel is able to acquire a slot straightaway.

If a STOP CHANNEL command is issued while the channel is waiting to become active, the channel goes to STOPPED state. A Channel-Stopped event is raised as usual.

Server-connection channels are included in the maximum number of active channels.

For more information about specifying the maximum number of active channels, see the WebSphere MQ System Administration Guide for WebSphere MQ for UNIX systems, and Windows systems, the WebSphere MQ for i5/OS System Administration Guide for WebSphere MQ for i5/OS, or WebSphere MQ Script (MQSC) Command Reference for WebSphere MQ for z/OS.

Channel errors:

Errors on channels cause the channel to stop further transmissions. If the channel is a sender or server, it goes to RETRY state because it is possible that the problem may clear itself. If it cannot go to RETRY state, the channel goes to STOPPED state. For sending channels, the associated transmission queue is set to GET(DISABLED) and triggering is turned off. (A STOP command with STATUS(STOPPED) takes the side that issued it to STOPPED state; only expiry of the disconnect interval or a STOP command with STATUS(INACTIVE) will make it end normally and become inactive.) Channels that are in STOPPED state need operator intervention before they will restart (see “Restarting stopped channels” on page 64).

Note: For i5/OS, UNIX systems, and Windows systems, a channel initiator must be running for retry to be attempted. If the channel initiator is not available, the channel becomes inactive and must be manually restarted. If you are using a script to start the channel, ensure the channel initiator is running before you try to run the script.

“Long retry count (LONGRTY)” on page 86 describes how retrying works. If the error clears, the channel restarts automatically, and the transmission queue is re-enabled. If the retry limit is reached without the error clearing, the channel goes to STOPPED state. A stopped channel must be restarted manually by the operator. If the error is still present, it does not retry again. When it does start successfully, the transmission queue is re-enabled.

If the channel initiator (on z/OS) or queue manager (on platforms other than z/OS) stops while a channel is in RETRYING or STOPPED status, the channel status is remembered when the channel initiator or queue manager is restarted. However, the channel status for the SVRCONN channel type is reset if the channel initiator (on z/OS) or queue manager (on platforms other than z/OS) stops while the channel is in STOPPED status.

If a channel is unable to put a message to the target queue because that queue is full or put inhibited, the channel can retry the operation a number of times (specified in the message-retry count attribute) at a given time interval (specified in the message-retry interval attribute). Alternatively, you can write your own message-retry exit that determines which circumstances cause a retry, and the number of attempts made. The channel goes to PAUSED state while waiting for the message-retry interval to finish.

See “Channel attributes” on page 71 for information about the channel attributes, and “Channel-exit programs” on page 375 for information about the message-retry exit.

Server-connection channel limits

You can set server-connection channel limits to prevent client applications from exhausting queue manager channel resources and to prevent a single client application from exhausting server-connection channel capacity.

A maximum total number of channels can be active at any time on an individual queue manager, and the total number of server-connection channel instances are included in the maximum number of active channels.

If you do not specify the maximum number of simultaneous instances of a server-connection channel that can be started, then it is possible for a single client application, connecting to a single server-connection channel, to exhaust the maximum number of active channels that are available. When the maximum number of active channels is reached, it prevents any other channels from being started on the queue manager. To avoid this, you must limit the number of simultaneous instances of an individual server-connection channel that can be started, regardless of which client started them.

If the value of the limit is reduced to below the currently running number of instances of the server connection channel, even to zero, then the running channels are not affected. However, new instances cannot be started until sufficient existing instances have ceased to run so that the number of currently running instances is less than the value of the limit.

Also, many different client-connection channels can connect to an individual server-connection channel. The limit on the number of simultaneous instances of an individual server-connection channel that can be started, regardless of which client started them, prevents any client from exhausting the maximum active channel capacity of the queue manager. However, if you do not also limit the number of simultaneous instances of an individual server-connection channel that can be started from an individual client, then it is possible for a single, faulty client application to open so many connections that it exhausts the channel capacity allocated to an individual server-connection channel, and this prevents other clients that need to use the channel from connecting to it. To avoid this, you must limit the number of simultaneous instances of an individual server-connection channel that can be started from an individual client.

If the value of the individual client limit is reduced below the number of instances of the server-connection channel that are currently running from individual clients, even to zero, then the running channels are not affected. However, new instances of the server-connection channel cannot be started from an individual client that exceeds the new limit until sufficient existing instances from that client have ceased to run so that the number of currently running instances is less than the value of this parameter.

Checking that the other end of the channel is still available

Heartbeats:

You can use the heartbeat-interval channel attribute to specify that flows are to be passed from the sending MCA when there are no messages on the transmission queue. This is described in “Heartbeat interval (HBINT)” on page 84.

Keep Alive:

In WebSphere MQ for z/OS, if you are using TCP/IP as the transport protocol, you can also specify a value for the KeepAlive Interval channel attribute (KAINT). You are recommended to give the KeepAliveInterval a higher value than the heartbeat interval, and a smaller value than the disconnect value. You can use this attribute to specify a time-out value for each channel. This is described in “KeepAlive Interval (KAINT)” on page 84.

In WebSphere MQ for i5/OS, UNIX systems, and Windows systems, if you are using TCP as your transport protocol, you can set `keepalive=yes`. If you specify this option, TCP periodically checks that the other end of the connection is still available, and if it is not, the channel is terminated. This is described in “KeepAlive Interval (KAINT)” on page 84.

If you have unreliable channels that are suffering from TCP errors, use of `KEEPALIVE` will mean that your channels are more likely to recover

You can specify time intervals to control the behavior of the `KEEPALIVE` option. When you change the time interval, only TCP/IP channels started after the change are affected. The value that you choose for the time interval should be less than the value of the disconnect interval for the channel.

For more information about using the `KEEPALIVE` option on z/OS, see WebSphere MQ for z/OS Concepts and Planning Guide . For other platforms, see the chapter about setting up communications for your platform in this manual.

Receive Time Out:

If you are using TCP as your transport protocol, the receiving end of an idle non-MQI channel connection is also closed if no data is received for a period of time. This period of time is determined according to the HBINT (heartbeat interval) value.

In WebSphere MQ for i5/OS, UNIX systems, and Windows systems, the time-out value is set as follows:

1. For an initial number of flows, before any negotiation has taken place, the timeout is twice the HBINT value from the channel definition.
2. When the channels have negotiated a HBINT value, if HBINT is set to less than 60 seconds, the timeout is set to twice this value. If HBINT is set to 60 seconds or more, the timeout value is set to 60 seconds greater than the value of HBINT.

In WebSphere MQ for z/OS, the time-out value is set as follows:

1. For an initial number of flows, before any negotiation has taken place, the timeout is twice the HBINT value from the channel definition.
2. If `RCVTIME` is set, the timeout is set to one of

- the negotiated HBINT multiplied by a constant
- the negotiated HBINT plus a constant number of seconds
- a constant number of seconds

depending on the RCVTTYPE parameter, and subject to any limit imposed by RCVTMIN. If you use a constant value of RCVTIME and you use a heartbeat interval, do not specify an RCVTIME less than the heartbeat interval. For details of the RCVTIME, RCVTMIN and RCVTTYPE attributes, see ALTER QMGR in WebSphere MQ Script (MQSC) Command Reference.

Note:

1. If either of the above values is zero, there is no timeout.
2. For connections that do not support heartbeats, the HBINT value is negotiated to zero in step 2 and hence there is no timeout, so you must use TCP/IP KEEPALIVE.
3. For client connections, heartbeats are flowed from the server only when the client issues an MQGET call with wait; none are flowed during other MQI calls. Therefore, you are not recommended to set the heartbeat interval too small for client channels. For example, if the heartbeat is set to ten seconds, an MQCMIT call will fail (with MQRC_CONNECTION_BROKEN) if it takes longer than twenty seconds to commit because no data will have been flowed during this time. This can happen with large units of work. However, it should not happen if appropriate values are chosen for the heartbeat interval because only MQGET with wait should take significant periods of time.
4. Aborting the connection after twice the heartbeat interval is valid because a data or heartbeat flow is expected at least every heartbeat interval. Setting the heartbeat interval too small, however, can cause problems, especially if you are using channel exits. For example, if the HBINT value is one second, and a send or receive exit is used, the receiving end will only wait for two seconds before aborting the channel. If the MCA is performing a task such as encrypting the message, this value might be too short.

Adopting an MCA

If a channel suffers a communications failure, the receiver channel could be left in a 'communications receive' state. When communications are re-established the sender channel attempts to reconnect. If the remote queue manager finds that the receiver channel is already running it does not allow another version of the same receiver channel to be started. This problem requires user intervention to rectify the problem or the use of system keepalive.

The Adopt MCA function solves the problem automatically. It enables WebSphere MQ to cancel a receiver channel and to start a new one in its place.

The function can be set up with various options. For more information see WebSphere MQ Script (MQSC) Command Reference for z/OS, WebSphere MQ for i5/OS System Administration Guide for i5/OS and WebSphere MQ System Administration Guide for other platforms.

Stopping and quiescing channels

Message channels are designed to be long-running connections between queue managers with orderly termination controlled only by the disconnect interval

channel attribute. This mechanism works well unless the operator needs to terminate the channel before the disconnect time interval expires. This can occur in the following situations:

- System quiesce
- Resource conservation
- Unilateral action at one end of a channel

In this case, you can stop the channel. You can do this using:

- the STOP CHANNEL MQSC command
- the Stop Channel PCF command
- the WebSphere MQ Explorer
- other platform-specific mechanisms, as follows:

For z/OS:

The Stop a channel panel

For i5/OS:

The ENDMQMCHL CL command or the END option on the WRKMQMCHL panel

There are three options for stopping channels using these commands:

QUIESCE

The QUIESCE option attempts to end the current batch of messages before stopping the channel.

FORCE

The FORCE option attempts to stop the channel immediately and may require the channel to resynchronize when it restarts because the channel may be left in doubt.

TERMINATE

The TERMINATE option attempts to stop the channel immediately, and terminates the channel's thread or process.

Note that all of these options leave the channel in a STOPPED state, requiring operator intervention to restart it.

Stopping the channel at the sending end is quite effective but does require operator intervention to restart. At the receiving end of the channel, things are much more difficult because the MCA is waiting for data from the sending side, and there is no way to initiate an *orderly* termination of the channel from the receiving side; the stop command is pending until the MCA returns from its wait for data.

Consequently there are three recommended ways of using channels, depending upon the operational characteristics required:

- If you want your channels to be long running, you should note that there can be orderly termination only from the sending end. When channels are interrupted, that is, stopped, operator intervention (a START CHANNEL command) is required in order to restart them.
- If you want your channels to be active only when there are messages for them to transmit, you should set the disconnect interval to a fairly low value. Note that the default setting is quite high and so is not recommended for channels where this level of control is required. Because it is difficult to interrupt the receiving channel, the most economical option is to have the channel

automatically disconnect and reconnect as the workload demands. For most channels, the appropriate setting of the disconnect interval can be established heuristically.

- You can use the heartbeat-interval attribute to cause the sending MCA to send a heartbeat flow to the receiving MCA during periods in which it has no messages to send. This releases the receiving MCA from its wait state and gives it an opportunity to quiesce the channel without waiting for the disconnect interval to expire. Give the heartbeat interval a lower value than the value of the disconnect interval.

Note:

1. You are advised to set the disconnect interval to a low value, or to use heartbeats, for server channels. This is to allow for the case where the requester channel ends abnormally (for example, because the channel was canceled) when there are no messages for the server channel to send. If the disconnect interval is set high and heartbeats are not in use, the server does not detect that the requester has ended (it will only do this the next time it tries to send a message to the requester). While the server is still running, it holds the transmission queue open for exclusive input in order to get any more messages that arrive on the queue. If an attempt is made to restart the channel from the requester, the start request receives an error because the server still has the transmission queue open for exclusive input. It is necessary to stop the server channel, and then restart the channel from the requester again.

Restarting stopped channels

When a channel goes into STOPPED state (either because you have stopped the channel manually using one of the methods given in “Stopping and quiescing channels” on page 62, or because of a channel error) you have to restart the channel manually.

To do this, issue one of the following commands:

- The START CHANNEL MQSC command
- The Start Channel PCF command
- the WebSphere MQ Explorer
-

other platform-specific mechanisms, as follows:

For z/OS:

The Start a channel panel

For i5/OS:

The STRMQMCHL CL command or the START option on the WRKMQMCHL panel

For sender or server channels, when the channel entered the STOPPED state, the associated transmission queue was set to GET(DISABLED) and triggering was set off. When the start request is received, these attributes are reset automatically.

If the channel initiator (on z/OS) or queue manager (on distributed platforms) stops while a channel is in RETRYING or STOPPED status, the channel status is remembered when the channel initiator or queue manager is restarted. However, the channel status for the SVRCONN channel type is reset if the channel initiator or queue manager stops while the channel is in STOPPED status.

In-doubt channels

An in-doubt channel is a channel that is in doubt with a remote channel about which messages have been sent and received. Note the distinction between this and a queue manager being in doubt about which messages should be committed to a queue.

You can reduce the opportunity for a channel to be placed in doubt by using the Batch Heartbeat channel parameter (BATCHHB). When a value for this parameter is specified, a sender channel checks that the remote channel is still active before taking any further action. If no response is received the receiver channel is considered to be no longer active. The messages can be rolled-back, and re-routed, and the sender-channel is not put in doubt. This reduces the time when the channel could be placed in doubt to the period between the sender channel verifying that the receiver channel is still active, and verifying that the receiver channel has received the sent messages. See “Channel attributes” on page 71 for more information on the batch heartbeat parameter.

In-doubt channel problems are usually resolved automatically. Even when communication is lost, and a channel is placed in doubt with a message batch at the sender whose receipt status is unknown, the situation is resolved when communication is re-established. Sequence number and LUWID records are kept for this purpose. The channel is in doubt until LUWID information has been exchanged, and only one batch of messages can be in doubt for the channel.

You can, when necessary, resynchronize the channel manually. The term *manual* includes use of operators or programs that contain WebSphere MQ system management commands. The manual resynchronization process works as follows. This description uses MQSC commands, but you can also use the PCF equivalents.

1. Use the DISPLAY CHSTATUS command to find the last-committed logical unit of work ID (LUWID) for *each* side of the channel. Do this using the following commands:

- For the in-doubt side of the channel:

```
DISPLAY CHSTATUS(name) SAVED CURLUWID
```

You can use the CONNAME and XMITQ parameters to further identify the channel.

- For the receiving side of the channel:

```
DISPLAY CHSTATUS(name) SAVED LSTLUWID
```

You can use the CONNAME parameter to further identify the channel.

The commands are different because only the sending side of the channel can be in doubt. The receiving side is never in doubt.

On WebSphere MQ for i5/OS, the DISPLAY CHSTATUS command can be executed from a file using the STRMQMMQSC command or the Work with MQM Channel Status CL command, WRKMQMCHST

2. If the two LUWIDs are the same, the receiving side has committed the unit of work that the sender considers to be in doubt. The sending side can now remove the in-doubt messages from the transmission queue and re-enable it. This is done with the following channel RESOLVE command:

```
RESOLVE CHANNEL(name) ACTION(COMMIT)
```

3. If the two LUWIDs are different, the receiving side has not committed the unit of work that the sender considers to be in doubt. The sending side needs to retain the in-doubt messages on the transmission queue and re-send them. This is done with the following channel RESOLVE command:

RESOLVE CHANNEL(*name*) ACTION(BACKOUT)

On WebSphere MQ for i5/OS, you can use the Resolve MQM Channel command, RSVMQMCHL.

Once this process is complete the channel is no longer in doubt. The transmission queue can now be used by another channel, if required.

Problem determination

There are two distinct aspects to problem determination:

- Problems discovered when a command is being submitted
- Problems discovered during operation of the channels

Command validation:

Commands and panel data must be free from errors before they are accepted for processing. Any errors found by the validation are immediately notified to the user by error messages.

Problem diagnosis begins with the interpretation of these error messages and taking the recommended corrective action.

Processing problems:

Problems found during normal operation of the channels are notified to the system console or the system log. Problem diagnosis begins with the collection of all relevant information from the log, and continues with analysis to identify the problem.

Confirmation and error messages are returned to the terminal that initiated the commands, when possible.

WebSphere MQ produces accounting and statistical data, which you can use to identify trends in utilization and performance. On z/OS, this information is produced in the form of SMF records, see *WebSphere MQ for z/OS System Setup Guide* for details. The equivalent information on other platforms is produced as PCF records, see *Monitoring WebSphere MQ* for details.

Messages and codes:

Where provided, the *Messages and Codes* manual of the particular platform can help with the primary diagnosis of the problem.

What happens when a message cannot be delivered?

Figure 31 on page 67 shows the processing that occurs when an MCA is unable to put a message to the destination queue. (Note that the options shown do not apply on all platforms.)

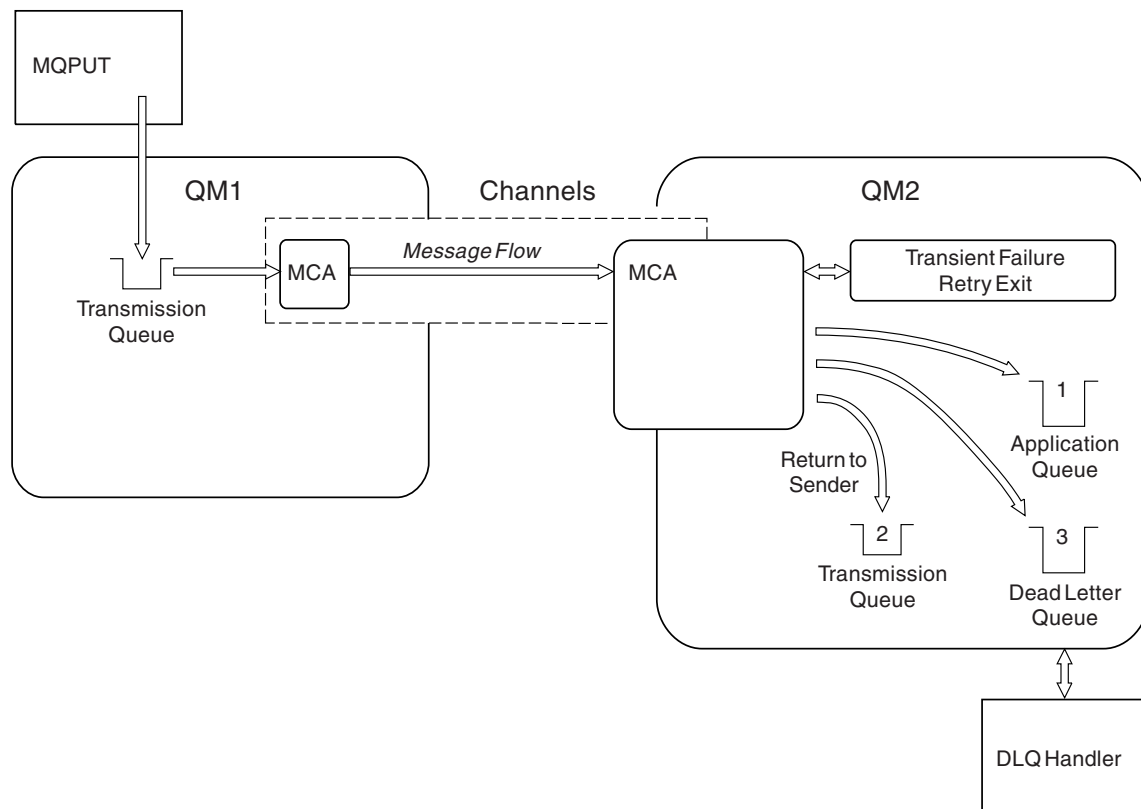


Figure 31. What happens when a message cannot be delivered

As shown in the figure, the MCA can do several things with a message that it cannot deliver. The action taken is determined by options specified when the channel is defined and on the MQPUT report options for the message.

1. Message-retry

If the MCA is unable to put a message to the target queue for a reason that could be transitory (for example, because the queue is full), the MCA has the option to wait and retry the operation later. You can determine if the MCA waits, for how long, and how many times it retries.

- You can specify a message-retry time and interval for MQPUT errors when you define your channel. If the message cannot be put to the destination queue because the queue is full, or is inhibited for puts, the MCA retries the operation the number of times specified, at the time interval specified.
- You can write your own message-retry exit. The exit enables you to specify under what conditions you want the MCA to retry the MQPUT or MQOPEN operation. Specify the name of the exit when you define the channel.

2. Return-to-sender

If message-retry was unsuccessful, or a different type of error was encountered, the MCA can send the message back to the originator.

To enable this, you need to specify the following options in the message descriptor when you put the message to the original queue:

- The MQRO_EXCEPTION_WITH_FULL_DATA report option
- The MQRO_DISCARD_MSG report option
- The name of the reply-to queue and reply-to queue manager

If the MCA is unable to put the message to the destination queue, it generates an exception report containing the original message, and puts it on a transmission queue to be sent to the reply-to queue specified in the original message. (If the reply-to queue is on the same queue manager as the MCA, the message is put directly to that queue, not to a transmission queue.)

3. Dead-letter queue

If a message cannot be delivered or returned, it is put on to the dead-letter queue (DLQ). You can use the DLQ handler to process the message. This is described in the WebSphere MQ System Administration Guide for WebSphere MQ for UNIX and Windows systems, and in the WebSphere MQ for z/OS System Administration Guide for z/OS.

If the dead-letter queue is not available, the sending MCA leaves the message on the transmission queue, and the channel stops. On a fast channel, nonpersistent messages that cannot be written to a dead-letter queue are lost.

On WebSphere MQ Version 5 and later, if no local dead-letter queue is defined, the remote queue is not available or defined, and there is no remote dead-letter queue, the channel stops abnormally, and messages are not rolled back to the sending transmission queue. You must resolve the channel using the COMMIT or BACKOUT functions.

Initialization and configuration files

The handling of channel initialization data depends on your WebSphere MQ platform.

z/OS

In WebSphere MQ for z/OS, initialization and configuration information is specified using the ALTER QMGR MQSC command. If you put ALTER QMGR commands in the CSQINP2 initialization input data set, they are processed every time the queue manager is started. To run MQSC commands such as START LISTENER every time you start the channel initiator, put them in the CSQINPX initialization input data set and specify the optional DD statement CSQINPX in the channel initiator started task procedure. See WebSphere MQ for z/OS Concepts and Planning Guide for information about CSQINP2 and CSQINPX, and WebSphere MQ Script (MQSC) Command Reference for information about ALTER QMGR.

Windows systems

On WebSphere MQ for Windows systems, the *registry* file holds basic configuration information about the WebSphere MQ. That is, information relevant to all of the queue managers on the WebSphere MQ system and also information relating to individual queue managers. You can examine or change this information using the WebSphere MQ Explorer. Do not edit the registry directly.

i5/OS and UNIX systems

In WebSphere MQ for i5/OS and WebSphere MQ on UNIX systems, there are *configuration files* to hold basic configuration information about the WebSphere MQ installation.

There are two configuration files: one applies to the machine, the other applies to an individual queue manager.

WebSphere MQ configuration file:

This holds information relevant to all of the queue managers on the WebSphere MQ system. The file is called `mqs.ini`. It is fully described in the WebSphere MQ System Administration Guide for WebSphere MQ for UNIX systems, and in the WebSphere MQ for i5/OS System Administration Guide for WebSphere MQ for i5/OS.

Queue manager configuration file:

The queue manager configuration file holds configuration information relating to one particular queue manager. The file is called `qm.ini`.

It is created during queue manager creation and may hold configuration information relevant to any aspect of the queue manager. Information held in the file includes details of how the configuration of the log differs from the default in WebSphere MQ configuration file.

The queue manager configuration file is held in the root of the directory tree occupied by the queue manager. On WebSphere MQ for Windows, the information is held in the registry. For example, for the `DefaultPath` attributes, the queue manager configuration files for a queue manager called `QMNAME` would be:

For UNIX systems:

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

An excerpt of a `qm.ini` file follows. It specifies that the TCP/IP listener is to listen on port 2500, the maximum number of current channels is to be 200 and the maximum number of active channels is to be 100.

```
TCP:
  Port=2500
CHANNELS:
  MaxChannels=200
  MaxActiveChannels=100
```

In MQSeries V5.2 and WebSphere MQ, you can specify a range of TCP/IP ports to be used by an outbound channel. One method is to use the `qm.ini` file to specify the start and end of a range of port values. The example below shows a `qm.ini` file specifying a range of channels:

```
TCP:
  StrPort=2500
  EndPort=3000
CHANNELS:
  MaxChannels=200
  MaxActiveChannels=100
```

If you specify a value for `StrPort` or `EndPort` then you must specify a value for both. The value of `EndPort` must always be greater than the value of `StrPort`.

The channel tries to use each of the port values in the range specified. When the connection is successful, the port value is the port that the channel then uses.

For i5/OS:

```
/QIBM/UserData/mqm/qmgrs/QMNAME/qm.ini
```

For more information about qm.ini files see Chapter 8, “Configuration file stanzas for distributed queuing,” on page 491.

Data conversion

A WebSphere MQ message consists of two parts:

- Control information in a message descriptor
- Application data

Either of the two parts may require data conversion when sent between queues on different queue managers. For information about data conversion, see the WebSphere MQ Application Programming Guide.

Writing your own message channel agents

WebSphere MQ allows you to write your own message channel agent (MCA) programs or to install one from an independent software vendor. You might want to do this to make WebSphere MQ interoperate over your own, proprietary communications protocol or to send messages over a protocol that WebSphere MQ does not support. (You cannot write your own MCA to interoperate with a WebSphere MQ-supplied MCA at the other end.)

If you decide to use an MCA that was not supplied by WebSphere MQ, you need to consider the following.

Message sending and receiving

You need to write a sending application that gets messages from wherever your application puts them, for example from a transmission queue (see the WebSphere MQ Application Programming Reference book), and sends them out on a protocol with which you want to communicate. You also need to write a receiving application that takes messages from this protocol and puts them onto destination queues. The sending and receiving applications use the message queue interface (MQI) calls, not any special interfaces.

You need to ensure that messages are delivered once and once only. Syncpoint coordination can be used to help with this.

Channel control function

You need to provide your own administration functions to control channels. You cannot use WebSphere MQ channel administration functions either for configuring (for example, the DEFINE CHANNEL command) or monitoring (for example, DISPLAY CHSTATUS) your channels.

Initialization file

You need to provide your own initialization file, if you require one.

Application data conversion

You will probably want to allow for data conversion for messages you send to a different system. If so, use the MQGMO_CONVERT option on the MQGET call when retrieving messages from wherever your application puts them, for example the transmission queue.

User exits

Consider whether you need user exits. If so, you can use the same interface definitions that WebSphere MQ uses.

Triggering

If your application puts messages to a transmission queue, you can set up the transmission queue attributes so that your sending MCA is triggered when messages arrive on the queue.

Channel initiator

You may need to provide your own channel initiator.

Channel attributes

The previous chapters have introduced the basic concepts of the product, the business perspective basis of its design, its implementation, and the control features.

This chapter describes the channel attributes held in the channel definitions. This is product-sensitive programming interface information.

You choose the attributes of a channel to be optimal for a given set of circumstances for each channel. However, when the channel is running, the actual values may have changed during startup negotiations. See “Preparing channels” on page 54.

Many attributes have default values, and you can use these for most channels. However, in those circumstances where the defaults are not optimal, refer to this chapter for guidance in selecting the correct values.

Note: In WebSphere MQ for i5/OS, most attributes can be specified as *SYSDFTCHL, which means that the value is taken from the system default channel in your system.

Channel attributes and channel types

Different types of channel support different channel attributes.

The channel types for WebSphere MQ channel attributes are listed in Table 7.

Table 7. Channel attributes for the channel types

Attribute field	SDR	SVR	RCVR	RQSTR	CLNT-CONN	SVR-CONN	CLUS-SDR	CLUS-RCVR
Alter date	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Alter time	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Batch heartbeat interval	Yes	Yes					Yes	Yes
Batch interval	Yes	Yes					Yes	Yes
Batch size	Yes	Yes	Yes	Yes			Yes	Yes
Channel name	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Channel statistics	Yes	Yes	Yes	Yes			Yes	Yes
Channel type	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Client channel weight					Yes			
Cluster							Yes	Yes
Cluster namelist							Yes	Yes

Table 7. Channel attributes for the channel types (continued)

Attribute field	SDR	SVR	RCVR	RQSTR	CLNT-CONN	SVR-CONN	CLUS-SDR	CLUS-RCVR
Cluster workload priority							Yes	Yes
Cluster workload rank							Yes	Yes
Cluster workload weight							Yes	Yes
Connection affinity					Yes			
Connection name	Yes	Yes		Yes	Yes		Yes	Yes
Convert message	Yes	Yes					Yes	Yes
Data compression	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Description	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Disconnect interval	Yes	Yes				Yes ¹	Yes	Yes
Disposition	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Header compression	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Heartbeat interval	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Keepalive interval	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Local address	Yes	Yes		Yes	Yes		Yes	Yes
Long retry count	Yes	Yes					Yes	Yes
Long retry interval	Yes	Yes					Yes	Yes
LU 6.2 mode name	Yes	Yes		Yes	Yes		Yes	Yes
LU 6.2 transaction program name	Yes	Yes		Yes	Yes		Yes	Yes
Maximum instances						Yes		
Maximum instances per client						Yes		
Maximum message length	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Message channel agent name								
Message channel agent type	Yes	Yes		Yes			Yes	Yes
Message channel agent user	Yes	Yes	Yes	Yes		Yes	Yes	Yes
Message exit name	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Message exit user data	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Message-retry exit name			Yes	Yes				Yes
Message-retry exit user data			Yes	Yes				Yes
Message retry count			Yes	Yes				Yes
Message retry interval			Yes	Yes				Yes
Monitoring	Yes	Yes	Yes	Yes		Yes	Yes	Yes

Table 7. Channel attributes for the channel types (continued)

Attribute field	SDR	SVR	RCVR	RQSTR	CLNT-CONN	SVR-CONN	CLUS-SDR	CLUS-RCVR
Network-connection priority								Yes
Nonpersistent message speed	Yes	Yes	Yes	Yes			Yes	Yes
Password	Yes	Yes		Yes	Yes		Yes	
Property control	Yes	Yes					Yes	Yes
PUT authority			Yes	Yes		Yes ¹		Yes
Queue manager name					Yes			
Receive exit	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Receive exit user data	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Security exit	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Security exit user data	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Send exit	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Send exit user data	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Sequence number wrap	Yes	Yes	Yes	Yes			Yes	Yes
Short retry count	Yes	Yes					Yes	Yes
Short retry interval	Yes	Yes					Yes	Yes
SSL Cipher Specification	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
SSL Client Authentication		Yes	Yes	Yes		Yes		Yes
SSL Peer	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Transmission queue	Yes	Yes						
Transport type	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
User ID	Yes	Yes		Yes	Yes		Yes	
Note:								
1. Valid on z/OS only.								

Channel attributes in alphabetical order

This topic describes each attribute of a channel object, with its valid values and notes on its use where appropriate.

WebSphere MQ for some platforms might not implement all the attributes shown in the list. Exceptions and platform differences are mentioned in the individual attribute descriptions, where relevant.

The keyword that you can specify in MQSC is shown in brackets for each attribute.

The attributes are arranged in alphabetical order, as follows:

Alter date (ALTDATE)

This is the date on which the definition was last altered, in the form yyyy-mm-dd.

This attribute is valid for all channel types.

Alter time (ALTTIME)

This is the time at which the definition was last altered, in the form hh:mm:ss.

This attribute is valid for all channel types.

Batch Heartbeat Interval (BATCHEHB)

The batch heartbeat interval allows a sending channel to verify that the receiving channel is still active just before committing a batch of messages, so that if the receiving channel is not active, the batch can be backed out rather than becoming in-doubt, as would otherwise be the case. By backing out the batch, the messages remain available for processing so they could, for example, be redirected to another channel.

If the sending channel has had a communication from the receiving channel within the batch heartbeat interval, the receiving channel is assumed to be still active, otherwise a 'heartbeat' is sent to the receiving channel to check.

The value must be in the range zero through 999 999. A value of zero indicates that batch heartbeating is not used.

This attribute is valid for channel types of:

- Sender
- Server
- Cluster sender
- Cluster receiver

Batch interval (BATCHINT)

The batch interval is a period of time, in milliseconds, during which the channel will keep a batch open even if there are no messages on the transmission queue. You can specify any number of milliseconds, from zero through 999 999 999. The default value is zero.

If you do not specify a batch interval, the batch closes when the number of messages specified in BATCHSZ has been sent or when the transmission queue becomes empty. On lightly loaded channels, where the transmission queue frequently becomes empty the effective batch size may be much smaller than BATCHSZ.

You can use the BATCHINT attribute to make your channels more efficient by reducing the number of short batches. Be aware, however, that you may slow down the response time, because batches will last longer and messages will remain uncommitted for longer.

If you specify a BATCHINT, batches close only when one of the following conditions is met:

- The number of messages specified in BATCHSZ have been sent.

- There are no more messages on the transmission queue and a time interval of BATCHINT has elapsed while waiting for messages (since the first message of the batch was retrieved).

Note: BATCHINT specifies the total amount of time that is spent waiting for messages. It does not include the time spent retrieving messages that are already available on the transmission queue, or the time spent transferring messages.

This attribute is valid for channel types of:

- Sender
- Server
- Cluster sender
- Cluster receiver

Batch size (BATCHSZ)

The batch size is the maximum number of messages to be sent before a syncpoint is taken. The batch size does not affect the way the channel transfers messages; messages are always transferred individually, but are committed or backed out as a batch.

To improve performance, you can set a batch size to define the maximum number of messages to be transferred between two *syncpoints*. The batch size to be used is negotiated when a channel starts up, and the lower of the two channel definitions is taken. On some implementations, the batch size is calculated from the lowest of the two channel definitions and the two queue manager MAXUMSGS values. The actual size of a batch can be less than this; for example, a batch completes when there are no messages left on the transmission queue or the batch interval expires.

A large value for the batch size increases throughput, but recovery times are increased because there are more messages to back out and re-send. The default BATCHSZ is 50, and you are advised to try that value first. You might choose a lower value for BATCHSZ if your communications are unreliable, making the need to recover more likely.

Syncpoint procedure needs a unique logical unit of work identifier to be exchanged across the link every time a syncpoint is taken, to coordinate batch commit procedures.

If the synchronized batch commit procedure is interrupted, an *in-doubt* situation may arise. In-doubt situations are resolved automatically when a message channel starts up. If this resolution is not successful, manual intervention may be necessary, making use of the RESOLVE command.

Some considerations when choosing the number for batch size:

- If the number is too large, the amount of queue space taken up on both ends of the link becomes excessive. Messages take up queue space when they are not committed, and cannot be removed from queues until they are committed.
- If there is likely to be a steady flow of messages, you can improve the performance of a channel by increasing the batch size. However, this has the negative effect of increasing restart times, and very large batches may also affect performance.

- If message flow characteristics indicate that messages arrive intermittently, a batch size of 1 with a relatively large disconnect time interval may provide a better performance.
- The number may be in the range 1 through 9999. However, for data integrity reasons, channels connecting to any of the current platforms, as described in this book, should specify a batch size greater than 1. (A value of 1 is for use with Version 1 products, apart from MQSeries for MVS/ESA™.)
- Even though nonpersistent messages on a fast channel do not wait for a syncpoint, they do contribute to the batch-size count.

This attribute is valid for channel types of:

- Sender
- Server
- Receiver
- Requester
- Cluster sender
- Cluster receiver

Channel name (CHANNEL)

Specifies the name of the channel definition. The name can contain up to 20 characters, although as both ends of a message channel must have the same name, and other implementations may have restrictions on the size, the actual number of characters may have to be smaller.

Where possible, channel names should be unique to one channel between any two queue managers in a network of interconnected queue managers.

The name must contain characters from the following list:

Alphabetic	(A-Z, a-z; note that uppercase and lowercase are significant)
Numerics	(0-9)
Period	(.)
Forward slash	(/)
Underscore	(_)
Percentage sign	(%)

Note:

1. Embedded blanks are not allowed, and leading blanks are ignored.
2. On systems using EBCDIC Katakana, you cannot use lowercase characters.

This attribute is valid for all channel types.

Channel statistics (STATCHL)

Controls the collection of statistics data for channels.

The possible values are:

QMGR

Statistics data collection for this channel is based upon the setting of the queue manager attribute STATCHL. This is the default value.

OFF Statistics data collection for this channel is disabled.

LOW Statistics data collection for this channel is enabled with a low ratio of data collection.

MEDIUM

Statistics data collection for this channel is enabled with a moderate ratio of data collection.

HIGH Statistics data collection for this channel is enabled with a high ratio of data collection.

For more information on channel statistics, see *Monitoring WebSphere MQ*.

This attribute is not supported on z/OS.

This attribute is valid for channel types of:

- Sender
- Server
- Receiver
- Requester
- Cluster sender
- Cluster receiver

Channel type (CHLTYPE)

Specifies the type of the channel being defined. The possible channel types are:

Message channel types:

- Sender
- Server
- Receiver
- Requester
- Cluster-sender
- Cluster-receiver

MQI channel types:

- Client-connection (WebSphere MQ for Windows systems, and UNIX systems only)

Note: Client-connection channels can also be defined on z/OS for use on other platforms.

- Server-connection

The two ends of a channel must have the same name and have compatible types:

- Sender with receiver
- Requester with server
- Requester with sender (for callback)
- Server with receiver (server is used as a sender)
- Client-connection with server-connection
- Cluster-sender with cluster-receiver

Client channel weight (CLNTWGHT)

Specifies a weighting to influence which client-connection channel definition is used.

The client channel weighting attribute is used so that client channel definitions can be selected at random based on their weighting when more than one suitable definition is available.

When a client issues an MQCONN requesting connection to a queue manager group, by specifying a queue manager name starting with an asterisk, which enables client weight balancing across several queue managers, and more than one suitable channel definition is available in the client channel definition table (CCDT), the definition to use is randomly selected based on the weighting, with any applicable CLNTWGHT(0) definitions selected first in alphabetical order.

Specify a value in the range 0 – 99. The default is 0.

A value of 0 indicates that no load balancing is performed and applicable definitions are selected in alphabetical order. To enable load balancing choose a value in the range 1 - 99 where 1 is the lowest weighting and 99 is the highest. The distribution of connections between two or more channels with non-zero weightings is approximately proportional to the ratio of those weightings. For example, three channels with CLNTWGHT values of 2, 4, and 14 are selected approximately 10%, 20%, and 70% of the time. This distribution is not guaranteed. If the AFFINITY attribute of the connection is set to PREFERRED, the first connection chooses a channel definition according to client weightings, and then subsequent connections will continue to use the same channel definition.

This attribute is valid for the client-connection channel type only.

Cluster (CLUSTER)

The name of the cluster to which the channel belongs. The maximum length is 48 characters conforming to the rules for naming WebSphere MQ objects.

Up to one of the resultant values of CLUSTER or CLUSNL can be non-blank. If one of the values is non-blank, the other must be blank.

This attribute is valid for channel types of:

- Cluster sender
- Cluster receiver

Cluster namelist (CLUSNL)

The name of the namelist that specifies a list of clusters to which the channel belongs.

Up to one of the resultant values of CLUSTER or CLUSNL can be nonblank. If one of the values is nonblank, the other must be blank.

This attribute is valid for channel types of:

- Cluster sender
- Cluster receiver

Cluster workload priority (CLWLPRTY)

Specifies the priority of the channel. The value must be in the range 0 through 9, where 0 is the lowest priority and 9 is the highest.

This attribute is valid for channel types of:

- Cluster sender
- Cluster receiver

Cluster workload rank (CLWLRANK)

Specifies the rank of the channel. The value must be in the range 0 through 9, where 0 is the lowest rank and 9 is the highest.

This attribute is valid for channel types of:

- Cluster sender
- Cluster receiver

Cluster workload weight (CLWLWGHT)

Applies a weighting factor to the channel so the proportion of messages sent down that channel can be controlled. The value must be in the range 1 through 99, where 1 is the lowest weighting and 99 is the highest.

This attribute is valid for channel types of:

- Cluster sender
- Cluster receiver

Connection affinity (AFFINITY)

Specifies which client channel definition that the client applications use to connect to the queue manager if multiple connections are available.

Use this attribute when multiple applicable channel definitions are available.

The possible values are:

PREFERRED

The first connection in a process reading a client channel definition table (CCDT) creates a list of applicable definitions based on the client channel weight, with any definitions having a weight of 0 first and in alphabetical order. Each connection in the process attempts to connect using the first definition in the list. If a connection is unsuccessful the next definition is used. Unsuccessful definitions with client channel weight values other than 0 are moved to the end of the list. Definitions with a client channel weight of 0 remain at the start of the list and are selected first for each connection.

Each client process with the same hostname always creates the same list.

For client applications written in C, C++, or the .NET programming framework (including fully managed .NET) the list is updated if the CCDT has been modified since the list was created.

This is the default value.

NONE

The first connection in a process reading a CCDT creates a list of applicable definitions. All connections in a process select an applicable definition based on the client channel weight, with any definitions having a weight of 0 selected first in alphabetical order.

For client applications written in C, C++, or the .NET programming framework (including fully managed .NET) the list is updated if the CCDT has been modified since the list was created.

This attribute is valid for the client-connection channel type only.

Connection name (CONNAME)

This is the communications connection identifier. It specifies the particular communications link to be used by this channel.

It is optional for server channels, unless the server channel is triggered, in which case it **MUST** specify a connection name.

The name is up to 48 characters for z/OS, 264 characters for other platforms, and:

If the transport type is TCP

This is either the hostname or the network address of the remote machine (or the local machine for cluster-receiver channels). For example, (MACH1.ABC.COM), (fe80:43e4:0204:acff:fe97:2c34:fde0:3485) or (19.22.11.162). It may include the port number, for example (MACHINE(123)). It can include the IP_name of a z/OS dynamic DNS group or a network dispatcher input port.

If you use an IPV6 address in a network that only supports IPV4, the connection name will not be resolved. In a network which uses both IPV4 and IPV6, Connection name interacts with Local Address to determine which IP stack is used. See “Local Address (LOCLADDR)” on page 85 for further information.

If the transport type is LU 6.2

For WebSphere MQ for i5/OS, Windows systems, and UNIX systems, give the fully-qualified name of the partner LU if the TPNAME and MODENAME are specified. For other versions or if the TPNAME and MODENAME are blank, give the CPI-C side information object name as described in the section in this book about setting up communication for your platform.

On z/OS there are two forms in which to specify the value:

- Logical unit name

The logical unit information for the queue manager, comprising the logical unit name, TP name, and optional mode name. This can be specified in one of 3 forms:

luname, for example IGY12355

luname/TPname, for example IGY12345/APING

luname/TPname/modename, for example IGY12345/APINGD/#INTER

For the first form, the TP name and mode name must be specified for the TPNAME and MODENAME attributes ; otherwise these attributes must be blank.

Note: For client-connection channels, only the first form is allowed.

- Symbolic name

The symbolic destination name for the logical unit information for the queue manager, as defined in the side information data set. The TPNAME and MODENAME attributes must be blank.

Note: For cluster-receiver channels, the side information is on the other queue managers in the cluster. Alternatively, in this case it can be a name that a channel auto-definition exit can resolve into the appropriate logical unit information for the local queue manager.

The specified or implied LU name can be that of a VTAM[®] generic resources group.

If the transmission protocol is NetBIOS

This is the NetBIOS name defined on the remote machine.

If the transmission protocol is SPX

This is an SPX-style address consisting of a 4-byte network address, a 6-byte node address and a 2-byte socket number. Enter these in hexadecimal, with the network and node addresses separated by a fullstop and the socket number in brackets. For example:

```
CONNNAME('0a0b0c0d.804abcde23a1(5e86)')
```

If the socket number is omitted, the default WebSphere MQ SPX socket number is used. The default is X'5E86'.

This attribute is valid for channel types of:

- Sender
- Server
- Requester
- Client connection
- Cluster sender
- Cluster receiver

It is optional for server channels, unless the server channel is triggered, in which case it **MUST** specify a connection name.

Note: The definition of transmission protocol is contained in “Transport type (TRPTYPE)” on page 100.

Convert message (CONVERT)

Application message data is usually converted by the receiving application. However, if the remote queue manager is on a platform that does not support data conversion, use this channel attribute to specify that the message should be converted into the format required by the receiving system *before* transmission.

The possible values are ‘yes’ and ‘no’. If you specify ‘yes’, the application data in the message is converted before sending if you have specified one of the built-in format names, or a data conversion exit is available for a user-defined format (See the WebSphere MQ Application Programming Guide). If you specify ‘no’, the application data in the message is not converted before sending.

This attribute is valid for channel types of:

- Sender
- Server
- Cluster sender
- Cluster receiver

Data compression (COMPMSG)

This is a list of message data compression techniques supported by the channel. For sender, server, cluster-sender, cluster-receiver, and client-connection channels the values specified are in order of preference. The first compression technique supported by the remote end of the channel is used. The channels’ mutually

supported compression techniques are passed to the sending channel's message exit where the compression technique used can be altered on a per message basis. Compression will alter the data passed to send and receive exits. See "Header compression (COMPHDR)" on page 84 for compression of the message header.

The possible values are:

NONE

No message data compression is performed. This is the default value.

RLE Message data compression is performed using run-length encoding.

ZLIBFAST

Message data compression is performed using the zlib compression technique. A fast compression time is preferred.

ZLIBHIGH

Message data compression is performed using the zlib compression technique. A high level of compression is preferred.

ANY Allows the channel to support any compression technique that the queue manager supports. Only supported for Receiver, Requester and Server-Connection channels.

This attribute is valid for all channel types.

Description (DESCR)

This contains up to 64 bytes of text that describes the channel definition.

Note: The maximum number of characters is reduced if the system is using a double byte character set (DBCS).

Use characters from the character set identified by the coded character set identifier (CCSID) for the queue manager to ensure that the text is translated correctly if it is sent to another queue manager.

This attribute is valid for all channel types.

Disconnect interval (DISCINT)

The disconnect interval is the length of time after which a channel will close down if no message arrives on the transmission queue during that period.

This is a time-out attribute, specified in seconds, for the server, cluster-sender, sender, and cluster-receiver channels. The interval is measured from the point at which a batch ends, that is when the batch size is reached or when the batch interval expires and the transmission queue becomes empty. If no messages arrive on the transmission queue during the specified time interval, the channel closes down. (The time is approximate.)

The close-down exchange of control data between the two ends of the channel includes an indication of the reason for closing. This ensures that the corresponding end of the channel remains available to start again.

You can specify any number of seconds from zero through 999 999 where a value of zero means no disconnect; wait indefinitely.

For server-connection channels using the TCP protocol, the interval represents the client inactivity disconnect value, specified in seconds. If a server-connection has

received no communication from its partner client for this duration, it will terminate the connection. The server-connection inactivity interval only applies between MQ API calls from a client, therefore no client will be disconnected during a long-running MQGET with wait call.

This attribute is valid for channel types of:

- Sender
- Server
- Server connection (z/OS only)
- Cluster sender
- Cluster receiver

This attribute is not applicable for server-connection channels using protocols other than TCP.

Note: Performance is affected by the value specified for the disconnect interval.

A very low value (a few seconds) can cause excessive overhead in constantly starting up the channel. A very large value (more than an hour) might mean that system resources are unnecessarily held up. You can also specify a heartbeat interval, so that when there are no messages on the transmission queue, the sending MCA will send a heartbeat flow to the receiving MCA, thus giving the receiving MCA an opportunity to quiesce the channel without waiting for the disconnect interval to expire. For these two values to work together effectively, the heartbeat interval value must be significantly lower than the disconnect interval value.

A value for the disconnect interval of a few minutes is a reasonable value to use. Change this value only if you understand the implications for performance, and you need a different value for the requirements of the traffic flowing down your channels.

For more information, see “Stopping and quiescing channels” on page 62.

Disposition (QSGDISP)

This attribute specifies the disposition of the channel in a queue-sharing group. Values are:

QMGR

The channel is defined on the page set of the queue manager that executes the command. This is the default.

GROUP

The channel is defined in the shared repository. This is allowed only if there is a shared queue manager environment. When a channel is defined with QSGDISP(GROUP), the command DEFINE CHANNEL(name) NOREPLACE QSGDISP(COPY) is generated automatically and sent to all active queue managers to cause them to make local copies on page set 0

COPY The channel is defined on the page set of the queue manager that executes the command, copying its definition from the QSGDISP(GROUP) channel of the same name. This is allowed only if there is a shared queue manager environment.

This attribute is valid for all channel types.

Header compression (COMPHDR)

This is a list of header data compression techniques supported by the channel. For sender, server, cluster-sender, cluster-receiver and client-connection channels the values specified are in order of preference with the first compression technique supported by the remote end of the channel being used. The channels' mutually supported compression techniques are passed to the sending channel's message exit where the compression technique used can be altered on a per message basis. Compression will alter the data passed to send and receive exits.

Possible values are:

NONE

No header data compression is performed. This is the default value.

SYSTEM

Header data compression is performed.

This attribute is valid for all channel types.

Heartbeat interval (HBINT)

You can specify the approximate time between heartbeat flows that are to be passed from a sending MCA when there are no messages on the transmission queue. Heartbeat flows unblock the receiving MCA, which is waiting for messages to arrive or for the disconnect interval to expire. When the receiving MCA is unblocked it can disconnect the channel without waiting for the disconnect interval to expire. Heartbeat flows also free any storage buffers that have been allocated for large messages and close any queues that have been left open at the receiving end of the channel.

The value is in seconds and must be in the range 0 through 999 999. A value of zero means that no heartbeat flows are to be sent. The default value is 300. To be most useful, the value should be significantly less than the disconnect interval value.

For a description of heartbeating with server-connection and client-connection channels, see *WebSphere MQ Clients*.

KeepAlive Interval (KAINT)

The KeepAlive Interval attribute is used to specify a time-out value for a channel.

The KeepAlive Interval attribute is a value passed to the communications stack specifying the KeepAlive timing for the channel. It allows you to specify a different keepalive value for each channel.

For this attribute to have any effect, TCP/IP keepalive must be enabled. On z/OS, you do this by issuing the ALTER QMGR TCPKEEP(YES) MQSC command. On other platforms, it occurs when the KEEPALIVE=YES parameter is specified in the TCP stanza in the distributed queuing configuration file, *qm.ini*, or through the WebSphere MQ Explorer. Keepalive must also be switched on within TCP/IP itself, using the TCP profile configuration data set.

The value indicates a time, in seconds, and must be in the range 0 to 99999. A KeepAlive Interval value of 0 indicates that channel-specific KeepAlive is not enabled for the channel and only the system-wide KeepAlive value set in TCP/IP

will be used. You can also set KAIN T to a value of AUTO (this is the default). If KAIN T is set to AUTO, the KeepAlive value is based on the value of the negotiated heartbeat interval (HBINT) as follows:

Table 8. Negotiated HBINT value and the corresponding KAIN T value

Negotiated HBINT	KAIN T
>0	Negotiated HBINT + 60 seconds
0	0

This attribute is valid for all channel types.

The value is ignored for all channels that have a TransportType (TRPTYPE) other than TCP or SPX

You can set the KeepAlive Interval (KAIN T) attribute for channels on a per-channel basis. On platforms other than z/OS, you can access and modify the parameter, but it is only stored and forwarded; there is no functional implementation of the parameter. If you need the functionality provided by the KAIN T parameter, use the Heartbeat Interval (HBINT) parameter, as described in "Heartbeat interval (HBINT)" on page 84.

Local Address (LOCLADDR)

This attribute only applies if Transport type (TRPTYPE) is TCP/IP. For all other transport types it is ignored.

This attribute specifies the local communications address for the channel. When a LOCLADDR value is specified, a channel that is stopped and then restarted continues to use the TCP/IP address specified in LOCLADDR. In recovery scenarios, this could be useful when the channel is communicating through a firewall, because it removes problems caused by the channel restarting with a different IP address, specified by the TCP/IP stack to which it is connected. You can also use this attribute to force a channel to use an IPV4 or an IPV6 stack on a dual stack system or use a dual mode stack on a single stack system.

This attribute is valid for channel types of:

- Sender
- Server
- Requester
- Client connection
- Cluster sender
- Cluster receiver

The value used is the optional IP address and optional port or port range to be used for outbound TCP/IP communications. The format is as follows:

```
LOCLADDR([ip-addr] [(low-port[,high-port])])
```

where "ip-addr" is specified in IPV4 dotted decimal form (for example 9.20.9.30), IPV6 hexadecimal form (for example fe80:43e4:0204:acff:fe97:2c34:fde0:3485), or dotted alphanumeric form (for example MACH1.ABC.COM), and "low-port" and "high-port" are port numbers enclosed in parentheses. When two port values are specified, the channel binds to the address specified, using an available port within the range covered by the two port values. All values are optional.

The maximum length of the string is MQ_LOCAL_ADDRESS_LENGTH.

When a channel is started the values specified for connection name (CONNAME) and local address (LOCLADDR) determine which IP stack is used for communication. The IP stack used is determined as follows:

- If the system has only an IPV4 stack configured, the IPV4 stack will always be used. If a local address (LOCLADDR) or connection name (CONNAME) is specified as an IPV6 network address, an error is generated and the channel will fail to start.
- If the system has only an IPV6 stack configured, the IPV6 stack will always be used. If a local address (LOCLADDR) is specified as an IPV4 network address, an error is generated and the channel will fail to start. On platforms that support IPV6 mapped addressing, if a connection name (CONNAME) is specified as an IPV4 network address, the IPV4 address is mapped to an IPV6 address (for example "xxx.xxx.xxx.xxx" is mapped to "::ffff:xxx.xxx.xxx.xxx"). Note that the use of mapped addresses may require protocol translators. Avoid the use of mapped addresses where possible.
- If a local address (LOCLADDR) is specified as an IP address for a channel, the stack for that IP address is used. If the local address (LOCLADDR) is specified as a hostname that resolves to both IPV4 and IPV6 addresses, the connection name (CONNAME) is used to determine which of the stacks is used. If both the local address (LOCLADDR) and connection name (CONNAME) are specified as hostnames that resolve to both IPV4 and IPV6 addresses, the stack used is determined by the queue manager attribute IPADDRV.
- If the system has dual IPV4 and IPV6 stacks configured and a local address (LOCLADDR) is not specified for a channel, the connection name (CONNAME) specified for the channel determines which IP stack to use. If the connection name (CONNAME) is specified as a hostname that resolves to both IPV4 and IPV6 addresses, the stack used is determined by the queue manager attribute IPADDRV.

Note: If the LOCLADDR port is in use, TCP/IP requires a time period to release the previously used port. If enough time is not left, and if only 1 LOCLADDR port is specified, the previously used port will not be available and so a random port will be chosen rather than the LOCLADDR port.

Long retry count (LONGRTY)

Specify the maximum number of times that the channel is to try allocating a session to its partner. If the initial allocation attempt fails, the *short retry count* number is decremented and the channel retries the remaining number of times. If it still fails, it retries a *long retry count* number of times with an interval of *long retry interval* between each try. If it is still unsuccessful, the channel closes down. The channel must subsequently be restarted with a command (it is not started automatically by the channel initiator).

(Retry is not attempted if the cause of failure is such that a retry is not likely to be successful.)

If the channel initiator (on z/OS) or queue manager (on distributed platforms) stops while the channel is retrying, the *short retry count* and *long retry count* are reset when the channel initiator or queue manager is restarted or when a message is successfully put at the sender channel. However, if the channel initiator (on z/OS) or queue manager (on distributed platforms) is shut down and restarted

immediately after either of those actions, the *short retry count* and *long retry count* are not reset. The channel retains the retry count values it had before the channel restart or the message being put.

The *long retry count* attribute can be set from zero through 999 999 999.

This attribute is valid for channel types of:

- Sender
- Server
- Cluster sender
- Cluster receiver

Note: For i5/OS, UNIX systems, and Windows systems, in order for retry to be attempted a channel initiator must be running. The channel initiator must be monitoring the initiation queue specified in the definition of the transmission queue that the channel is using.

Long retry interval (LONGTMR)

The approximate interval in seconds that the channel is to wait before retrying to establish connection, during the long retry mode.

The interval between retries may be extended if the channel has to wait to become active.

The channel tries to connect *long retry count* number of times at this long interval, after trying the *short retry count* number of times at the short retry interval.

This attribute can be set from zero through 999 999.

This attribute is valid for channel types of:

- Sender
- Server
- Cluster sender
- Cluster receiver

LU 6.2 mode name (MODENAME)

This is for use with LU 6.2 connections. It gives extra definition for the session characteristics of the connection when a communication session allocation is performed.

When using side information for SNA communications, the mode name is defined in the CPI-C Communications Side Object or APPC side information and this attribute should be left blank; otherwise, it should be set to the SNA mode name.

The name must be one to eight alphanumeric characters long.

This attribute is valid for channel types of:

- Sender
- Server
- Requester

- Client connection
- Cluster sender
- Cluster receiver

It is not valid for receiver or server-connection channels.

LU 6.2 transaction program name (TPNAME)

This is for use with LU 6.2 connections. It is the name, or generic name, of the transaction program (MCA) to be run at the far end of the link.

When using side information for SNA communications, the transaction program name is defined in the CPI-C Communications Side Object or APPC side information and this attribute should be left blank. Otherwise, this name is required by sender channels and requester channels.

The name can be up to 64 characters long.

The name should be set to the SNA transaction program name, unless the CONNAME contains a side-object name in which case it should be set to blanks. The actual name is taken instead from the CPI-C Communications Side Object, or the APPC side information data set.

This information is set in different ways on different platforms; see the section in this book about setting up communication for your platform.

This attribute is valid for channel types of:

- Sender
- Server
- Requester
- Client connection
- Cluster sender
- Cluster receiver

Maximum instances (MAXINST)

Specifies the maximum number of simultaneous instances of a server-connection channel that can be started.

This attribute can be set from zero through 999 999 999. A value of zero indicates that no client connections are allowed on this channel. The default value is 999 999 999.

If you do not have the Client Attachment feature (CAF) installed, the attribute can be set from zero to five only on the SYSTEM.ADMIN.SVRCONN channel. A value greater than five is interpreted as zero without the CAF installed.

If the value is reduced below the number of instances of the server-connection channel that are currently running, then the running channels are not affected. However, new instances are not able to start until sufficient existing ones have ceased to run.

This attribute is valid for server-connection channels only.

Maximum instances per client (MAXINSTC)

Specifies the maximum number of simultaneous instances of a server-connection channel that can be started from a single client.

This attribute can be set from zero through 999 999 999. A value of zero indicates that no client connections are allowed on this channel. The default value is 999 999 999.

If you do not have the Client Attachment feature (CAF) installed, the attribute can be set from zero to five only on the SYSTEM.ADMIN.SVRCONN channel. A value greater than five is interpreted as zero without the CAF installed.

If the value is reduced below the number of instances of the server-connection channel that are currently running from individual clients, then the running channels are not affected. However, new instances from those clients are not able to start until sufficient existing ones have ceased to run.

This attribute is valid for server-connection channels only.

Maximum message length (MAXMSGL)

Specifies the maximum length of a message that can be transmitted on the channel.

On WebSphere MQ for i5/OS, UNIX systems, and Windows systems, specify a value greater than or equal to zero, and less than or equal to the maximum message length for the queue manager. See the MAXMSGL parameter of the ALTER QMGR command in the WebSphere MQ Script (MQSC) Command Reference book for more information. On WebSphere MQ for z/OS, specify a value greater than or equal to zero, and less than or equal to *104 857 600 bytes*.

Because various implementations of WebSphere MQ systems exist on different platforms, the size available for message processing may be limited in some applications. This number must reflect a size that your system can handle without stress. When a channel starts up, the lower of the two numbers at each end of the channel is taken.

Note:

1. If splitting of messages is not supported at either end of a channel, the maximum message size cannot be greater than the negotiated maximum transmission size.
2. The IBM® WebSphere MQ products that this edition of the book applies to all support message splitting. Other WebSphere MQ products do not support message splitting.
3. For a comparison of the functions available, including the different maximum message lengths available see the WebSphere MQ Application Programming Guide.
4. You can use a maximum message size of 0 which will be taken to mean that the size is to be set to the local queue manager maximum value.

This attribute is valid for all channel types.

Message channel agent name (MCANAME)

This attribute is reserved and if specified must only be set to blanks. Its maximum length is 20 characters.

Message channel agent type (MCATYPE)

This attribute can specify the message channel agent as a *process* or a *thread*. On WebSphere MQ for z/OS, it is supported only for channels with a channel type of cluster-receiver.

Advantages of running as a process include:

- Isolation for each channel providing greater integrity
- Job authority specific for each channel
- Control over job scheduling

Advantages of threads include:

- Much reduced use of storage
- Easier configuration by typing on the command line
- Faster execution - it is quicker to start a thread than to instruct the operating system to start a process

For channel types of sender, server, and requester, the default is 'process'. For channel types of cluster-sender and cluster-receiver, the default is 'thread'. These defaults can change during your installation.

If you specify 'process' on the channel definition, a RUNMQCHL process is started. If you specify 'thread', the MCA runs on a thread of the AMQRMPPA process, or of the RUNMQCHI process if MQNOREMPOOL is specified. On the machine that receives the inbound allocates, the MCA runs as a thread or process depending on whether you use RUNMQLSR or inetd respectively.

On WebSphere MQ for z/OS, this attribute is supported only for channels with a channel type of cluster-receiver. On other platforms it is valid for channel types of:

- Sender
- Server
- Requester
- Cluster sender
- Cluster receiver

Message channel agent user identifier (MCAUSER)

This attribute is the user identifier (a string) to be used by the MCA for authorization to access WebSphere MQ resources, including (if PUT authority is DEF) authorization to put the message to the destination queue for receiver or requester channels.

On WebSphere MQ for Windows, the user identifier may be domain-qualified by using the format, `user@domain`, where the `domain` must be either the Windows systems domain of the local system or a trusted domain.

If this attribute is blank, the MCA uses its default user identifier.

This attribute is valid for channel types of:

- Sender
- Server
- Receiver

- Requester
- Server connection
- Cluster sender
- Cluster receiver

Message exit name (MSGEXIT)

Specifies the name of the user exit program to be run by the channel message exit. This can be a list of names of programs that are to be run in succession. Leave blank, if no channel message exit is in effect.

The format and maximum length of this attribute depend on the platform, as for “Receive exit name (RCVEXIT)” on page 95.

This attribute is valid for all channel types.

Message exit user data (MSGDATA)

Specifies user data that is passed to the channel message exits.

You can run a sequence of message exits. The limitations on the user data length and an example of how to specify MSGDATA for more than one exit are as shown for RCVDATA. See “Receive exit user data (RCVDATA)” on page 96.

This attribute is valid for all channel types.

Message-retry exit name (MREXIT)

Specifies the name of the user exit program to be run by the message-retry user exit. Leave blank if no message-retry exit program is in effect.

The format and maximum length of the name depend on the platform, as for “Receive exit name (RCVEXIT)” on page 95. However, there can only be one message-retry exit specified

This attribute is valid for channel types of:

- Receiver
- Requester
- Cluster receiver

Message-retry exit user data (MRDATA)

This is passed to the channel message-retry exit when it is called.

This attribute is valid for channel types of:

- Receiver
- Requester
- Cluster receiver

Message retry count (MRRTY)

This is the number of times the channel will retry before it decides it cannot deliver the message.

This attribute controls the action of the MCA only if the message-retry exit name is blank. If the exit name is not blank, the value of MRRTY is passed to the exit for the exit's use, but the number of retries performed (if any) is controlled by the exit, and not by this attribute.

The value must be in the range 0 to 999 999 999. A value of zero means that no retries will be performed. The default is 10.

This attribute is valid for channel types of:

- Receiver
- Requester
- Cluster receiver

Message retry interval (MRTMR)

This is the minimum interval of time that must pass before the channel can retry the MQPUT operation. This time interval is in milliseconds.

This attribute controls the action of the MCA only if the message-retry exit name is blank. If the exit name is not blank, the value of MRTMR is passed to the exit for the exit's use, but the retry interval is controlled by the exit, and not by this attribute.

The value must be in the range 0 to 999 999 999. A value of zero means that the retry will be performed as soon as possible (provided that the value of MRRTY is greater than zero). The default is 100.

This attribute is valid for channel types of:

- Receiver
- Requester
- Cluster receiver

Monitoring (MONCHL)

This attribute controls the collection of online Monitoring data.

Possible values are:

QMGR

The collection of Online Monitoring Data is inherited from the setting of the MONCHL attribute in the queue manager object. This is the default value.

OFF Online Monitoring Data collection for this channel is switched off.

LOW A low ratio of data collection with a minimal impact on performance. However, the monitoring results shown may not be totally up to date.

MEDIUM

A moderate ratio of data collection with limited impact on the performance of the system.

HIGH A high ratio of data collection with the possibility of an impact on performance. However, the monitoring results shown will be most current.

This attribute is valid for channel types of:

- Sender

- Server
- Receiver
- Requester
- Server connection
- Cluster sender
- Cluster receiver

Network-connection priority (NETPRTY)

The priority for the network connection. Distributed queuing will choose the path with the highest priority if there are multiple paths available. The value must be in the range 0 through 9; 0 is the lowest priority.

This attribute is valid for channel types of:

- Cluster receiver

Nonpersistent message speed (NPMSPEED)

The speed at which nonpersistent messages are to be sent. Possible values are:

NORMAL

Nonpersistent messages on a channel are transferred within transactions.

FAST Nonpersistent messages on a channel are not transferred within transactions.

The default is FAST. The advantage of this is that nonpersistent messages become available for retrieval far more quickly. The disadvantage is that because they are not part of a transaction, messages may be lost if there is a transmission failure or if the channel stops when the messages are in transit. See “Fast, nonpersistent messages” on page 20.

This attribute is valid for channel types of:

- Sender
- Server
- Receiver
- Requester
- Cluster sender
- Cluster receiver

Password (PASSWORD)

You can specify a password of maximum length 12 characters, although only the first 10 characters are used.

The password may be used by the MCA when attempting to initiate a secure LU 6.2 session with a remote MCA. It is valid for channel types of sender, server, requester, or client-connection.

On WebSphere MQ for z/OS, this attribute is valid only for client connection channels. On other platforms, it is valid for channel types of:

- Sender
- Server
- Requester

- Client connection
- Cluster sender

PUT authority (PUTAUT)

Use this attribute to choose the type of security processing to be carried out by the MCA when executing:

- An MQPUT command to the destination queue (for message channels) , or
- An MQI call (for MQI channels).

You can choose one of the following:

Process security, also called default authority (DEF)

The default user ID is used.

On platforms with Process security, you choose to have the queue security based on the user ID that the process is running under. The user ID is that of the process or user running the MCA at the receiving end of the message channel.

The queues are opened with this user ID and the open option MQOO_SET_ALL_CONTEXT.

Context security (CTX)

The alternate user ID is used from the context information associated with the message.

The *UserIdentifier* in the message descriptor is moved into the *AlternateUserId* field in the object descriptor. The queue is opened with the open options MQOO_SET_ALL_CONTEXT and MQOO_ALTERNATE_USER_AUTHORITY.

The user ID used to check open authority on the queue for MQOO_SET_ALL_CONTEXT and MQOO_ALTERNATE_USER_AUTHORITY is that of the process or user running the MCA at the receiving end of the message channel. The user ID used to check open authority on the queue for MQOO_OUTPUT is the *UserIdentifier* in the message descriptor.

Only Message Channel Agent security (ONLYMCA)

The default user ID is used.

On platforms with ONLYMCA security, you choose to have the queue security based on the user ID that the process is running under. The user ID is that of the process or user running the MCA at the receiving end of the message channel.

The queues are opened with this user ID and the open option MQOO_SET_ALL_CONTEXT.

This value only applies to z/OS.

Alternate Message Channel Agent security (ALTMCA)

This is the same as for ONLYMCA security but allows you to use context.

This value only applies to z/OS.

This attribute is valid for channel types of:

- Receiver
- Requester
- Server connection (z/OS only)

- Cluster receiver

Context security (CTX) and alternate message channel agent security (ALTMCA) values are not supported on server-connection channels.

Further details about context fields and open options can be found in the WebSphere MQ Application Programming Guide.

More information on security can be found in WebSphere MQ Security, the WebSphere MQ System Administration Guide for WebSphere MQ for i5/OS, UNIX systems, and Windows systems, or in the WebSphere MQ for z/OS System Administration Guide

Note: On WebSphere MQ for z/OS it is possible for two userids to be checked. Specific details of userids used by the channel initiator on z/OS can be found in the WebSphere MQ for z/OS System Setup Guide .

Queue manager name (QMNAME)

This is the name of the queue manager or queue manager group to which a WebSphere MQ client application can request connection.

This attribute is valid for channel types of:

- Client connection

Receive exit name (RCVEXIT)

Specifies the name of the user exit program to be run by the channel receive user exit. This can be a list of names of programs that are to be run in succession. Leave blank, if no channel receive user exit is in effect.

The format and maximum length of this attribute depend on the platform:

- On z/OS it is a load module name, maximum length 8 characters, except for client-connection channels where the maximum length is 128 characters.
- On i5/OS it is of the form:

libname/progname

when specified in CL commands.

When specified in WebSphere MQ Commands (MQSC) it has the form:

progname libname

where *progname* occupies the first 10 characters, and *libname* the second 10 characters (both blank-padded to the right if necessary). The maximum length of the string is 20 characters.

- On Windows it is of the form:

dllname(functionname)

where *dllname* is specified without the suffix “.DLL”. The maximum length of the string is 40 characters.

- On UNIX systems it is of the form:

libraryname(functionname)

The maximum length of the string is 40 characters.

During cluster sender channel auto-definition on z/OS, channel exit names are converted from the distributed platform format to z/OS format. If you want to control how exit names are converted, you can write a channel auto-definition exit. For more information, see “Channel auto-definition exit program” on page 393.

You can specify a list of receive, send, or message exit program names. The names should be separated by a comma, a space, or both. For example:

```
RCVEXIT(exit1 exit2)
MSGEXIT(exit1,exit2)
SENDEXIT(exit1, exit2)
```

The total length of the string of exit names and strings of user data for a particular type of exit is limited to 500 characters. In WebSphere MQ for i5/OS you can list up to 10 exit names. In WebSphere MQ for z/OS you can list up to eight exit names.

This attribute is valid for all channel types.

Receive exit user data (RCVDATA)

Specifies user data that is passed to the receive exit.

You can run a sequence of receive exits. The string of user data for a series of exits should be separated by a comma, spaces, or both. For example:

```
RCVDATA(exit1_data exit2_data)
MSGDATA(exit1_data,exit2_data)
SENDDATA(exit1_data, exit2_data)
```

In WebSphere MQ for UNIX systems, and Windows systems, the length of the string of exit names and strings of user data is limited to 500 characters. In WebSphere MQ for i5/OS you can specify up to 10 exit names and the length of user data for each is limited to 32 characters. In WebSphere MQ for z/OS you can specify up to eight strings of user data each of length 32 characters.

This attribute is valid for all channel types.

Security exit name (SCYEXIT)

Specifies the name of the exit program to be run by the channel security exit. Leave blank if no channel security exit is in effect.

The format and maximum length of the name depend on the platform, as for “Receive exit name (RCVEXIT)” on page 95. However, you can only specify one security exit.

This attribute is valid for all channel types.

Security exit user data (SCYDATA)

Specifies user data that is passed to the security exit. The maximum length is 32 characters.

This attribute is valid for all channel types.

Send exit name (SENDEXIT)

Specifies the name of the exit program to be run by the channel send exit. This can be a list of names of programs that are to be run in sequence. Leave blank if no channel send exit is in effect.

The format and maximum length of this attribute depend on the platform, as for “Receive exit name (RCVEXIT)” on page 95.

This attribute is valid for all channel types.

Send exit user data (SENDDATA)

Specifies user data that is passed to the send exit.

You can run a sequence of send exits. The limitations on the user data length and an example of how to specify SENDDATA for more than one exit, are as shown for RCVDATA. See “Receive exit user data (RCVDATA)” on page 96.

This attribute is valid for all channel types.

Sequence number wrap (SEQWRAP)

This is the highest number the message sequence number reaches before it restarts at 1.

The value of the number should be high enough to avoid a number being reissued while it is still being used by an earlier message. The two ends of a channel must have the same sequence number wrap value when a channel starts up; otherwise, an error occurs.

The value may be set from 100 through 999 999 999.

This attribute is valid for channel types of:

- Sender
- Server
- Receiver
- Requester
- Cluster sender
- Cluster receiver

Short retry count (SHORTRTY)

Specify the maximum number of times that the channel is to try allocating a session to its partner. If the initial allocation attempt fails, the *short retry count* is decremented and the channel retries the remaining number of times with an interval, defined in the *short retry interval* attribute, between each attempt. If it still fails, it retries *long retry count* number of times with an interval of *long retry interval* between each attempt. If it is still unsuccessful, the channel terminates.

(Retry is not attempted if the cause of failure is such that a retry is not likely to be successful.)

If the channel initiator (on z/OS) or queue manager (on distributed platforms) stops while the channel is retrying, the *short retry count* and *long retry count* are reset when the channel initiator or queue manager is restarted or when a message is successfully put at the sender channel. However, if the channel initiator (on z/OS) or queue manager (on distributed platforms) is shut down and restarted immediately after either of those actions, the *short retry count* and *long retry count* are not reset. The channel retains the retry count values it had before the channel restart or the message being put.

This attribute can be set from zero through 999 999 999.

This attribute is valid for channel types of:

- Sender
- Server
- Cluster sender
- Cluster receiver

Note: On i5/OS, UNIX systems, and Windows systems, in order for retry to be attempted a channel initiator must be running. The channel initiator must be monitoring the initiation queue specified in the definition of the transmission queue that the channel is using.

Short retry interval (SHORTTMR)

Specify the approximate interval in seconds that the channel is to wait before retrying to establish connection, during the short retry mode.

The interval between retries may be extended if the channel has to wait to become active.

This attribute can be set from zero through 999 999.

This attribute is valid for channel types of:

- Sender
- Server
- Cluster sender
- Cluster receiver

SSL Cipher Specification (SSLCIPH)

SSLCIPH defines a single CipherSpec for an SSL connection. Both ends of a WebSphere MQ SSL channel definition must include the attribute and the SSLCIPH values must specify the same CipherSpec on both ends of the channel. The value is a string with a maximum length of 32 characters.

This attribute is valid for all channel types.

It is valid only for channels with a transport type (TRPTYPE) of TCP. If the TRPTYPE is not TCP, the data is ignored and no error message is issued.

SSLCIPH is an optional attribute.

For more information on SSLCIPH, see WebSphere MQ Script (MQSC) Command Reference and WebSphere MQ Security.

SSL Client Authentication (SSLCAUTH)

SSLCAUTH is used to define whether the channel needs to receive and authenticate an SSL certificate from an SSL client. Possible values are:

OPTIONAL

If the peer SSL client sends a certificate, the certificate is processed as normal but authentication does not fail if no certificate is sent.

REQUIRED

If the SSL client does not send a certificate, authentication fails.

The default value is REQUIRED.

You can specify a value for SSLCAUTH on a non-SSL channel definition, one on which SSLCIPH is missing or blank. You can use this to temporarily disable SSL for debugging without first having to clear and then reinput the SSL parameters.

SSLCAUTH is an optional attribute.

This attribute is valid on all channel types that can ever receive a channel initiation flow, except for sender channels.

This attribute is valid for channel types of:

- Server
- Receiver
- Requester
- Server connection
- Cluster receiver

For more information on SSLCAUTH, see WebSphere MQ Script (MQSC) Command Reference and WebSphere MQ Security.

SSL Peer (SSLPEER)

The SSLPEER attribute is used to check the Distinguished Name (DN) of the certificate from the peer queue manager or client at the other end of a WebSphere MQ channel. If the DN received from the peer does not match the SSLPEER value, the channel does not start.

SSLPEER is an optional attribute. If a value is not specified, the peer DN is not checked when the channel is started.

On z/OS the maximum length of the attribute is 256 bytes. On all other platforms it is 1024 bytes.

On z/OS the attribute values used are not checked. If you input incorrect values, the channel fails at startup, and error messages are written to the error log at both ends of the channel. A Channel SSL Error event is also generated at both ends of the channel. On platforms that support SSLPEER, other than z/OS, the validity of the string is checked when it is first input.

You can specify a value for SSLPEER on a non-SSL channel definition, one on which SSLCIPH is missing or blank. You can use this to temporarily disable SSL for debugging without having to clear and later reinput the SSL parameters.

For more information on using SSLPEER, see WebSphere MQ Script (MQSC) Command Reference and WebSphere MQ Security.

This attribute is valid for all channel types.

Transmission queue name (XMITQ)

The name of the transmission queue from which messages are retrieved. This is required for channels of type sender or server, it is not valid for other channel types.

Provide the name of the transmission queue to be associated with this sender or server channel, that corresponds to the queue manager at the far side of the channel. The transmission queue may be given the same name as the queue manager at the remote end.

This attribute is valid for channel types of:

- Sender
- Server

Transport type (TRPTYPE)

The possible values are:

LU62	LU 6.2
TCP	TCP/IP
NETBIOS	NetBIOS (1)
SPX	SPX (1)
Notes:	
1. For use on Windows. Can also be used on z/OS for defining client-connection channels for use on Windows.	

This attribute is valid for all channel types.

User ID (USERID)

You can specify a task user identifier of 20 characters.

The user ID may be used by the MCA when attempting to initiate a secure SNA session with a remote MCA. It is valid for channel types of sender, server, requester, or client-connection.

This does not apply to WebSphere MQ for z/OS except for client-connection channels.

On the receiving end, if passwords are kept in encrypted format and the LU 6.2 software is using a different encryption method, an attempt to start the channel fails with invalid security details. You can avoid this by modifying the receiving SNA configuration to either:

- Turn off password substitution, or
- Define a security user ID and password.

On WebSphere MQ for z/OS, this attribute is valid only for client connection channels. On other platforms it is valid for channel types of:

- Sender
- Server
- Requester
- Client connection
- Cluster sender

Example configuration chapters in this book

Throughout the following parts of the book, there is a series of chapters containing examples of how to configure the various platforms to communicate with each other. These chapters describe tasks performed to establish a working WebSphere MQ network. The tasks were to establish WebSphere MQ *sender* and *receiver* channels to enable bi-directional message flow between the platforms over all supported protocols.

To use channel types other than sender-receiver, see the DEFINE CHANNEL command in WebSphere MQ Script (MQSC) Command Reference.

Figure 32 is a conceptual representation of a single channel and the WebSphere MQ objects associated with it.

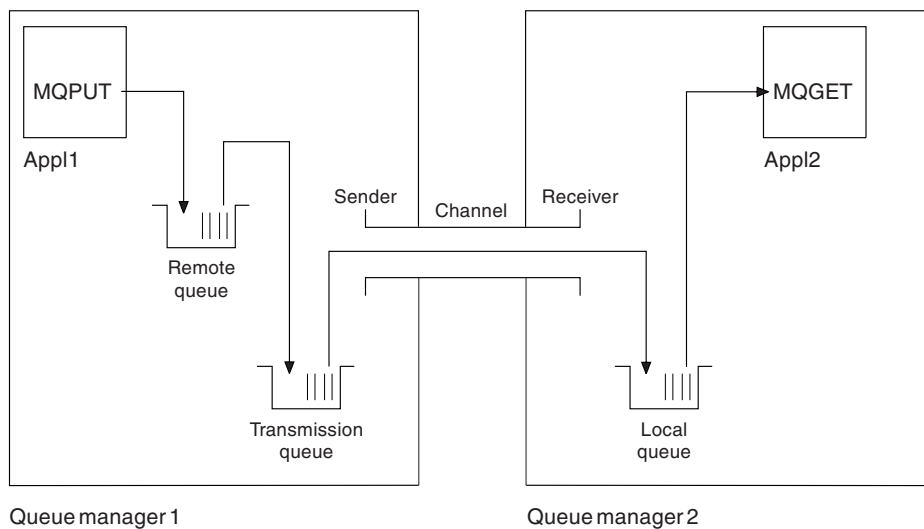


Figure 32. WebSphere MQ channel to be set up in the example configuration chapters in this book

This is a simple example, intended to introduce only the basic elements of the WebSphere MQ network. It does not demonstrate the use of triggering which is described in “Triggering channels” on page 18.

The objects in this network are:

- A remote queue
- A transmission queue
- A local queue
- A sender channel
- A receiver channel

App1 and App2 are both application programs; App1 is putting messages and App2 is receiving them.

Appl1 puts messages to a remote queue. The definition for this remote queue specifies the name of a target queue manager, a local queue on that queue manager, and a transmission queue on this the local queue manager.

When the queue manager receives the request from Appl1 to put a message to the remote queue, it looks at the queue definition and sees that the destination is remote. It therefore puts the message, along with a transmission header, straight onto the transmission queue specified in the definition. The message remains on the transmission queue until the channel becomes available, which may happen immediately.

A sender channel has in its definition a reference to one, and one only, transmission queue. When a channel is started, and at other times during its normal operation, it will look at this transmission queue and send any messages on it to the target system. The message has in its transmission header details of the destination queue and queue manager.

The intercommunication examples in the following chapters describe in detail the creation of each of the objects described above, for a variety of platform combinations.

On the target queue manager, definitions are required for the local queue and the receiver side of the channel. These objects operate independently of each other and so can be created in any sequence.

On the local queue manager, definitions are required for the remote queue, the transmission queue, and the sender side of the channel. Since both the remote queue definition and the channel definition refer to the transmission queue name, it is advisable to create the transmission queue first.

Network infrastructure

The configuration examples assume that all the systems are connected to a Token Ring network with the exception of z/OS which communicates via a 3745 (or equivalent) that is attached to the Token Ring, and Solaris, which is on an adjacent local area network (LAN) also attached to the 3745.

It is also assumed that, for SNA, all the required definitions in VTAM and network control program (NCP) are in place and activated for the LAN-attached platforms to communicate over the wide area network (WAN).

Similarly, for TCP, it is assumed that nameserver function is available, either via a domain nameserver or via locally held tables (for example a host file).

Communications software

Working configurations are given in the following chapters for the following network software products:

- SNA
 - Communications Manager/2 Version 1.11
 - Communications Server for Windows NT®, Version 5.0
 - AIX Communications Server, V5.0
 - Hewlett-Packard SNAplus2
 - i5/OS

- Data Connection SNAP-IX Version 6.2 or later
- OS/390® Version 2 Release 4
- TCP
 - Microsoft® Windows XP, Windows Server 2003, Windows Vista
 - AIX Version 4 Release 1.4
 - HP-UX Version 10.2 or later
 - Sun Solaris Release 2.4 or later
 - i5/OS
 - TCP for z/OS
 - HP Tru64 UNIX
- NetBIOS
- SPX

How to use the communication examples

The following chapters contain example configurations:

- “Example configuration - IBM WebSphere MQ for Windows” on page 129
- “Example configuration - IBM WebSphere MQ for AIX” on page 155
- “Example configuration - IBM WebSphere MQ for HP-UX” on page 172
- “Example configuration - IBM WebSphere MQ for Solaris” on page 194
- “Example configuration - IBM WebSphere MQ for z/OS” on page 266
- “Example configuration - IBM WebSphere MQ for i5/OS” on page 350
- “Example configuration - IBM WebSphere MQ for z/OS using queue-sharing groups” on page 286

The information in the example-configuration chapters describes the tasks that were carried out on a single platform, to set up communication to another of the platforms, and then describes the WebSphere MQ tasks to establish a working channel to that platform. Wherever possible, the intention is to make the information as generic as possible. Thus, to connect any two WebSphere MQ queue managers on different platforms, you should need to refer to only the relevant two chapters. Any deviations or special cases are highlighted as such. Of course, you can also connect two queue managers running on the same platform (on different machines or on the same machine). In this case, all the information can be derived from the one chapter.

The examples only cover how to set up communications where clustering is not being used. For information about setting up communications while using clustering, see the WebSphere MQ Queue Manager Clusters book. The communications’ configuration values given here still apply.

Each chapter contains a worksheet in which you can find the parameters used in the example configurations. There is a short description of each parameter and some guidance on where to find the equivalent values in your system. When you have a set of values of your own, record these in the spaces on the worksheet. As you proceed through the chapter, you will find cross-references to these values as you need them.

IT responsibilities

Because the IT infrastructure can vary greatly between organizations, it is difficult to indicate who, within an organization, controls and maintains the information required to complete each parameter value. To understand the terminology used in the following chapters, consider the following guidelines as a starting point.

- *System administrator* is used to describe the person (or group of people) who installs and configures the software for a specific platform.
- *Network administrator* is used to describe the person who controls LAN connectivity, LAN address assignments, network naming conventions, and so on. This person may be in a separate group or may be part of the system administration group.

In most z/OS installations, there is a group responsible for updating the ACF/VTAM, ACF/NCP, and TCP/IP software to support the network configuration. The people in this group should be the main source of information needed when connecting any WebSphere MQ platform to WebSphere MQ for z/OS. They may also influence or mandate network naming conventions on LANs and you should verify their span of control before creating your definitions.

- A specific type of administrator, for example *CICS[®] administrator* is indicated in cases where we can more clearly describe the responsibilities of the person.

The example-configuration chapters do not attempt to indicate who is responsible for and able to set each parameter. In general, several different people may be involved.

Chapter 3. DQM in WebSphere MQ for Windows and Unix platforms

Monitoring and controlling channels on Windows and Unix platforms

For DQM you need to create, monitor, and control the channels to remote queue managers. You can use the following types of command to do this:

The WebSphere MQ commands (MQSC)

You can use the MQSC as single commands in an MQSC session in Windows and UNIX systems. To issue more complicated, or multiple, commands the MQSC can be built into a file that you then run from the command line. For full details see the WebSphere MQ Script (MQSC) Command Reference. This chapter gives some simple examples of using MQSC for distributed queuing.

Control commands

You can also issue *control commands* at the command line for some of these functions. Reference material for these commands is contained in the WebSphere MQ System Administration Guide for WebSphere MQ for i5/OS, UNIX systems, and Windows systems.

Programmable command format commands

See the WebSphere MQ Programmable Command Formats and Administration Interface book for information about using these commands.

WebSphere MQ Explorer

On Unix and Windows, you can use the WebSphere MQ Explorer. This provides a graphical administration interface to perform administrative tasks as an alternative to using control commands or MQSC commands.

Each queue manager has a DQM component for controlling interconnections to compatible remote queue managers.

For a list of the functions available to you when setting up and controlling message channels, using the different types of command, see Table 9 on page 106.

DQM channel control

DQM channel control is achieved using commands, programs, WMQ Explorer, files for the channel definitions, and a storage area for synchronization information. The following is a brief description of the components.

- The channel commands are a subset of the WebSphere MQ Commands (MQSC).
- You use MQSC and the control commands to:
 - Create, copy, display, change, and delete channel definitions
 - Start and stop channels, ping, reset channel sequence numbers, and resolve in-doubt messages when links cannot be re-established
 - Display status information about channels
- Channel configuration and control commands are included in WebSphere MQ Explorer.

- Channel definitions are held as queue manager objects. The location of the default definitions is documented in the WebSphere MQ System Administration Guide.
- A storage area holds sequence numbers and *logical unit of work (LUW)* identifiers. These are used for channel synchronization purposes.

Functions available

Table 9 shows the full list of WebSphere MQ functions that you may need when setting up and controlling channels. The channel functions are explained in this chapter.

For more details of the control commands that you issue at the command line, see the WebSphere MQ System Administration Guide.

The MQSC commands are fully described in WebSphere MQ Script (MQSC) Command Reference.

Table 9. Functions available in Windows systems and UNIX systems

Function	Control commands	MQSC	WebSphere MQ Explorer equivalent?
Queue manager functions			
Change queue manager		ALTER QMGR	Yes
Create queue manager	crtmqm		Yes
Delete queue manager	dlmqm		Yes
Display queue manager		DISPLAY QMGR	Yes
End queue manager	endmqm		Yes
Ping queue manager		PING QMGR	No
Start queue manager	strmqm		Yes
Command server functions			
Display command server	dspmqcsv		No
End command server	endmqcsv		No
Start command server	strmqcsv		No
Queue functions			
Change queue		ALTER QALIAS ALTER QLOCAL ALTER QMODEL ALTER QREMOTE	Yes
Clear queue		CLEAR QLOCAL CLEAR QUEUE	Yes
Create queue		DEFINE QALIAS DEFINE QLOCAL DEFINE QMODEL DEFINE QREMOTE	Yes
Delete queue		DELETE QALIAS DELETE QLOCAL DELETE QMODEL DELETE QREMOTE	Yes
Display queue		DISPLAY QUEUE	Yes
Process functions			

Table 9. Functions available in Windows systems and UNIX systems (continued)

Function	Control commands	MQSC	WebSphere MQ Explorer equivalent?
Change process		ALTER PROCESS	Yes
Create process		DEFINE PROCESS	Yes
Delete process		DELETE PROCESS	Yes
Display process		DISPLAY PROCESS	Yes
Channel functions			
Change channel		ALTER CHANNEL	Yes
Create channel		DEFINE CHANNEL	Yes
Delete channel		DELETE CHANNEL	Yes
Display channel		DISPLAY CHANNEL	Yes
Display channel status		DISPLAY CHSTATUS	Yes
End channel		STOP CHANNEL	Yes
Ping channel		PING CHANNEL	Yes
Reset channel		RESET CHANNEL	Yes
Resolve channel		RESOLVE CHANNEL	Yes
Run channel	runmqchl	START CHANNEL	Yes
Run channel initiator	runmqchi	START CHINIT	No
Run listener ¹	runmqslr	START LISTENER	No
End listener	endmqslr (Windows systems, AIX, HP-UX, and Solaris only)		No
Note:			
1. A listener may be started automatically when the queue manager starts. See the DEFINE LISTENER command in WebSphere MQ Script (MQSC) Command Reference.			

Getting started with objects

Use the WebSphere MQ commands (MQSC) or the WebSphere MQ Explorer to:

1. Define message channels and associated objects
2. Monitor and control message channels

The associated objects you may need to define are:

- Transmission queues
- Remote queue definitions
- Queue manager alias definitions
- Reply-to queue alias definitions
- Reply-to local queues
- Processes for triggering (MCAs)
- Message channel definitions

Channels must be completely defined, and their associated objects must exist and be available for use, before a channel can be started. This chapter shows you how to do this.

In addition, the particular communication link for each channel must be defined and available before a channel can be run. For a description of how LU 6.2, TCP/IP, NetBIOS, SPX, and DECnet links are defined, see the particular communication guide for your installation. See also the example configuration chapters in this book.

Creating associated objects

Use MQSC to create the queue and alias objects: transmission queues, remote queue definitions, queue manager alias definitions, reply-to queue alias definitions, and reply-to local queues.

Also create the definitions of processes for triggering (MCAs) in a similar way.

For an example showing how to create all the required objects see “Message channel planning example for distributed platforms” on page 238.

Creating default objects

Default objects are created automatically when a queue manager is created. These objects are queues, channels, a process definition, and administration queues.

How are default objects created?:

When you use the `crtmqm` command to create a queue manager, the command also initiates a program to create a set of default objects.

1. Each default object is created in turn. The program keeps a count of how many objects are successfully defined, how many already existed and were replaced, and how many unsuccessful attempts there were.
2. The program displays the results to you and if any errors occurred, directs you to the appropriate error log for details.

When the program has finished running, you can use the `strmqm` command to start the queue manager.

See the WebSphere MQ System Administration Guide book for information about the `crtmqm` and `strmqm` commands and a list of default objects.

Changing the default objects:

After the default objects have been created, you can replace them at any time by running the `strmqm` command with the `-c` option.

When you specify the `-c` option, the queue manager is started temporarily while the objects are created and is then shut down again. Issuing `strmqm` with the `-c` option refreshes existing system objects with the default values (for example, the `MCAUSER` attribute of a channel definition is set to blanks). You must use the `strmqm` command again, without the `-c` option, if you want to start the queue manager.

If you want to make any changes to the default objects, you can create your own version of the old `amqscoma.tst` file and edit it.

Creating a channel

To create a new channel you have to create *two* channel definitions, one at each end of the connection. You create the first channel definition at the first queue manager. Then you create the second channel definition at the second queue manager, on the other end of the link.

Both ends must be defined using the *same* channel name. The two ends must have **compatible** channel types, for example: Sender and Receiver.

To create a channel definition for one end of the link use the MQSC command DEFINE CHANNEL. Include the name of the channel, the channel type for this end of the connection, a connection name, a description (if required), the name of the transmission queue (if required), and the transmission protocol. Also include any other attributes that you want to be different from the system default values for the required channel type, using the information you have gathered previously.

You are provided with help in deciding on the values of the channel attributes in “Channel attributes” on page 71.

Note: You are very strongly recommended to name all the channels in your network uniquely. Including the source and target queue manager names in the channel name is a good way to do this.

Create channel example:

```
DEFINE CHANNEL(QM1.TO.QM2) CHLTYPE(SDR) +  
DESCR('Sender channel to QM2') +  
CONNAME(QM2) TRPTYPE(TCP) XMITQ(QM2) CONVERT(YES)
```

In all the examples of MQSC the command is shown as it would appear in a file of commands, and as it would be typed in Windows or UNIX systems. The two methods look identical, except that to issue a command interactively, you must first start an MQSC session. Type `runmqsc`, for the default queue manager, or `runmqsc qmname` where *qmname* is the name of the required queue manager. Then type any number of commands, as shown in the examples.

For portability, you should restrict the line length of your commands to 72 characters. Use the concatenation character, `+`, as shown to continue over more than one line. On Windows use Ctrl-z to end the input at the command line. On UNIX systems, use Ctrl-d. Alternatively, on UNIX or Windows, use the **end** command.

Displaying a channel

Use the MQSC command DISPLAY CHANNEL, specifying the channel name, the channel type (optional), and the attributes you want to see, or specifying that all attributes are to be displayed. The ALL parameter of the DISPLAY CHANNEL command is assumed by default if no specific attributes are requested and the channel name specified is not generic.

The attributes are described in “Channel attributes” on page 71.

Display channel examples:

```
DISPLAY CHANNEL(QM1.TO.QM2) TRPTYPE,CONVERT  
  
DISPLAY CHANNEL(QM1.TO.*) TRPTYPE,CONVERT
```

```
DISPLAY CHANNEL(*) TRPTYPE, CONVERT
```

```
DISPLAY CHANNEL(QM1.TO.QMR34) ALL
```

Displaying channel status

Use the MQSC command `DISPLAY CHSTATUS`, specifying the channel name and whether you want the current status of channels or the status of saved information.

Display channel status examples:

```
DISPLAY CHSTATUS(*) CURRENT
```

```
DISPLAY CHSTATUS(QM1.TO.*) SAVED
```

Note that the saved status does not apply until at least one batch of messages has been transmitted on the channel. Status is also saved when a channel is stopped (using the `STOP CHL` command) and when the queue manager is ended.

Starting a channel

For applications to be able to exchange messages you must start a listener program for inbound connections (or, in the case of UNIX systems, create a listener attachment). On Windows and Unix systems, use the `runmqlsr` command to start the WebSphere MQ listener process. By default, any inbound requests for channel attachment causes the listener process to start MCAs as threads of the `amqrmppa` process.

```
runmqlsr -t tcp -m QM2
```

For outbound connections you must start the channel in one of the following three ways:

1. Use the MQSC command `START CHANNEL`, specifying the channel name, to start the channel as a process or a thread, depending on the `MCATYPE` parameter. (If channels are started as threads, they are threads of a channel initiator.)

```
START CHANNEL(QM1.TO.QM2)
```
2. Use the control command `runmqchl` to start the channel as a process.

```
runmqchl -c QM1.TO.QM2 -m QM1
```
3. Use the channel initiator to trigger the channel.

Renaming a channel

To rename a message channel, use MQSC to carry out the following steps:

1. Use `STOP CHANNEL` to stop the channel.
2. Use `DEFINE CHANNEL` to create a duplicate channel definition with the new name.
3. Use `DISPLAY CHANNEL` to check that it has been created correctly.
4. Use `DELETE CHANNEL` to delete the original channel definition.

If you decide to rename a message channel, remember that a channel has *two* channel definitions, one at each end. Make sure you rename the channel at both ends at the same time.

Channel attributes and channel types

The channel attributes for each type of channel are shown in Table 7 on page 71. The channel attributes are described in detail in “Channel attributes” on page 71. Client-connection channels and server-connection channels are described in the WebSphere MQ Clients book.

Channel functions

The channel functions available are shown in Table 9 on page 106. Here some more detail is given about the channel functions.

Create

Use the MQSC command `DEFINE CHANNEL` to create a new channel definition.

You can create a new channel definition using the default values supplied by WebSphere MQ, specifying the name of the channel, the type of channel you are creating, the communication method to be used, the transmission queue name and the connection name.

The channel name must be the same at both ends of the channel, and unique within the network. However, you must restrict the characters used to those that are valid for WebSphere MQ object names.

Change

Use the MQSC command `ALTER CHANNEL` to change an existing channel definition, except for the channel name, or channel type.

Delete

Use the MQSC command `DELETE CHANNEL` to delete a named channel.

Display

Use the MQSC command `DISPLAY CHANNEL` to display the current definition for the channel.

Display Status

The MQSC command `DISPLAY CHSTATUS` displays the status of a channel whether the channel is active or inactive. It applies to all message channels. It does not apply to MQI channels other than server-connection channels.. See “Displaying channel status” on page 110.

Information displayed includes:

- Channel name
- Communication connection name
- In-doubt status of channel (where appropriate)
- Last sequence number
- Transmission queue name (where appropriate)
- The in-doubt identifier (where appropriate)
- The last committed sequence number
- Logical unit of work identifier

- Process ID
- Thread ID (Windows only)

Ping

Use the MQSC command PING CHANNEL to exchange a fixed data message with the remote end. This gives some confidence to the system supervisor that the link is available and functioning.

Ping does not involve the use of transmission queues and target queues. It uses channel definitions, the related communication link, and the network setup. It can only be used if the channel is not currently active.

It is available from sender and server channels only. The corresponding channel is started at the far side of the link, and performs the startup parameter negotiation. Errors are notified normally.

The result of the message exchange is presented as Ping complete or an error message.

Ping with LU 6.2:

When Ping is invoked, by default no USERID or password flows to the receiving end. If USERID and password are required, they can be created at the initiating end in the channel definition. If a password is entered into the channel definition, it is encrypted by WebSphere MQ before being saved. It is then decrypted before flowing across the conversation.

Start

Use the MQSC command START CHANNEL for sender, server, and requester channels. It should not be necessary where a channel has been set up with queue manager triggering.

When started, the sending MCA reads the channel definitions and opens the transmission queue. A channel start-up sequence is executed, which remotely starts the corresponding MCA of the receiver or server channel. When they have been started, the sender and server processes await messages arriving on the transmission queue and transmit them as they arrive.

When you use triggering or run channels as threads, ensure the channel initiator is available to monitor the initiation queue. The channel initiator is started by default as part of the queue manager.

However, TCP and LU 6.2 do provide other capabilities:

- For TCP on UNIX systems, inetd can be configured to start a channel. This will be started as a separate process.
- For LU 6.2 in UNIX systems, configure your SNA product to start the LU 6.2 responder process.
- For LU 6.2 in Windows systems, using SNA Server you can use TpStart (a utility provided with SNA Server) to start a channel. This will be started as a separate process.

Use of the Start option always causes the channel to re-synchronize, where necessary.

For the start to succeed:

- Channel definitions, local and remote, must exist. If there is no appropriate channel definition for a receiver or server-connection channel, a default one is created automatically if the channel is auto-defined. See “Channel auto-definition exit program” on page 393.
- Transmission queue must exist, and have no other channels using it.
- MCAs, local and remote, must exist.
- Communication link must be available.
- Queue managers must be running, local and remote.
- Message channel must not be already running.

A message is returned to the screen confirming that the request to start a channel has been accepted. For confirmation that the start command has succeeded, check the error log, or use DISPLAY CHSTATUS. The error logs are:

Windows

mqmtop\qmgrs\qmname\errors\AMQERR01.LOG (for each queue manager called *qmname*)

mqmtop\qmgrs\@SYSTEM\errors\AMQERR01.LOG (for general errors)

Note: On Windows systems, you still also get a message in the Windows systems application event log.

UNIX systems

/var/mqm/qmgrs/qmname/errors/AMQERR01.LOG (for each queue manager called *qmname*)

/var/mqm/qmgrs/@SYSTEM/errors/AMQERR01.LOG (for general errors)

Stop

Use the MQSC command STOP CHANNEL to request the channel to stop activity. The channel will not start a new batch of messages until the operator starts the channel again. (For information about restarting stopped channels, see “Restarting stopped channels” on page 64.)

This command can be issued to a channel of any type except MQCHT_CLNTCONN.

You can select the type of stop you require:

Stop quiesce example:

```
STOP CHANNEL(QM1.TO.QM2) MODE(QUIESCE)
```

This command requests the channel to close down in an orderly way. The current batch of messages is completed and the syncpoint procedure is carried out with the other end of the channel.

Note: If the channel is idle this command will not terminate a receiving channel.

Stop force example:

```
STOP CHANNEL(QM1.TO.QM2) MODE(FORCE)
```

This option stops the channel immediately, but does not terminate the channel’s thread or process. The channel does not complete processing the current batch of

messages, and can, therefore, leave the channel in doubt. In general, it is recommended that operators use the quiesce stop option.

Stop terminate example:

```
STOP CHANNEL(QM1.TO.QM2) MODE(TERMINATE)
```

This option stops the channel immediately, and terminates the channel's thread or process.

Stop (quiesce) stopped example:

```
STOP CHANNEL(QM1.TO.QM2) STATUS(STOPPED)
```

This command does not specify a MODE, so will default to MODE(QUIESCE). It requests that the channel be stopped so that it cannot be restarted automatically but must be started manually.

Stop (quiesce) inactive example:

```
STOP CHANNEL(QM1.TO.QM2) STATUS(INACTIVE)
```

This command does not specify a MODE, so will default to MODE(QUIESCE). It requests that the channel be made inactive so that it will be restarted automatically when required.

Reset

Use the MQSC command RESET CHANNEL to change the message sequence number. This command is available for any message channel, but not for MQI channels (client-connection or server-connection). The first message starts the new sequence the next time the channel is started.

If the command is issued on a sender or server channel, it informs the other side of the change when the channel is restarted.

Resolve

Use the MQSC command RESOLVE CHANNEL when messages are held in-doubt by a sender or server, for example because one end of the link has terminated, and there is no prospect of it recovering. The RESOLVE CHANNEL command accepts one of two parameters: BACKOUT or COMMIT. Backout restores messages to the transmission queue, while Commit discards them.

The channel program does not try to establish a session with a partner. Instead, it determines the logical unit of work identifier (LUWID) which represents the in-doubt messages. It then issues, as requested, either:

- BACKOUT to restore the messages to the transmission queue; or
- COMMIT to delete the messages from the transmission queue.

For the resolution to succeed:

- The channel must be inactive
- The channel must be in doubt
- The channel type must be sender or server
- A local channel definition must exist
- The local queue manager must be running

Preparing WebSphere MQ for distributed platforms

This chapter describes the WebSphere MQ preparations required before DQM can be used in Windows and UNIX systems. It includes “Transmission queues and triggering” and “Channel programs” on page 116.

Transmission queues and triggering

Before a channel (other than a requester channel) can be started, the transmission queue must be defined as described in this chapter, and must be included in the message channel definition.

In addition, where needed, the triggering arrangement must be prepared with the definition of the necessary processes and queues.

Creating a transmission queue

Define a local queue with the USAGE attribute set to XMITQ for each sending message channel. If you want to make use of a specific transmission queue in your remote queue definitions, create a remote queue as shown below.

To create a transmission queue, use the WebSphere MQ Commands (MQSC), as shown in the following examples:

Create transmission queue example

```
DEFINE QLOCAL(QM2) DESCR('Transmission queue to QM2') USAGE(XMITQ)
```

Create remote queue example

```
DEFINE QREMOTE(PAYROLL) DESCR('Remote queue for QM2') +  
XMITQ(QM2) RNAME(PAYROLL) RQMNAME(QM2)
```

The recommended name for the transmission queue is the queue manager name on the remote system, as shown in the examples above.

Triggering channels

An overview of triggering is given in “Triggering channels” on page 18, while it is described in depth in the WebSphere MQ Application Programming Guide. This description provides you with information specific to WebSphere MQ for UNIX and Windows systems.

You can create a process definition in WebSphere MQ, defining processes to be triggered. Use the MQSC command DEFINE PROCESS to create a process definition naming the process to be triggered when messages arrive on a transmission queue. The USERDATA attribute of the process definition should contain the name of the channel being served by the transmission queue.

Alternatively, for WebSphere MQ for UNIX systems and Windows, you can eliminate the need for a process definition by specifying the channel name in the TRIGDATA attribute of the transmission queue.

If you do not specify a channel name, the channel initiator searches the channel definition files until it finds a channel that is associated with the named transmission queue.

Example definitions for triggering:

Define the local queue (QM4), specifying that trigger messages are to be written to the default initiation queue SYSTEM.CHANNEL.INITQ, to trigger the application (process P1) that starts channel (QM3.TO.QM4):

```
DEFINE QLOCAL(QM4) TRIGGER INITQ(SYSTEM.CHANNEL.INITQ) PROCESS(P1) USAGE (XMITQ)
```

Define the application (process P1) to be started:

```
DEFINE PROCESS(P1) USERDATA(QM3.TO.QM4)
```

Examples for WebSphere MQ for UNIX systems and Windows systems:

Define the local queue (QM4), specifying that trigger messages are to be written to the initiation queue (IQ) to trigger the application that starts channel (QM3.TO.QM4):

```
DEFINE QLOCAL(QM4) TRIGGER INITQ(SYSTEM.CHANNEL.INITQ) +  
USAGE (XMITQ) TRIGDATA(QM3.TO.QM4)
```

Starting the channel initiator:

Triggering is implemented using the channel initiator process. This process is started with the MQSC command START CHINIT. Unless you are using the default initiation queue, specify the name of the initiation queue on the command. For example, to use the START CHINIT command to start queue IQ for the default queue manager, enter:

```
START CHINIT INITQ(IQ)
```

By default, a channel initiator is started automatically using the default initiation queue, SYSTEM.CHANNEL.INITQ. If you want to start all your channel initiators manually, follow these steps:

1. Create and start the queue manager.
2. Alter the queue manager's SCHINIT property to MANUAL
3. End and restart the queue manager

The number of channel initiators that you can start is limited. The default limit is 3, which is also the maximum. You can reduce this limit using MAXINITIATORS in the qm.ini file for UNIX systems, or by using the WebSphere MQ Explorer. If you increase the value of MAXINITIATORS beyond 3, it has the same effect as setting it to 3.

See the WebSphere MQ System Administration Guide for details of the run channel initiator command runmqchi, and the other control commands.

Stopping the channel initiator:

The default channel initiator is started automatically when you start a queue manager. All channel initiators are stopped automatically when a queue manager is stopped.

Channel programs

There are different types of channel programs (MCAs) available for use at the channels. The names are shown in the following tables.

Table 10. Channel programs for Windows and UNIX systems

Program name	Direction of connection	Communication
amqrmppa		Any
runmqlsr	Inbound	Any
amqcrs6a	Inbound	LU 6.2
amqcrsta	Inbound	TCP
runmqchl	Outbound	Any
runmqchi	Outbound	Any

runmqlsr (Run WebSphere MQ listener), runmqchl (Run WebSphere MQ channel), and runmqchi (Run WebSphere MQ channel initiator) are control commands that you can enter at the command line.

amqcrsta is invoked for TCP channels on UNIX using inetd, where no listener is started.

amqcrs6a is invoked as a transaction program when using LU6.2

Examples of the use of these channel programs are given in the following chapters.

Other things to consider

Here are some other topics that you should consider when preparing WebSphere MQ for distributed queue management.

Undelivered-message queue

A DLQ handler is provided with WebSphere MQ on UNIX systems. See the WebSphere MQ System Administration Guide book for WebSphere MQ for information about this.

Queues in use

MCAs for receiver channels may keep the destination queues open even when messages are not being transmitted; this results in the queues appearing to be “in use”.

Security of WebSphere MQ objects

This section deals with remote messaging aspects of security.

You need to provide users with authority to make use of the WebSphere MQ facilities, and this is organized according to actions to be taken with respect to objects and definitions. For example:

- Queue managers can be started and stopped by authorized users
- Applications need to connect to the queue manager, and have authority to make use of queues
- Message channels need to be created and controlled by authorized users
- Objects are kept in libraries, and access to these libraries may be restricted

The message channel agent at a remote site needs to check that the message being delivered originated from a user with authority to do so at this remote site. In

addition, as MCAs can be started remotely, it may be necessary to verify that the remote processes trying to start your MCAs are authorized to do so. There are three possible ways for you to deal with this:

1. Specify `PUTAUT=CTX` in the channel definition to indicate that messages must contain acceptable *context* authority, otherwise they will be discarded.
2. Implement *user exit* security checking to ensure that the corresponding message channel is authorized. The security of the installation hosting the corresponding channel ensures that all users are properly authorized, so that you do not need to check individual messages.
3. Implement *user exit* message processing to ensure that individual messages are vetted for authorization.

On UNIX systems:

Administration users must be part of the `mqm` group on your system (including root) if this ID is going to use WebSphere MQ administration commands.

You should always run `amqcrsta` as the “mqm” user ID.

User IDs on UNIX systems:

The queue manager converts all uppercase or mixed case user identifiers into lowercase, before inserting them into the context part of a message, or checking their authorization. All authorizations should therefore be based only on lowercase identifiers.

Message descriptor extension (MQMDE):

When the listener program (`amqcrsta`, for example) is started by `inetd` it inherits the locale from `inetd`. It is possible that the MQMDE will not be honored and will be placed on the queue as message data.

To ensure that the MQMDE is honored (merged) the locale must be set correctly. The locale set by `inetd` may not match that chosen for other locales used by WebSphere MQ processes.

To set the locale, create a shell script which sets the locale environment variables `LANG`, `LC_COLLATE`, `LC_CTYPE`, `LC_MONETARY`, `LC_NUMERIC`, `LC_TIME`, and `LC_MESSAGES` to the locale used for other WebSphere MQ processes. In the same shell script call the listener program. Modify the `inetd.conf` file to call your shell script in place of the listener program.

On Windows systems:

Administration users must be part of both the `mqm` group and the administrators group on Windows systems if this ID is going to use WebSphere MQ administration commands.

User IDs on Windows systems:

On Windows systems, *if there is no message exit installed*, the queue manager converts any uppercase or mixed case user identifiers into lowercase, before inserting them into the context part of a message, or checking their authorization. All authorizations should therefore be based only on lowercase identifiers.

User IDs across systems:

Platforms other than Windows systems and UNIX systems use uppercase characters for user IDs. To allow Windows systems and UNIX systems to use lowercase user IDs, the following conversions are carried out by the message channel agent (MCA) on these platforms:

At the sending end

The alpha characters in all user IDs are converted to uppercase, *if there is no message exit installed.*

At the receiving end

The alpha characters in all user IDs are converted to lowercase, *if there is no message exit installed.*

Note that the automatic conversions are *not* carried out if you provide a message exit on UNIX systems and Windows systems for any other reason.

System extensions and user-exit programs

A facility is provided in the channel definition to allow extra programs to be run at defined times during the processing of messages. These programs are not supplied with WebSphere MQ, but may be provided by each installation according to local requirements.

In order to run, these user-exit programs must have predefined names and be available on call to the channel programs. The names of the user-exit programs are included in the message channel definitions.

There is a defined control block interface for handing over control to these programs, and for handling the return of control from these programs.

The precise places where these programs are called, and details of control blocks and names, are to be found in Chapter 6, "Further intercommunication considerations," on page 375.

Running channels and listeners as trusted applications

If performance is an important consideration in your environment and your environment is stable, you can choose to run your channels and listeners as trusted, that is, using the fastpath binding. There are two factors that influence whether or not channels and listeners run as trusted:

- The environment variable MQ_CONNECT_TYPE=FASTPATH or MQ_CONNECT_TYPE=STANDARD. This is case sensitive. If you specify a value that is not valid it is ignored.
- MQIBindType in the Channels stanza of the qm.ini or registry file. You can set this to FASTPATH or STANDARD and it is not case-sensitive. The default is STANDARD.

You can use MQIBindType in association with the environment variable to achieve the required effect as follows:

MQIBindType	Environment variable	Result
STANDARD	UNDEFINED	STANDARD
FASTPATH	UNDEFINED	FASTPATH
STANDARD	STANDARD	STANDARD
FASTPATH	STANDARD	STANDARD

MQIBindType	Environment variable	Result
STANDARD	FASTPATH	STANDARD
FASTPATH	FASTPATH	FASTPATH

In summary, there are only two ways of actually making channels and listeners run as trusted:

1. By specifying MQIBindType=FASTPATH in qm.ini or registry and not specifying the environment variable.
2. By specifying MQIBindType=FASTPATH in qm.ini or registry and setting the environment variable to FASTPATH.

You are recommended to run listeners as trusted, because listeners are stable processes. You are recommended to run channels as trusted, unless you are using unstable channel exits or the command STOP CHANNEL MODE(TERMINATE).

What next?

When you have made the preparations described in this chapter you are ready to set up communications. Proceed to one of the following chapters, depending on what platform you are using:

- “Setting up communication for Windows”
- “Setting up communication on UNIX systems” on page 150

Setting up communication for Windows

When a distributed-queuing management channel is started, it tries to use the connection specified in the channel definition. For this to succeed, it is necessary for the connection to be defined and available. This chapter explains how to do this. You may also find it helpful to refer to “Example configuration - IBM WebSphere MQ for Windows” on page 129.

For UNIX systems see “Setting up communication on UNIX systems” on page 150.

Deciding on a connection

There are four forms of communication for WebSphere MQ for Windows systems:

- TCP
- LU 6.2
- NetBIOS
- SPX (Windows XP and Windows 2003 Server only)

Each channel definition must specify only one protocol as the Transmission protocol (Transport Type) attribute. One or more protocols may be used by a queue manager.

For WebSphere MQ clients, it may be useful to have alternative channels using different transmission protocols. See the WebSphere MQ Clients book.

Defining a TCP connection

The channel definition at the sending end specifies the address of the target. A listener program must be run at the receiving end.

Sending end

Specify the host name, or the TCP address of the target machine, in the Connection name field of the channel definition. The port to connect to will default to 1414. Port number 1414 is assigned by the Internet Assigned Numbers Authority to WebSphere MQ.

To use a port number other than the default, change the connection name field thus:

```
Connection Name OS2R0G3(1822)
```

where 1822 is the port required. (This must be the port that the listener at the receiving end is listening on.)

You can change the default port number by specifying it in the registry for WebSphere MQ for Windows:

```
TCP:  
  Port=1822
```

Note: To select which TCP/IP port number to use, WebSphere MQ uses the first port number it finds in the following sequence:

1. The port number explicitly specified in the channel definition or command line. This number allows the default port number to be overridden for a channel.
2. The port attribute specified in the registry. This number allows the default port number to be overridden for a queue manager.
3. The default value of 1414. This is the number assigned to WebSphere MQ by the Internet Assigned Numbers Authority.

For more information about the values you set using `qm.ini`, see Chapter 8, "Configuration file stanzas for distributed queuing," on page 491.

Receiving on TCP

Receiving channel programs are started in response to a startup request from the sending channel. To do this, a listener program has to be started to detect incoming network requests and start the associated channel. You should use the WebSphere MQ listener.

Using the WebSphere MQ listener:

To run the Listener supplied with WebSphere MQ, that starts new channels as threads, use the `RUNMQLSR` command. For example:

```
RUNMQLSR -t tcp [-m QMNAME] [-p 1822]
```

The square brackets indicate optional parameters; `QMNAME` is not required for the default queue manager, and the port number is not required if you are using the default (1414).

For the best performance, run the WebSphere MQ listener as a trusted application as described in “Running channels and listeners as trusted applications” on page 119. See the WebSphere MQ Application Programming Guide for information about trusted applications.

You can stop all WebSphere MQ listeners running on a queue manager that is inactive, using the command:

```
ENDMQLSR [-m QMNAME]
```

If you do not specify a queue manager name, the default queue manager is assumed.

Using the TCP/IP SO_KEEPALIVE option:

If you want to use the SO_KEEPALIVE option (as discussed in “Checking that the other end of the channel is still available” on page 61) you need to add the following entry to your registry:

```
TCP:
    KeepAlive=yes
```

On Windows, the TCP configuration registry value for KeepAliveTime controls the interval that elapses before the connection will be checked. The default is two hours. For information about changing this value, see the Microsoft article *TCP/IP and NBT Configuration Parameters for XP*.

Defining an LU 6.2 connection

SNA must be configured so that an LU 6.2 conversation can be established between the two machines. Then proceed as follows.

See the following table for information.

Table 11. Settings on the local Windows system for a remote queue manager platform

Remote platform	TPNAME	TPPATH
z/OS or OS/390 or MVS/ESA without CICS	The same as in the corresponding side information on the remote queue manager.	-
z/OS or OS/390 or MVS/ESA using CICS	CKRC (sender) CKSV (requester) CKRC (server)	-
i5/OS	The same as the compare value in the routing entry on the i5/OS system.	-
UNIX systems	The same as in the corresponding side information on the remote queue manager.	mqmtop/bin/amqcrs6a
Windows	As specified in the Windows Run Listener command, or the invocable Transaction Program that was defined using TpSetup on Windows.	mqmtop\bin\amqcrs6a

If you have more than one queue manager on the same machine, ensure that the TPnames in the channel definitions are unique.

Sending end

Create a CPI-C side object (symbolic destination) from the administration application of the LU 6.2 product you are using, and enter this name in the Connection name field in the channel definition. Also create an LU 6.2 link to the partner.

In the CPI-C side object enter the partner LU Name at the receiving machine, the TP Name and the Mode Name. For example:

```
Partner LU Name          OS2R0G2
Partner TP Name          recv
Mode Name                 #INTER
```

Receiving on LU 6.2

Receiving channel programs are started in response to a startup request from the sending channel. To do this, a listener program has to be started to detect incoming network requests and start the associated channel. You start this listener program with the RUNMQLSR command, giving the TpName to listen on. Alternatively, you can use TpStart under SNA Server for Windows.

Using the RUNMQLSR command:

Example of the command to start the listener:

```
RUNMQLSR -t LU62 -n RECV [-m QMNAME]
```

where RECV is the TpName that is specified at the other (sending) end as the "TpName to start on the remote side". The last part in square brackets is optional and is not required for the default queue manager.

It is possible to have more than one queue manager running on one machine. You must assign a different TpName to each queue manager, and then start a listener program for each one. For example:

```
RUNMQLSR -t LU62 -m QM1 -n TpName1
RUNMQLSR -t LU62 -m QM2 -n TpName2
```

For the best performance, run the WebSphere MQ listener as a trusted application as described in "Running channels and listeners as trusted applications" on page 119. See the WebSphere MQ Application Programming Guide for information about trusted applications.

You can stop all WebSphere MQ listeners running on a queue manager that is inactive, using the command:

```
ENDMQLSR [-m QMNAME]
```

If you do not specify a queue manager name, the default queue manager is assumed.

Using Microsoft SNA Server on Windows:

You can use TpSetup (from the SNA Server SDK) to define an invocable TP that then drives amqcrs6a.exe, or you can set various registry values manually. The parameters that should be passed to amqcrs6a.exe are:

```
-m QM -n TpName
```

where *QM* is the Queue Manager name and *TpName* is the TP Name. See the *Microsoft SNA Server APPC Programmers Guide* or the *Microsoft SNA Server CPI-C Programmers Guide* for more information.

Defining a NetBIOS connection

WebSphere MQ uses three types of NetBIOS resource when establishing a NetBIOS connection to another WebSphere MQ product: sessions, commands, and names. Each of these resources has a limit, which is established either by default or by choice during the installation of NetBIOS.

Each running channel, regardless of type, uses one NetBIOS session and one NetBIOS command. The IBM NetBIOS implementation allows multiple processes to use the same local NetBIOS name. Therefore, only one NetBIOS name needs to be available for use by WebSphere MQ. Other vendors' implementations, for example Novell's NetBIOS emulation, require a different local name per process. Verify your requirements from the documentation for the NetBIOS product you are using.

In all cases, ensure that sufficient resources of each type are already available, or increase the maximums specified in the configuration. Any changes to the values will require a system restart.

During system startup, the NetBIOS device driver displays the number of sessions, commands, and names available for use by applications. These resources are available to any NetBIOS-based application that is running on the same system. Therefore, it is possible for other applications to consume these resources before WebSphere MQ needs to acquire them. Your LAN network administrator should be able to clarify this for you.

Defining the WebSphere MQ local NetBIOS name

The local NetBIOS name used by WebSphere MQ channel processes can be specified in three ways. In order of precedence they are:

1. The value specified in the `-l` parameter of the `RUNMQLSR` command, for example:

```
RUNMQLSR -t NETBIOS -l my_station
```

2. The `MQNAME` environment variable whose value is established by the command:

```
SET MQNAME=my_station
```

You can set the `MQNAME` value for each process. Alternatively, you may set it at a system level in the Windows registry.

If you are using a NetBIOS implementation that requires unique names, you must issue a `SET MQNAME` command in each window in which a WebSphere MQ process is started. The `MQNAME` value is arbitrary but it must be unique for each process.

3. The `NETBIOS` stanza in the queue manager configuration file `qm.ini` or in the Windows registry. For example:

```
NETBIOS:
```

```
LocalName=my_station
```

Note:

1. Due to the variations in implementation of the NetBIOS products supported, you are advised to make each NetBIOS name unique in the network. If you do not, unpredictable results may occur. If you have problems establishing a NetBIOS channel and there are error messages in the queue-manager error log showing a NetBIOS return code of X'15', review your use of NetBIOS names.
2. On Windows you cannot use your machine name as the NetBIOS name because Windows already uses it.
3. Sender channel initiation requires that a NetBIOS name be specified either via the MQNAME environment variable or the LocalName in the qm.ini file or in the Windows registry.

Establishing the queue manager NetBIOS session, command, and name limits

The queue manager limits for NetBIOS sessions, commands, and names can be specified in two ways. In order of precedence they are:

1. The values specified in the RUNMQLSR command:

```
-s Sessions
-e Names
-o Commands
```

If the -m operand is not specified in the command, the values will apply only to the default queue manager.

2. The NETBIOS stanza in the queue manager configuration file qm.ini or in the Windows registry. For example:

```
NETBIOS:
    NumSess=Qmgr_max_sess
    NumCmds=Qmgr_max_cmds
    NumNames=Qmgr_max_names
```

Establishing the LAN adapter number

For channels to work successfully across NetBIOS, the adapter support at each end must be compatible. WebSphere MQ allows you to control the choice of LAN adapter (LANA) number by using the AdapterNum value in the NETBIOS stanza of your qm.ini file or the Windows registry and by specifying the -a parameter on the runmqsr command.

The default LAN adapter number used by WebSphere MQ for NetBIOS connections is 0. Verify the number being used on your system as follows:

On Windows, it is not possible to query the LAN adapter number directly through the operating system. Instead, you use the LANACFG.EXE command line utility, available from Microsoft. The output of the tool shows the virtual LAN adapter numbers and their effective bindings. For further information on LAN adapter numbers, see the Microsoft Knowledge Base article 138037 *HOWTO: Use LANA Numbers in a 32-bit Environment*.

Specify the correct value in the NETBIOS stanza of the queue manager configuration file, qm.ini, or the Windows registry:

```
NETBIOS:
    AdapterNum=n
```

where n is the correct LAN adapter number for this system.

Initiating the connection

To initiate the connection, follow these steps at the sending end:

1. Define the NetBIOS station name using the MQNAME or LocalName value as described above.
2. Verify the LAN adapter number being used on your system and specify the correct file using the AdapterNum as described above.
3. In the ConnectionName field of the channel definition, specify the NetBIOS name being used by the target listener program. On Windows, NetBIOS channels *must* be run as threads. Do this by specifying MCATYPE(THREAD) in the channel definition.

```
DEFINE CHANNEL (chname) CHLTYPE(SDR) +  
    TRPTYPE(NETBIOS) +  
    CONNAME(your_station) +  
    XMITQ(xmitq) +  
    MCATYPE(THREAD) +  
    REPLACE
```

Target listener

At the receiving end, follow these steps:

1. Define the NetBIOS station name using the MQNAME or LocalName value as described above.
2. Verify the LAN adapter number being used on your system and specify the correct file using the AdapterNum as described above.
3. Define the receiver channel:

```
DEFINE CHANNEL (chname) CHLTYPE(RCVR) +  
    TRPTYPE(NETBIOS) +  
    REPLACE
```

4. Start the WebSphere MQ listener program to establish the station and make it possible to contact it. For example:

```
RUNMQLSR -t NETBIOS -l your_station [-m qmgr]
```

This command establishes your_station as a NetBIOS station waiting to be contacted. The NetBIOS station name must be unique throughout your NetBIOS network.

For the best performance, run the WebSphere MQ listener as a trusted application as described in "Running channels and listeners as trusted applications" on page 119. See the WebSphere MQ Application Programming Guide for information about trusted applications.

You can stop all WebSphere MQ listeners running on a queue manager that is inactive, using the command:

```
ENDMQLSR [-m QMNAME]
```

If you do not specify a queue manager name, the default queue manager is assumed.

Defining an SPX connection

An SPX connection applies only to a client and server running Windows XP and Windows 2003 Server.

The channel definition at the sending end specifies the address of the target. A listener program must be run at the receiving end.

Sending end

If the target machine is remote, specify the SPX address of the target machine in the Connection name field of the channel definition.

The SPX address is specified in the following form:

network.node(socket)

where:

network

Is the 4-byte network address of the network on which the remote machine resides,

node Is the 6-byte node address, which is the LAN address of the LAN adapter in the remote machine

socket Is the 2-byte socket number on which the remote machine will listen.

If the local and remote machines are on the same network then the network address need not be specified. If the remote end is listening on the default socket (5E86) then the socket need not be specified.

An example of a fully specified SPX address specified in the CONNAME parameter of an MQSC command is:

```
CONNNAME('00000001.08005A7161E5(5E87)')
```

In the default case, where the machines are both on the same network, this becomes:

```
CONNNAME(08005A7161E5)
```

The default socket number may be changed by specifying it in the queue manager configuration file (qm.ini) or the Windows registry:

SPX:

```
Socket=5E87
```

For more information about the values you set using qm.ini or the Windows registry, see Chapter 8, “Configuration file stanzas for distributed queuing,” on page 491.

Receiving on SPX

Receiving channel programs are started in response to a startup request from the sending channel. To do this, a listener program has to be started to detect incoming network requests and start the associated channel.

You should use the WebSphere MQ listener.

Using the SPX listener backlog option:

When receiving on SPX, a maximum number of outstanding connection requests is set. This can be considered a *backlog* of requests waiting on the SPX port for the listener to accept the request. The default listener backlog values are shown in Table 12 on page 128.

Table 12. Default outstanding connection requests on Windows

Platform	Default listener backlog value
Windows Server	5
Windows Workstation	5

If the backlog reaches the values in Table 12, the reason code, MQRC_Q_MGR_NOT_AVAILABLE is received when trying to connect to the queue manager using MQCONN or MQCONNX. If this happens, it is possible to try to connect again.

However, to avoid this error, you can add an entry in the qm.ini file or in the registry for Windows:

```
SPX:  
ListenerBacklog = n
```

This overrides the default maximum number of outstanding requests (see Table 12) for the SPX listener.

Note: Some operating systems support a larger value than the default. If necessary, this can be used to avoid reaching the connection limit.

To run the listener with the backlog option switched on either:

- Use the RUNMQLSR -b command, or
- Use the MQSC command DEFINE LISTENER with the BACKLOG attribute set to the desired value.

For information about the RUNMQLSR command, see the WebSphere MQ System Administration Guide book. For information about the DEFINE LISTENER command, see the WebSphere MQ Script (MQSC) Command Reference.

Using the WebSphere MQ listener:

To run the Listener supplied with WebSphere MQ, that starts new channels as threads, use the RUNMQLSR command. For example:

```
RUNMQLSR -t spx [-m QMNAME] [-x 5E87]
```

The square brackets indicate optional parameters; QMNAME is not required for the default queue manager, and the socket number is not required if you are using the default (5E86).

For the best performance, run the WebSphere MQ listener as a trusted application as described in “Running channels and listeners as trusted applications” on page 119. See the WebSphere MQ Application Programming Guide for information about trusted applications.

You can stop all WebSphere MQ listeners running on a queue manager that is inactive, using the command:

```
ENDMQLSR [-m QMNAME]
```

If you do not specify a queue manager name, the default queue manager is assumed.

IPX/SPX parameters

In most cases the default settings for the IPX/SPX parameters will suit your needs. However, you may need to modify some of them in your environment to tune its use for WebSphere MQ. The actual parameters and the method of changing them varies according to the platform and provider of SPX communications support. The following sections describe some of these parameters, particularly those that may influence the operation of WebSphere MQ channels and client connections.

Windows systems:

Please refer to the Microsoft documentation for full details of the use and setting of the NWLink IPX and SPX parameters. The IPX/SPX parameters are in the following paths in the registry:

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Service\NWLinkSPX\Parameters  
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Service\NWLinkIPX\Parameters
```

Example configuration - IBM WebSphere MQ for Windows

This chapter gives an example of how to set up communication links from WebSphere MQ for Windows to WebSphere MQ products on the following platforms:

- AIX
- HP Tru64 UNIX
- HP-UX
- Solaris
- Linux®
- i5/OS
- z/OS
- VSE/ESA™

This chapter first describes the parameters needed for an LU 6.2 connection, then it guides you through the following tasks:

- “Establishing an LU 6.2 connection” on page 134
- “Establishing a TCP connection” on page 142
- “Establishing a NetBIOS connection” on page 142
- “Establishing an SPX connection” on page 143

Once the connection is established, you need to define some channels to complete the configuration. This is described in “WebSphere MQ for Windows configuration” on page 145.

See “Example configuration chapters in this book” on page 101 for background information about this chapter and how to use it.

Configuration parameters for an LU 6.2 connection

Table 13 on page 130 presents a worksheet listing all the parameters needed to set up communication from Windows to one of the other WebSphere MQ platforms. The worksheet shows examples of the parameters, which have been tested in a working environment, and leaves space for you to fill in your own values. An

explanation of the parameter names follows the worksheet. Use the worksheet in this chapter in conjunction with the worksheet in the chapter for the platform to which you are connecting.

The steps required to set up an LU 6.2 connection are described, with numbered cross references to the parameters on the worksheet. These steps are:

- “Configuring the local node” on page 134
- “Adding a connection” on page 136
- “Adding a partner” on page 139
- “Adding a CPI-C entry” on page 140
- “Configuring an invocable TP” on page 140

Configuration worksheet

Use this worksheet to record the values you use for your configuration. Where numbers appear in the Reference column they indicate that the value must match that in the appropriate worksheet elsewhere in this book. The examples that follow in this chapter refer back to the values in the ID column. The entries in the Parameter Name column are explained in “Explanation of terms” on page 133.

Table 13. Configuration worksheet for IBM Communications Server for Windows systems

ID	Parameter Name	Reference	Example Used	User Value
<i>Definition for local node</i>				
1	Configuration name		NTCONFIG	
2	Network Name		NETID	
3	Control Point Name		WINNTCP	
4	Local Node ID (hex)		05D 30F65	
5	LU Name (local)		WINNTLU	
6	LU Alias (local)		NTQMGR	
7	TP Name		MQSERIES	
8	Command line		c:\Program Files\IBM\WebSphere MQ\bin\amqcrs6a.exe	
9	LAN adapter address		08005AA5FAB9	
<i>Connection to an AIX system</i>				
The values in this section of the table must match those used in Table 17 on page 156, as indicated.				
10	Connection		AIX	
11	Remote Network Address	8	123456789012	
12	Network Name	1	NETID	
13	Control Point Name	2	AIXPU	
14	Remote Node ID	3	071 23456	
15	LU Alias (remote)		AIXQMGR	
16	LU Name	4	AIXLU	
17	Mode	14	#INTER	
18	CPI-C Name		AIXCPIC	
19	Partner TP Name	6	MQSERIES	

Table 13. Configuration worksheet for IBM Communications Server for Windows systems (continued)

ID	Parameter Name	Reference	Example Used	User Value
Connection to an HP-UX system				
The values in this section of the table must match those used in Table 19 on page 172, as indicated.				
10	Connection		HPUX	
11	Remote Network Address	8	100090DC2C7C	
12	Network Name	4	NETID	
13	Control Point Name	2	HPUXPU	
14	Remote Node ID	3	05D 54321	
15	LU Alias (remote)		HPUXQMGR	
16	LU Name	5	HPUXLU	
17	Mode	17	#INTER	
18	CPI-C Name		HPUXCPIC	
19	Partner TP Name	7	MQSERIES	
Connection to a Solaris system				
The values in this section of the table must match those used in Table 21 on page 195, as indicated.				
10	Connection		SOLARIS	
11	Remote Network Address	5	08002071CC8A	
12	Network Name	2	NETID	
13	Control Point Name	3	SOLARPU	
14	Remote Node ID	6	05D 310D6	
15	LU Alias (remote)		SOLARQMGR	
16	LU Name	7	SOLARLU	
17	Mode	17	#INTER	
18	CPI-C Name		SOLCPIC	
19	Partner TP Name	8	MQSERIES	
Connection to a Linux (x86 platform) system				
The values in this section of the table must match those used in Configuration worksheet for Communications Server for Linux, as indicated.				
10	Connection		LINUX	
11	Remote Network Address	8	08005AC6DF33	
12	Network Name	4	NETID	
13	Control Point Name	2	LINUXPU	
14	Remote Node ID	3	05D 30A55	
15	LU Alias (remote)		LXQMGR	
16	LU Name	5	LINUXLU	
17	Mode	6	#INTER	
18	CPI-C Name		LXCPIC	
19	Partner TP Name	7	MQSERIES	
Connection to an i5/OS system				
The values in this section of the table must match those used in Table 35 on page 351, as indicated.				

Table 13. Configuration worksheet for IBM Communications Server for Windows systems (continued)

ID	Parameter Name	Reference	Example Used	User Value
10	Connection		AS400	
11	Remote Network Address	4	10005A5962EF	
12	Network Name	1	NETID	
13	Control Point Name	2	AS400PU	
14	Remote Node ID			
15	LU Alias (remote)		AS400QMGR	
16	LU Name	3	AS400LU	
17	Mode	17	#INTER	
18	CPI-C Name		AS4CPIC	
19	Partner TP Name	8	MQSERIES	
Connection to a z/OS system				
The values in this section of the table must match those used in Table 27 on page 267, as indicated.				
10	Connection		MVS™	
11	Remote Network Address	8	400074511092	
12	Network Name	2	NETID	
13	Control Point Name	3	MVSPU	
14	Remote Node ID			
15	LU Alias (remote)		MVSQMGR	
16	LU Name	4	MVSLU	
17	Mode	10	#INTER	
18	CPI-C Name		MVSCPIC	
19	Partner TP Name	7	MQSERIES	
Connection to a z/OS system using a generic interface				
The values in this section of the table must match those used in Table 27 on page 267, as indicated.				
10	Connection		MVS	
11	Remote Network Address	8	400074511092	
12	Network Name	2	NETID	
13	Control Point Name	3	MVSPU	
14	Remote Node ID			
15	LU Alias (remote)		MVSQMGR	
16	LU Name	10	MVSGR	
17	Mode	6	#INTER	
18	CPI-C Name		MVSCPIC	
19	Partner TP Name	7	MQSERIES	
Connection to a VSE/ESA system				
The values in this section of the table must match those used in your VSE/ESA system.				
10	Connection		MVS	
11	Remote Network Address	5	400074511092	
12	Network Name	1	NETID	

Table 13. Configuration worksheet for IBM Communications Server for Windows systems (continued)

ID	Parameter Name	Reference	Example Used	User Value
13	Control Point Name	2	VSEPU	
14	Remote Node ID			
15	LU Alias (remote)		VSEQMGR	
16	LU Name	3	VSELU	
17	Mode		#INTER	
18	CPI-C Name		VSECPIC	
19	Partner TP Name	4	MQ01	MQ01

Explanation of terms

1 Configuration Name

This is the name of the file in which the Communications Server configuration is saved.

2 Network Name

This is the unique ID of the network to which you are connected. It is an alphanumeric value and can be 1-8 characters long. The network name works with the Control Point Name to uniquely identify a system. Your network administrator will tell you the value.

3 Control Point Name

In Advanced Peer-to-Peer Networking® (APPN), a control point is responsible for managing a node and its resources. A control point is also a logical unit (LU). The Control Point Name is the name of the LU and is assigned to your system by the network administrator.

4 Local Node ID (hex)

Some SNA products require partner systems to specify a node identifier that uniquely identifies their workstation. The two systems exchange this node identifier in a message unit called the exchange identifier (XID). Your network administrator will assign this ID for you.

5 LU Name (local)

A logical unit (LU) is software that serves as an interface or translator between a transaction program and the network. An LU manages the exchange of data between transaction programs. The local LU Name is the name of the LU on your workstation. Your network administrator will assign this to you.

6 LU Alias (local)

The name by which your local LU will be known to your applications. You choose this name yourself. It can be 1-8 characters long.

7 TP Name

WebSphere MQ applications trying to converse with your workstation specify a symbolic name for the program that is to start running. This will have been defined on the channel definition at the sender. For simplicity, wherever possible use a transaction program name of MQSERIES, or in the case of a connection to VSE/ESA, where the length is limited to 4 bytes, use MQTP.

See Table 11 on page 122 for more information.

8 Command line

This is the path and name of the actual program to be run when a

conversation has been initiated with your workstation. The example shown on the worksheet assumes that WebSphere MQ is installed in the default directory, c:\Program Files\IBM\WebSphere MQ. The configuration pairs this name with the symbolic name 7 when you use TPSETUP (which is part of the SNA Server software developers kit).

9 LAN adapter address

This is the address of your token-ring card. To discover this type **net config server** at a command prompt. The address appears in the output. For example:

```
Server is active on 08005AA5FAB9
```

10 Connection

This is a meaningful symbolic name by which the connection to a partner node is known. It is used only within SNA Server administration and is specified by you.

15 LU Alias (remote)

This is a value known only in this server and is used to represent the fully qualified partner LU name. You supply the value.

17 Mode

This is the name given to the set of parameters that control the APPC conversation. An entry with this name and a similar set of parameters must be defined at each partner system. Your network administrator will tell you this name.

18 CPI-C Name

This is the name given to a locally held definition of a partner application. You supply the name and it must be unique within this server. The name is specified in the CONNAME attribute of the WebSphere MQ sender channel definition.

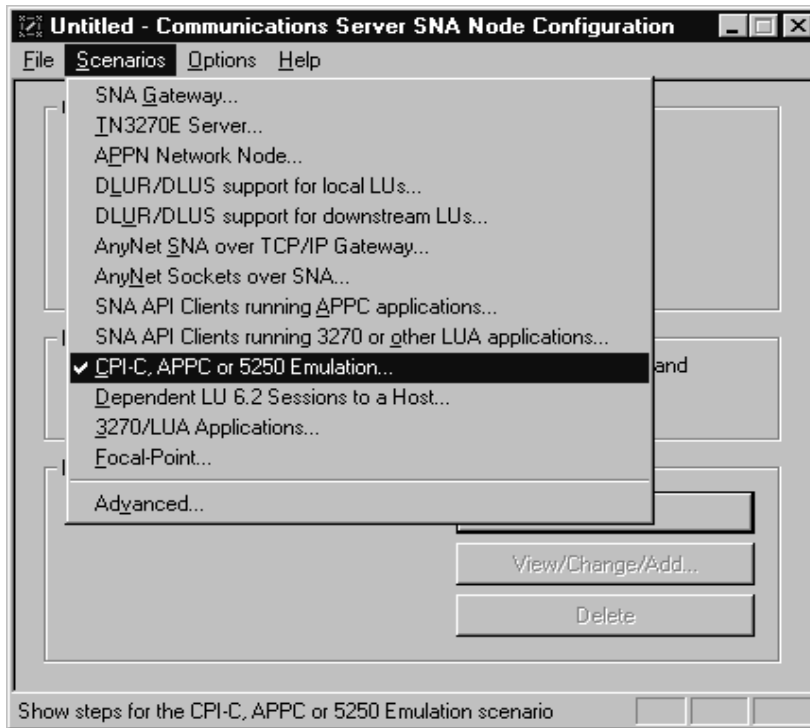
Establishing an LU 6.2 connection

This section describes how to establish an LU 6.2 connection using IBM Communications Server for Windows NT, Version 5.0. You may use any of the supported LU 6.2 products for this platform. The panels of other products will not be identical to those shown here, but most of their content will be similar.

Configuring the local node

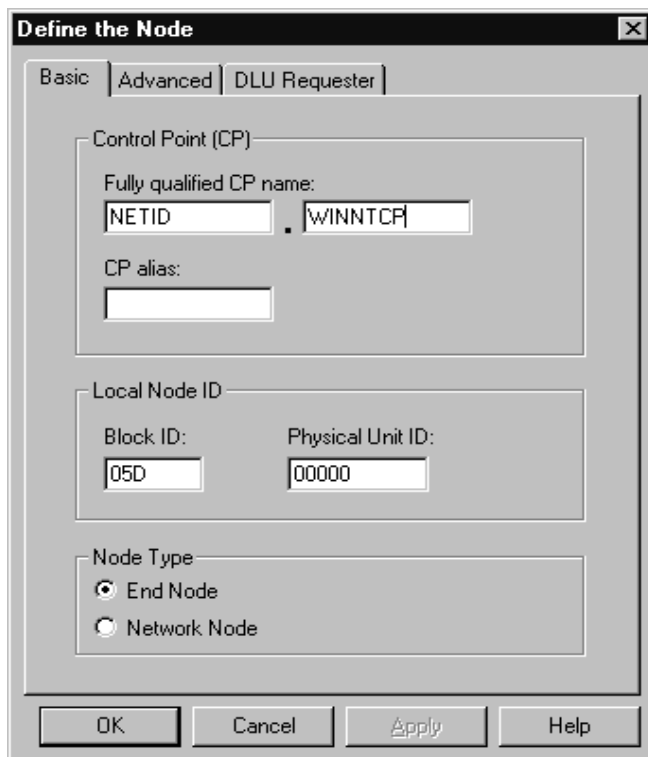
To configure the local node, follow these steps:

1. From the **Scenarios** pull-down of the Communications Server SNA Node Configuration window, select the **CPI-C, APPC or 5250 Emulation** scenario.



The CPI-C, APPC or 5250 Emulation scenario window is displayed.

2. Click on **Configure Node**, then click on **New**. The Define the Node property sheet is displayed.



3. In the **Fully qualified CP name** field on the Basic page, enter the unique ID of the network to which you are connected (2) and the control point name (3). Click on **OK** to continue.
4. From the SNA Node Configuration window, click on **Configure Local LU 6.2**, then click on **New**. The Define a Local LU 6.2 window is displayed.

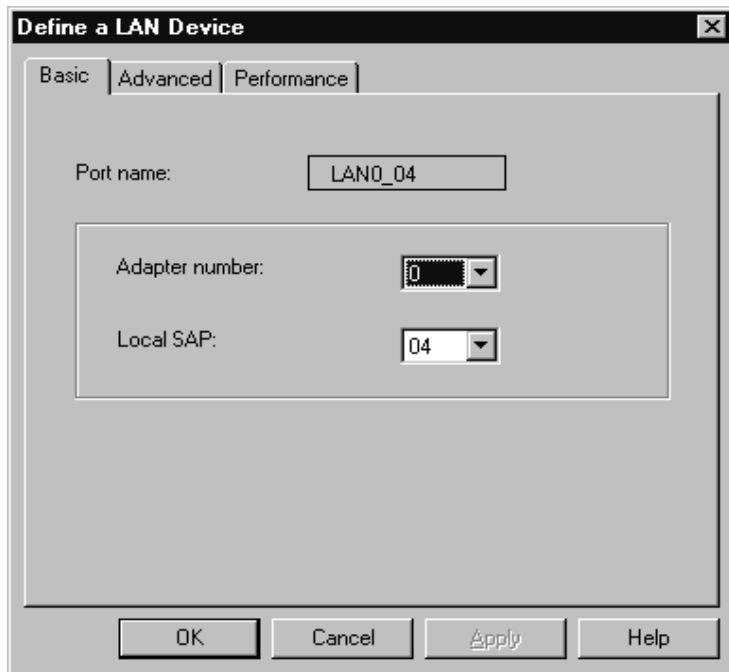
The screenshot shows a dialog box titled "Define a Local LU 6.2" with a close button in the top right corner. The "Basic" tab is active. The "Local LU name:" field contains the text "WINNTLU". Below it are two unchecked checkboxes: "Dependent LU" and "SNA API client use". The "Local LU alias:" field contains the text "NTQMGR". The "PU name:" field is a dropdown menu, and the "NAU address:" field is also a dropdown menu. The "LU session limit:" field contains the text "0". At the bottom of the dialog box are four buttons: "OK", "Cancel", "Apply", and "Help".

5. In the **Local LU name** field on the Basic page, enter the name of the LU on your workstation (5). In the **Local LU alias** field, enter the name by which your local LU will be known to your applications (6). Click on **OK** to continue.

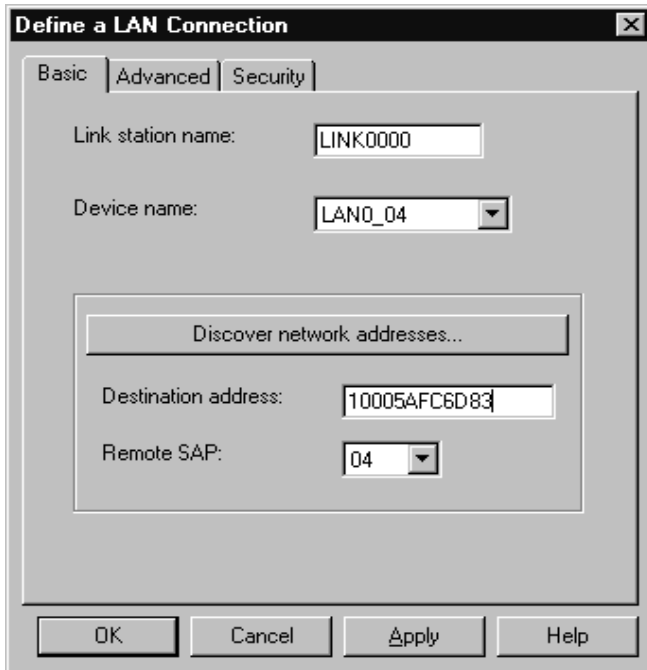
Adding a connection

To add a connection, follow these steps:

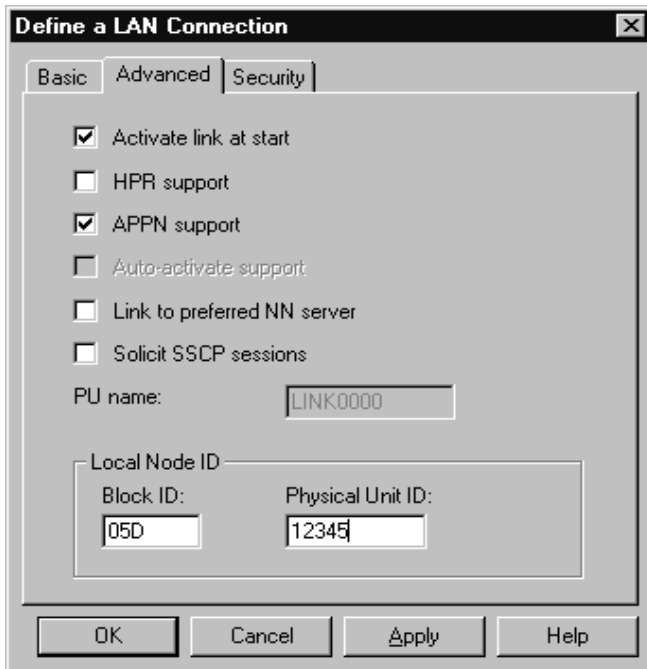
1. From the SNA Node Configuration window, select **Configure Devices**, select **LAN** as the DLC type, then click on **New**. The Define a LAN Device property sheet is displayed.



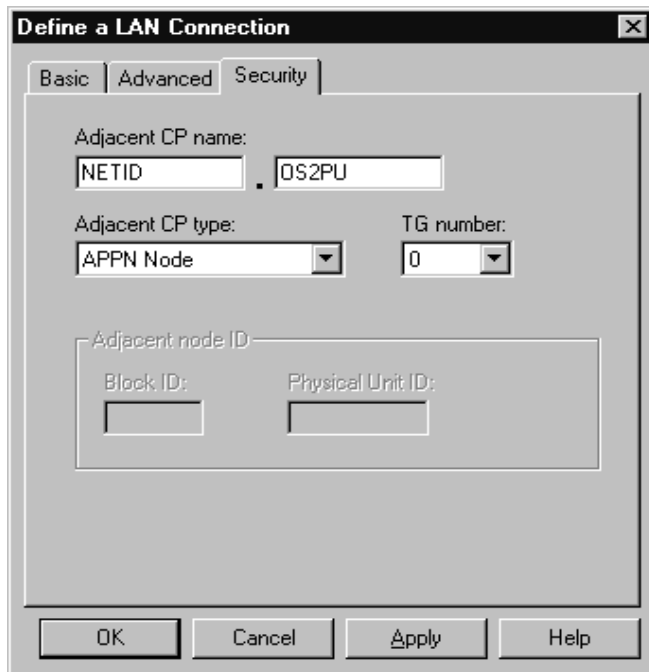
2. If you have the LLC2 protocol installed with Communications Server for Windows NT, the Adapter number list box lists the available LAN adapters. See the help file INLLC40.HLP (Windows NT 4.0) or INLLC35.HLP (Windows NT 3.51) in the Communications Server installation directory for LLC2 installation instructions.
3. The default values displayed on the Define a LAN Device Basic page may be accepted. Click on **OK** to continue.
4. From the SNA Node Configuration window, select **Configure Connections**, select **LAN** as the DLC type, then click on **New**. The Define a LAN Connection property sheet is displayed.



5. In the **Destination address** field on the Basic page, enter the LAN address of the system to which you are connecting (11). Select the Advanced page.



6. In the **Block ID** field on the Advanced page, enter the local node ID (hex) (4). Select the Security page.

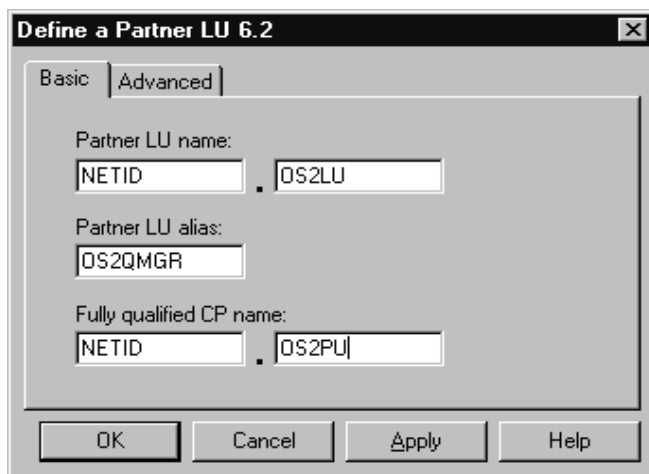


7. In the **Adjacent CP name** field on the Security page, enter the network name and control point name of the remote node (12 and 13). In the **Adjacent CP type** field, enter APPN Node. You do not need to complete the **Adjacent node ID** field for a peer-to-peer connection. Click on **OK** to continue. Take note of the default link name used to identify this new definition (for example, LINK0000).

Adding a partner

To add a partner LU definition, follow these steps:

1. From the SNA Node Configuration window, select **Configure Partner LU 6.2**, then click on **New**. The Define a Partner LU 6.2 property sheet is displayed.

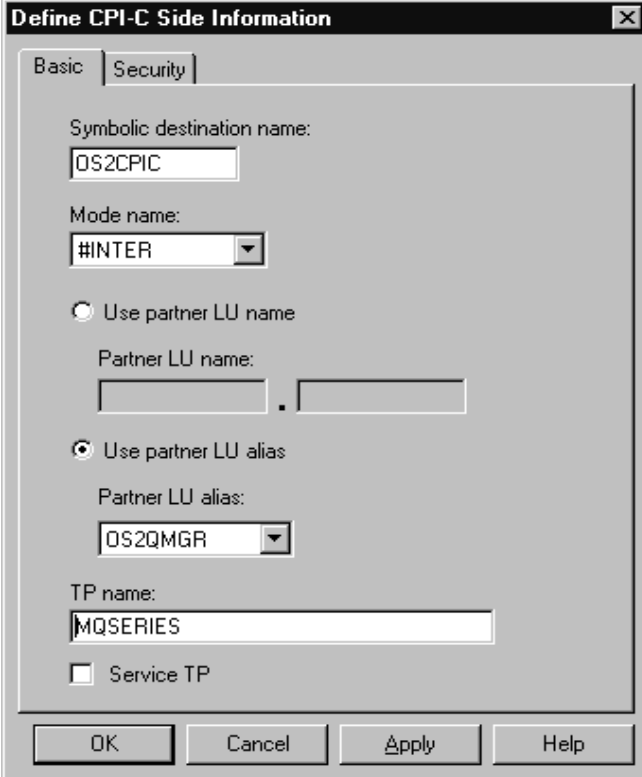


2. In the **Partner LU name** field on the Basic page, enter the network name (12) and LU name of the remote system (16). In the **Partner LU alias** field, enter the remote LU alias (15). In the **Fully qualified CP name** fields, enter the network name and control point name of the remote system (12 and 13). Click on **OK** to continue.

Adding a CPI-C entry

To add a CPI-C Side information entry, follow these steps:

1. From the SNA Node Configuration window, select **Configure CPI-C Side Information**, then click on **New**. The Define a CPI-C Side Information property sheet is displayed.



The screenshot shows a dialog box titled "Define CPI-C Side Information" with a close button (X) in the top right corner. The dialog has two tabs: "Basic" (selected) and "Security". The "Basic" tab contains the following fields and options:

- Symbolic destination name:** A text box containing "OS2CPIC".
- Mode name:** A dropdown menu showing "#INTER".
- Use partner LU name:** An unselected radio button.
- Partner LU name:** Two empty text boxes separated by a period.
- Use partner LU alias:** A selected radio button.
- Partner LU alias:** A dropdown menu showing "OS2QMGR".
- TP name:** A text box containing "MQSERIES".
- Service TP:** An unselected checkbox.

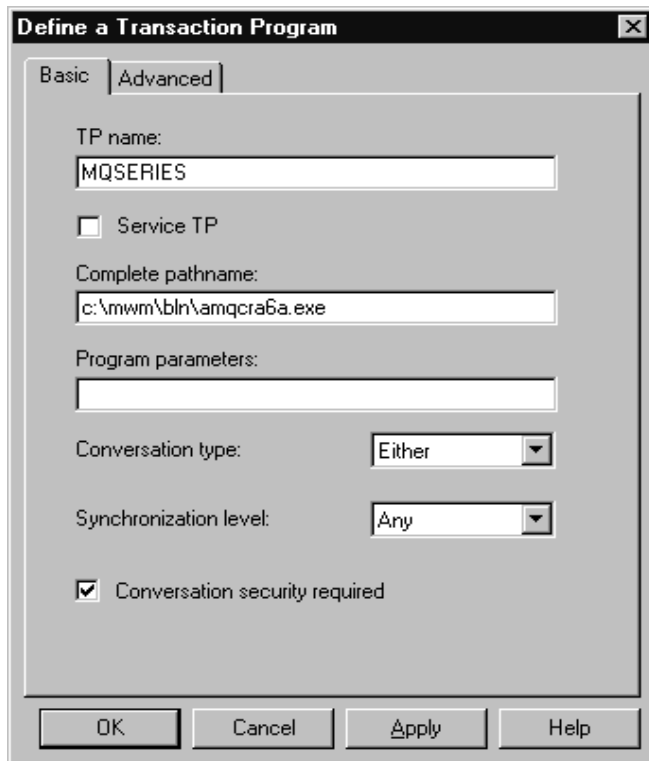
At the bottom of the dialog are four buttons: "OK", "Cancel", "Apply", and "Help".

2. In the **Symbolic destination name** field of the Basic page, enter the CPI-C name (18). In the **Mode name** field, enter the mode value (17). Enter either a **fully qualified partner LU name** (12.16) or a **partner LU alias** (15) depending on what you choose in the CPI-C Side Information property sheet. In the **TP name** field, enter the partner TP name (19). Click on **OK** to continue.

Configuring an invocable TP

To add a Transaction Program (TP) definition, follow these steps:

1. From the SNA Node Configuration window, select **Configure Transaction Programs**, then click on **New**. The Define a Transaction Program property sheet is displayed.



2. In the **TP name** field on the Basic page, enter the transaction program name (7). In the **Complete pathname** field, enter the actual path and name of the program that will be run when a conversation is initiated with your workstation (8). When you are happy with the settings, click on **OK** to continue.
3. In order to be able to stop the WebSphere MQ Transaction Program, you need to start it in one of the following ways:
 - a. Check **Service TP** on the Basic page. This starts the TP programs at Windows startup and will run the programs under the system user ID.
 - b. Check **Dynamically loaded** on the Advanced page. This dynamically loads and starts the programs as and when incoming SNA conversation requests arrive. It will run the programs under the same user ID as the rest of WebSphere MQ.

Note: To use dynamic loading, it is necessary to vary the user ID under which the WebSphere MQ SNA Transaction program runs. To do this, set the Attach Manager to run under the desired user context by modifying the startup parameters within the Control Panel in the Services applet for the AppnNode service.

 - c. Issue the WebSphere MQ command, runmqslr, to run the channel listener process.

Communications Server has a tuning parameter called the Receive_Allocate timeout parameter that is set in the Transaction Program. The default value of this parameter is 3600 and this indicates that the listener will only remain active for 3600 seconds, that is, 1 hour. You can make your listener run for longer than this by increasing the value of the Receive_Allocate timeout parameter. You can also make it run 'forever' by specifying zero.

What next?

The SNA configuration task is complete. From the **File** pull-down, select **Save** and specify a file name under which to save your SNA configuration information, for example, NTCONFIG (1). When prompted, select this configuration as the default.

From the SNA Node Operations application, start the node by clicking the **Start node** button on the toolbar. Specify the file name of the configuration you just saved. (It should appear in the file-name box by default, because you identified it as your default configuration.) When the node startup is complete, ensure that your link to the remote node has been established by selecting the **Connections** button on the toolbar, then find the link name you configured (for example, LINK0000). The link should be active if the remote node is active waiting for the link to be established.

A complementary SNA setup process is required on the node to which you are connecting before you can attempt WebSphere MQ server-to-server message transmissions.

The LU 6.2 connection is now established. You are ready to complete the configuration. Go to “WebSphere MQ for Windows configuration” on page 145.

Establishing a TCP connection

The TCP stack that is shipped with Windows systems does not include an *inet* daemon or equivalent.

The WebSphere MQ command used to start the WebSphere MQ for TCP listener is:

```
runmq1sr -t tcp
```

The listener must be started explicitly before any channels are started. It enables receiving channels to start automatically in response to a request from an inbound sending channel.

What next?

When the TCP/IP connection is established, you are ready to complete the configuration. Go to “WebSphere MQ for Windows configuration” on page 145.

Establishing a NetBIOS connection

A NetBIOS connection is initiated from a queue manager that uses the ConnectionName parameter on its channel definition to connect to a target listener. To set up a NetBIOS connection, follow these steps:

1. At each end of the channel specify the local NetBIOS name to be used by the WebSphere MQ channel processes, in the Windows registry or in the queue manager configuration file qm.ini. For example, the NETBIOS stanza in the Windows registry at the sending end might look like this:

```
NETBIOS:  
LocalName=WNTNETB1
```

and at the receiving end:

```
NETBIOS:  
LocalName=WNTNETB2
```

Each WebSphere MQ process must use a different local NetBIOS name. Do not use your machine name as the NetBIOS name because Windows already uses it.

2. At each end of the channel, verify the LAN adapter number being used on your system. The WebSphere MQ for Windows default for logical adapter number 0 is NetBIOS running over a TCP/IP network. To use native NetBIOS you need to select logical adapter number 1. See “Establishing the LAN adapter number” on page 125.

Specify the correct LAN adapter number in the NETBIOS stanza of the the Windows registry. For example:

```
NETBIOS:  
AdapterNum=1
```

3. So that sender channel initiation will work, specify the local NetBIOS name via the MQNAME environment variable:

```
SET MQNAME=WNTNETB1I
```

This name must be unique.

4. At the sending end, define a channel specifying the NetBIOS name being used at the other end of the channel. For example:

```
DEFINE CHANNEL (WINNT.OS2.NET) CHLTYPE(SDR) +  
TRPTYPE(NETBIOS) +  
CONNNAME(WNTNETB2) +  
XMITQ(OS2) +  
MCATYPE(THREAD) +  
REPLACE
```

You must specify the option MCATYPE(THREAD) because, on Windows, sender channels must be run as threads.

5. At the receiving end, define the corresponding receiver channel. For example:

```
DEFINE CHANNEL (WINNT.OS2.NET) CHLTYPE(RCVR) +  
TRPTYPE(NETBIOS) +  
REPLACE
```

6. Start the channel initiator because each new channel is started as a thread rather than as a new process.

```
runmqchi
```

7. At the receiving end, start the WebSphere MQ listener:

```
runmqlsr -t netbios
```

Optionally you may specify values for the queue manager name, NetBIOS local name, number of sessions, number of names, and number of commands. See “Defining a NetBIOS connection” on page 124 for more information about setting up NetBIOS connections.

Establishing an SPX connection

An SPX connection applies only to a client and server running Windows XP and Windows 2003 Server.

This section discusses the following topics:

- IPX/SPX parameters
- SPX addressing
- Receiving on SPX

IPX/SPX parameters

Please refer to the Microsoft documentation for full details of the use and setting of the NWLink IPX and SPX parameters. The IPX/SPX parameters are in the following paths in the registry:

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Service\NWLinkSPX\Parameters
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Service\NWLinkIPX\Parameters
```

SPX addressing

WebSphere MQ uses the SPX address of each machine to establish connectivity. The SPX address is specified in the following form:

network.node(socket)

where

network

Is the 4-byte network address of the network on which the remote machine resides,

node Is the 6-byte node address, which is the LAN address of the LAN adapter in the remote machine

socket Is the 2-byte socket number on which the remote machine will listen.

The default socket number used by WebSphere MQ is 5E86. You can change the default socket number by specifying it in the the Windows registry or in the queue manager configuration file qm.ini. The lines in the Windows registry might read:

```
SPX:
SOCKET=n
```

For more information about values you can set in qm.ini, see Chapter 8, “Configuration file stanzas for distributed queuing,” on page 491.

The SPX address is later specified in the CONNAME parameter of the sender channel definition. If the WebSphere MQ systems being connected reside on the same network, the network address need not be specified. Similarly, if the remote system is listening on the default socket number (5E86), it need not be specified. A fully qualified SPX address in the CONNAME parameter would be:

```
CONNNAME('network.node(socket)')
```

but if the systems reside on the same network and the default socket number is used, the parameter would be:

```
CONNNAME(node)
```

A detailed example of the channel configuration parameters is given in “WebSphere MQ for Windows configuration” on page 145.

Receiving on SPX

Receiving channel programs are started in response to a startup request from the sending channel. To do this, a listener program has to be started to detect incoming network requests and start the associated channel.

You should use the WebSphere MQ listener.

Using the WebSphere MQ listener:

To run the Listener supplied with WebSphere MQ, that starts new channels as threads, use the RUNMQLSR command. For example:

```
RUNMQLSR -t spx
```

Optionally you may specify the queue manager name or the socket number if you are not using the defaults.

WebSphere MQ for Windows configuration

Note:

1. You can use the sample program, AMQSBCG, to display the contents and headers of all the messages in a queue. For example:

```
AMQSBCG q_name qmgr_name
```

displays the contents of the queue *q_name* defined in queue manager *qmgr_name*. Alternatively, you can use the message browser in the WebSphere MQ Explorer.

2. You can start any channel from the command prompt using the command

```
runmqchl -c channel.name
```
3. Error logs can be found in the directories *mqmtop\qmgrs\qmgrname\errors* and *mqmtop\qmgrs\@system\errors*. In both cases, the most recent messages are at the end of *amqerr01.log*.
4. When you are using the command interpreter **runmqsc** to enter administration commands, a + at the end of a line indicates that the next line is a continuation. Ensure that there is a space between the last parameter and the continuation character.

Default configuration

You can create a default configuration by using the WebSphere MQ Postcard application to guide you through the process. For information about this, see the WebSphere MQ System Administration Guide book.

Basic configuration

You can create and start a queue manager from the WebSphere MQ Explorer or from the command prompt.

If you choose the command prompt:

1. Create the queue manager using the command:

```
crtmqm -u dlqname -q winnt
```

where:

winnt Is the name of the queue manager

-q Indicates that this is to become the default queue manager

-u dlqname

Specifies the name of the undeliverable message queue

This command creates a queue manager and a set of default objects.

2. Start the queue manager using the command:

```
strmqm winnt
```

where *winnt* is the name given to the queue manager when it was created.

Channel configuration

The following sections detail the configuration to be performed on the Windows queue manager to implement the channel described in Figure 32 on page 101.

In each case the MQSC command is shown. Either start **runmqsc** from a command prompt and enter each command in turn, or build the commands into a command file.

Examples are given for connecting WebSphere MQ for Windows and WebSphere MQ for AIX. If you wish to connect to WebSphere MQ on another platform use the appropriate set of values from the table in place of those for Windows.

Note: The words in **bold** are user-specified and reflect the names of WebSphere MQ objects used throughout these examples. If you change the names used here, ensure that you also change the other references made to these objects throughout this book. All others are keywords and should be entered as shown.

Table 14. Configuration worksheet for WebSphere MQ for Windows

	Parameter Name	Reference	Example Used	User Value
<i>Definition for local node</i>				
A	Queue Manager Name		WINNT	
B	Local queue name		WINNT.LOCALQ	
<i>Connection to WebSphere MQ for AIX</i>				
The values in this section of the table must match those used in Table 18 on page 168, as indicated.				
C	Remote queue manager name	A	AIX	
D	Remote queue name		AIX.REMOTEQ	
E	Queue name at remote system	B	AIX.LOCALQ	
F	Transmission queue name		AIX	
G	Sender (SNA) channel name		WINNT.AIX.SNA	
H	Sender (TCP) channel name		WINNT.AIX.TCP	
I	Receiver (SNA) channel name	G	AIX.WINNT.SNA	
J	Receiver (TCP) channel name	H	AIX.WINNT.TCP	
<i>Connection to MQSeries for HP Tru64 UNIX</i>				
The values in this section of the table must match those used in your HP Tru64 UNIX system.				
C	Remote queue manager name	A	DECUX	
D	Remote queue name		DECUX.REMOTEQ	
E	Queue name at remote system	B	DECUX.LOCALQ	
F	Transmission queue name		DECUX	
H	Sender (TCP) channel name		DECUX.WINNT.TCP	
J	Receiver (TCP) channel name	H	WINNT.DECUX.TCP	
<i>Connection to WebSphere MQ for HP-UX</i>				
The values in this section of the table must match those used in Table 20 on page 191, as indicated.				
C	Remote queue manager name	A	HPUX	
D	Remote queue name		HPUX.REMOTEQ	
E	Queue name at remote system	B	HPUX.LOCALQ	

Table 14. Configuration worksheet for WebSphere MQ for Windows (continued)

	Parameter Name	Reference	Example Used	User Value
F	Transmission queue name		HPUX	
G	Sender (SNA) channel name		WINNT.HPUX.SNA	
H	Sender (TCP) channel name		WINNT.HPUX.TCP	
I	Receiver (SNA) channel name	G	HPUX.WINNT.SNA	
J	Receiver (TCP/IP) channel name	H	HPUX.WINNT.TCP	
Connection to WebSphere MQ for Solaris				
The values in this section of the table must match those used in Table 22 on page 212, as indicated.				
C	Remote queue manager name	A	SOLARIS	
D	Remote queue name		SOLARIS.REMOTEQ	
E	Queue name at remote system	B	SOLARIS.LOCALQ	
F	Transmission queue name		SOLARIS	
G	Sender (SNA) channel name		WINNT.SOLARIS.SNA	
H	Sender (TCP) channel name		WINNT.SOLARIS.TCP	
I	Receiver (SNA) channel name	G	SOLARIS.WINNT.SNA	
J	Receiver (TCP) channel name	H	SOLARIS.WINNT.TCP	
Connection to WebSphere MQ for Linux				
The values in this section of the table must match those used in Table 24 on page 235, as indicated.				
C	Remote queue manager name	A	LINUX	
D	Remote queue name		LINUX.REMOTEQ	
E	Queue name at remote system	B	LINUX.LOCALQ	
F	Transmission queue name		LINUX	
G	Sender (SNA) channel name		WINNT.LINUX.SNA	
H	Sender (TCP) channel name		WINNT.LINUX.TCP	
I	Receiver (SNA) channel name	G	LINUX.WINNT.SNA	
J	Receiver (TCP) channel name	H	LINUX.WINNT.TCP	
Connection to WebSphere MQ for i5/OS				
The values in this section of the table must match those used in Table 36 on page 364, as indicated.				
C	Remote queue manager name	A	AS400	
D	Remote queue name		AS400.REMOTEQ	
E	Queue name at remote system	B	AS400.LOCALQ	
F	Transmission queue name		AS400	
G	Sender (SNA) channel name		WINNT.AS400.SNA	
H	Sender (TCP) channel name		WINNT.AS400.TCP	
I	Receiver (SNA) channel name	G	AS400.WINNT.SNA	
J	Receiver (TCP) channel name	H	AS400.WINNT.TCP	
Connection to WebSphere MQ for z/OS				
The values in this section of the table must match those used in Table 28 on page 272, as indicated.				
C	Remote queue manager name	A	MVS	
D	Remote queue name		MVS.REMOTEQ	

Table 14. Configuration worksheet for WebSphere MQ for Windows (continued)

	Parameter Name	Reference	Example Used	User Value
E	Queue name at remote system	B	MVS.LOCALQ	
F	Transmission queue name		MVS	
G	Sender (SNA) channel name		WINNT.MVS.SNA	
H	Sender (TCP) channel name		WINNT.MVS.TCP	
I	Receiver (SNA) channel name	G	MVS.WINNT.SNA	
J	Receiver (TCP/IP) channel name	H	MVS.WINNT.TCP	
Connection to WebSphere MQ for z/OS using queue-sharing groups				
The values in this section of the table must match those used in Table 30 on page 292, as indicated.				
C	Remote queue manager name	A	QSG	
D	Remote queue name		QSG.REMOTEQ	
E	Queue name at remote system	B	QSG.SHAREDQ	
F	Transmission queue name		QSG	
G	Sender (SNA) channel name		WINNT.QSG.SNA	
H	Sender (TCP) channel name		WINNT.QSG.TCP	
I	Receiver (SNA) channel name	G	QSG.WINNT.SNA	
J	Receiver (TCP/IP) channel name	H	QSG.WINNT.TCP	
Connection to MQSeries for VSE/ESA				
The values in this section of the table must match those used in your VSE/ESA system.				
C	Remote queue manager name	A	VSE	
D	Remote queue name		VSE.REMOTEQ	
E	Queue name at remote system	B	VSE.LOCALQ	
F	Transmission queue name		VSE	
G	Sender channel name		WINNT.VSE.SNA	
I	Receiver channel name	G	VSE.WINNT.SNA	

WebSphere MQ for Windows sender-channel definitions using SNA:

```

def ql (AIX) +                                     F
  usage(xmitq) +
  replace

def qr (AIX.REMOTEQ) +                             D
  rname(AIX.LOCALQ) +                             E
  rqmname(AIX) +                                   C
  xmitq(AIX) +                                     F
  replace

def chl (WINNT.AIX.SNA) chltype(sdr) +            G
  trptype(lu62) +
  conname(AIXCPIC) +                               18
  xmitq(AIX) +                                     F
  replace

```

WebSphere MQ for Windows receiver-channel definitions using SNA:


```
def ql (WINNT.LOCALQ) replace B
def chl (AIX.WINNT.SNA) chltype(rcvr) + I
  trptype(lu62) +
  replace
```

WebSphere MQ for Windows sender-channel definitions using TCP/IP:

```
def ql (AIX) + F
  usage(xmitq) +
  replace

def qr (AIX.REMOTEQ) + D
  rname(AIX.LOCALQ) + E
  rqmname(AIX) + C
  xmitq(AIX) + F
  replace

def chl (WINNT.AIX.TCP) chltype(sdr) + H
  trptype(tcp) +
  conname(remote_tcpip_hostname) +
  xmitq(AIX) + F
  replace
```

WebSphere MQ for Windows receiver-channel definitions using TCP:

```
def ql (WINNT.LOCALQ) replace B
def chl (AIX.WINNT.TCP) chltype(rcvr) + J
  trptype(tcp) +
  replace
```

Automatic startup

WebSphere MQ for Windows allows you to automate the startup of a queue manager and its channel initiator, channels, listeners, and command servers. Use the IBM WebSphere MQ Services snap-in to define the services for the queue manager. When you have successfully completed testing of your communications setup, set the relevant services to **automatic** within the snap-in. This file can be read by the supplied WebSphere MQ service when the system is started.

For more information about this, see the WebSphere MQ System Administration Guide book.

Running channels as processes or threads

WebSphere MQ for Windows provides the flexibility to run sending channels as Windows processes or Windows threads. This is specified in the MCATYPE parameter on the sender channel definition.

Most installations will select to run their sending channels as threads, because the virtual and real memory required to support a large number of concurrent channel connections will be reduced. However, a NetBIOS connection needs a separate process for the sending Message Channel Agent.

Multiple thread support — pipelining

You can optionally allow a message channel agent (MCA) to transfer messages using multiple threads. This process, called *pipelining*, enables the MCA to transfer messages more efficiently, with fewer wait states, which improves channel performance. Each MCA is limited to a maximum of two threads.

You control pipelining with the *PipeLineLength* parameter in the *qm.ini* file. This parameter is added to the CHANNELS stanza:

PipeLineLength=1|number

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

With WebSphere MQ for Windows, use the WebSphere MQ Explorer to set the *PipeLineLength* parameter in the registry. Refer to the WebSphere MQ System Administration Guide book for a complete description of the CHANNELS stanza.

Note:

1. *PipeLineLength* applies only to V5.2 or later products.
2. Pipelining is effective only for TCP/IP channels.

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

Channel exit considerations:

Note that pipelining can cause some exit programs to fail, because:

- Exits might not be called serially.
- Exits might be called alternately from different threads.

Check the design of your exit programs before you use pipelining:

- Exits must be reentrant at all stages of their execution.
- When you use MQI calls, remember that you cannot use the same MQI handle when the exit is invoked from different threads.

Consider a message exit that opens a queue and uses its handle for MQPUT calls on all subsequent invocations of the exit. This fails in pipelining mode because the exit is called from different threads. To avoid this failure, keep a queue handle for each thread and check the thread identifier each time the exit is invoked.

Setting up communication on UNIX systems

When a distributed-queuing management channel is started, it tries to use the connection specified in the channel definition. For this to succeed, it is necessary for the connection to be defined and available. This chapter explains how to do this. You might also find it helpful to refer to the following chapters:

- “Example configuration - IBM WebSphere MQ for AIX” on page 155
- “Example configuration - IBM WebSphere MQ for HP-UX” on page 172
- “Example configuration - IBM WebSphere MQ for Solaris” on page 194
- “Example configuration - IBM WebSphere MQ for Linux” on page 215

For Windows, see “Setting up communication for Windows” on page 120.

Deciding on a connection

There are two forms of communication for WebSphere MQ on UNIX systems:

- TCP
- LU 6.2

Each channel definition must specify one only as the transmission protocol (Transport Type) attribute. One or more protocols may be used by a queue manager.

For WebSphere MQ clients, it may be useful to have alternative channels using different transmission protocols. See the WebSphere MQ Clients book.

Defining a TCP connection

The channel definition at the sending end specifies the address of the target. The listener or inet daemon is configured for the connection at the receiving end.

Sending end

Specify the host name, or the TCP address of the target machine, in the Connection Name field of the channel definition. The port to connect to will default to 1414. Port number 1414 is assigned by the Internet Assigned Numbers Authority to WebSphere MQ.

To use a port number other than the default, change the connection name field thus:

```
Connection Name REMHOST(1822)
```

where REMHOST is the hostname of the remote machine and 1822 is the port number required. (This must be the port that the listener at the receiving end is listening on.)

Alternatively you can change the port number by specifying it in the queue manager configuration file (qm.ini):

```
TCP:  
  Port=1822
```

For more information about the values you set using qm.ini, see Chapter 8, "Configuration file stanzas for distributed queuing," on page 491.

Receiving on TCP

You can use either the TCP/IP listener, which is the inet daemon (inetd), or the WebSphere MQ listener.

Some Linux distributions now use the extended inet daemon (xinetd) instead of the inet daemon. For information about how to use the extended inet daemon on a Linux system, see "Using the extended inet daemon (XINETD)" on page 233.

Using the TCP/IP listener:

To start channels on UNIX, the /etc/services file and the inetd.conf file must be edited, following the instructions below:

1. Edit the /etc/services file:

Note: To edit the /etc/services file, you must be logged in as a superuser or root. You can change this, but it must match the port number specified at the sending end.

Add the following line to the file:

```
MQSeries 1414/tcp
```

where 1414 is the port number required by WebSphere MQ.

2. Add a line in the `inetd.conf` file to call the program `amqcrsta`:
`MQSeries stream tcp nowait mqm /mqmtop/bin/amqcrsta amqcrsta`
`[-m Queue_Man_Name]`

The updates are active after `inetd` has reread the configuration files. To do this, issue the following commands from the root user ID:

- On AIX:
`refresh -s inetd`
- On HP-UX, from the `mqm` user ID:
`inetd -c`
- On Solaris 10 or later:
`inetconv`
- On other UNIX systems (including Solaris 9):
`kill -1 <process number>`

When the listener program started by `inetd` inherits the locale from `inetd`, it is possible that the MQMDE will not be honored (merged) and will be placed on the queue as message data. To ensure that the MQMDE is honored, you must set the locale correctly. The locale set by `inetd` may not match that chosen for other locales used by WebSphere MQ processes. To set the locale:

1. Create a shell script which sets the locale environment variables `LANG`, `LC_COLLATE`, `LC_CTYPE`, `LC_MONETARY`, `LC_NUMERIC`, `LC_TIME`, and `LC_MESSAGES` to the locale used for other WebSphere MQ process.
2. In the same shell script, call the listener program.
3. Modify the `inetd.conf` file to call your shell script in place of the listener program.

It is possible to have more than one queue manager on the server machine. You must add a line to each of the two files, as above, for each of the queue managers. For example:

```
MQSeries1    1414/tcp
MQSeries2    1822/tcp
MQSeries2 stream tcp nowait mqm /mqmtop/bin/amqcrsta amqcrsta -m QM2
```

This avoids error messages being generated if there is a limitation on the number of outstanding connection requests queued at a single TCP port. For information about the number of outstanding connection requests, see “Using the TCP listener backlog option.”

Using the TCP listener backlog option:

In TCP, connections are treated incomplete unless three-way handshake takes place between the server and the client. These connections are called outstanding connection requests. A maximum value is set for these outstanding connection requests. This can be considered a backlog of requests waiting on the TCP port for the listener to accept the request. The default listener backlog values are shown in Table 15.

Table 15. Maximum outstanding connection requests queued at a TCP/IP port

Server platform	Maximum connection requests
AIX	100

Table 15. Maximum outstanding connection requests queued at a TCP/IP port (continued)

Server platform	Maximum connection requests
HP-UX	20
Linux	100
i5/OS	255
Solaris	128
Windows Server	200
Windows Workstation	5
z/OS	255

If the backlog reaches the values shown in Table 15 on page 152, the TCP/IP connection is rejected and the channel will not be able to start.

For MCA channels, this results in the channel going into a RETRY state and retrying the connection at a later time.

However, to avoid this error, you can add an entry in the `qm.ini` file:

```
TCP:  
ListenerBacklog = n
```

This overrides the default maximum number of outstanding requests (see Table 15 on page 152) for the TCP/IP listener.

Note: Some operating systems support a larger value than the default. If necessary, this can be used to avoid reaching the connection limit.

To run the listener with the backlog option switched on either:

- Use the `runmq1sr -b` command, or
- Use the MQSC command `DEFINE LISTENER` with the `BACKLOG` attribute set to the desired value.

For information about the `runmq1sr` command, see the *WebSphere MQ System Administration Guide* book. For information about the `DEFINE LISTENER` command, see the *WebSphere MQ Script (MQSC) Command Reference*.

Using the WebSphere MQ listener:

To run the listener supplied with WebSphere MQ, which starts new channels as threads, use the `runmq1sr` command. For example:

```
runmq1sr -t tcp [-m QMNAME] [-p 1822]
```

The square brackets indicate optional parameters; `QMNAME` is not required for the default queue manager, and the port number is not required if you are using the default (1414).

For the best performance, run the WebSphere MQ listener as a trusted application as described in “Running channels and listeners as trusted applications” on page 119. See the *WebSphere MQ Application Programming Guide* for information about trusted applications.

You can stop all WebSphere MQ listeners running on a queue manager that is inactive, using the command:

```
endmq1sr [-m QMNAME]
```

If you do not specify a queue manager name, the default queue manager is assumed.

Using the TCP/IP SO_KEEPALIVE option:

If you want to use the SO_KEEPALIVE option (as discussed in “Checking that the other end of the channel is still available” on page 61) you must add the following entry to your queue manager configuration file (qm.ini):

```
TCP:
  KeepAlive=yes
```

On some UNIX systems, you can define how long TCP waits before checking that the connection is still available, and how frequently it retries the connection if the first check fails. This is either a kernel tunable parameter, or can be entered at the command line. See the documentation for your UNIX system for more information.

Defining an LU 6.2 connection

SNA must be configured so that an LU 6.2 conversation can be established between the two machines.

See the *Multiplatform APPC Configuration Guide* and the following table for information.

Table 16. Settings on the local UNIX system for a remote queue manager platform

Remote platform	TPNAME	TPPATH
z/OS without CICS	The same as the corresponding TPName in the side information on the remote queue manager.	-
z/OS using CICS	CKRC (sender) CKSV (requester) CKRC (server)	-
i5/OS	The same as the compare value in the routing entry on the i5/OS system.	-
UNIX systems	The same as the corresponding TPName in the side information on the remote queue manager.	mqmtop/bin/amqcrs6a
Windows	As specified in the Windows Run Listener command, or the invocable Transaction Program that was defined using TpSetup on Windows.	mqmtop\bin\amqcrs6a

If you have more than one queue manager on the same machine, ensure that the TPnames in the channel definitions are unique.

Sending end

- On UNIX systems, create a CPI-C side object (symbolic destination) and enter this name in the Connection name field in the channel definition. Also create an LU 6.2 link to the partner.

In the CPI-C side object enter the partner LU name at the receiving machine, the transaction program name and the mode name. For example:

```
Partner LU Name          REMHOST
Remote TP Name           recv
Service Transaction Program no
Mode Name                 #INTER
```

On HP-UX, use the APPCLLU environment variable to name the local LU that the sender should use. On Solaris, set the APPC_LOCAL_LU environment variable to be the local LU name.

SECURITY PROGRAM is used, where supported by CPI-C, when WebSphere MQ attempts to establish an SNA session.

Receiving on LU 6.2

- On UNIX systems, create a listening attachment at the receiving end, an LU 6.2 logical connection profile, and a TPN profile.

In the TPN profile, enter the full path to the executable and the Transaction Program name:

```
Full path to TPN executable  mqmtop/bin/amqcrs6a
Transaction Program name     recv
User ID                       0
```

On systems where you can set the User ID, you should specify a user who is a member of the mqm group. On AIX, Solaris, and HP-UX, set the APPCTPN (transaction name) and APPCLLU (local LU name) environment variables (you can use the configuration panels for the invoked transaction program).

You may need to use a queue manager other than the default queue manager. If so, define a command file that calls:

```
amqcrs6a -m Queue_Man_Name
```

then call the command file.

Example configuration - IBM WebSphere MQ for AIX

This chapter gives an example of how to set up communication links from WebSphere MQ for AIX to WebSphere MQ products on the following platforms:

- Windows
- HP Tru64 UNIX
- HP-UX
- Solaris
- Linux
- i5/OS
- z/OS
- VSE/ESA

First it describes the parameters needed for an LU 6.2 connection, then it describes “Establishing a TCP connection” on page 166.

Once the connection is established, you need to define some channels to complete the configuration. This is described in “WebSphere MQ for AIX configuration” on page 167.

See “Example configuration chapters in this book” on page 101 for background information about this chapter and how to use it.

Configuration parameters for an LU 6.2 connection

Table 17 presents a worksheet listing all the parameters needed to set up communication from AIX to one of the other WebSphere MQ platforms. The worksheet shows examples of the parameters, which have been tested in a working environment, and leaves space for you to fill in your own values. An explanation of the parameter names follows the worksheet. Use the worksheet in this chapter in conjunction with the worksheet in the chapter for the platform to which you are connecting.

Configuration worksheet

Use the following worksheet to record the values you will use for this configuration. Where numbers appear in the Reference column they indicate that the value must match that in the appropriate worksheet elsewhere in this book. The examples that follow in this chapter refer back to the values in the ID column of this table. The entries in the Parameter Name column are explained in “Explanation of terms” on page 159.

Table 17. Configuration worksheet for Communications Server for AIX

ID	Parameter Name	Reference	Example	User Value
<i>Parameters for local node</i>				
1	Network name		NETID	
2	Control Point name		AIXPU	
3	Node ID		07123456	
4	Local LU name		AIXLU	
5	Local LU alias		AIXQMGR	
6	TP Name		MQSERIES	
7	Full path to TP executable		usr/lpp/mqm/bin/amqcrs6a	
8	Token-ring adapter address		123456789012	
9	Mode name		#INTER	
<i>Connection to a Windows system</i>				
The values in this section of the table must match those used in Table 13 on page 130, as indicated.				
10	Network name	2	NETID	
11	Remote LU name	5	WINNTLU	
12	Remote Transaction Program name	7	MQSERIES	
13	LU 6.2 CPI-C Side Information profile name		NTCPIC	
14	Mode name	17	#INTER	
15	LAN destination address	9	08005AA5FAB9	
16	Token-Ring Link Station profile name		NTPRO	
17	CP name of adjacent node	3	WINNTCP	
18	LU 6.2 partner LU profile name		NTLUPRO	
<i>Connection to an HP-UX system</i>				
The values in this section of the table must match those used in Table 19 on page 172, as indicated.				

Table 17. Configuration worksheet for Communications Server for AIX (continued)

ID	Parameter Name	Reference	Example	User Value
10	Network name	4	NETID	
11	Remote LU name	5	HPUXLU	
12	Remote Transaction Program name	7	MQSERIES	
13	LU 6.2 CPI-C Side Information profile name		HPUXCPIC	
14	Mode name	6	#INTER	
15	LAN destination address	8	100090DC2C7C	
16	Token-Ring Link Station profile name		HPUXPRO	
17	CP name of adjacent node	2	HPUXPU	
18	LU 6.2 partner LU profile name		HPUXLUPRO	
Connection to a Solaris system				
The values in this section of the table must match those used in Table 21 on page 195, as indicated.				
10	Network name	2	NETID	
11	Remote LU name	7	SOLARLU	
12	Remote Transaction Program name	8	MQSERIES	
17	LU 6.2 CPI-C Side Information profile name		SOLCPIC	
14	Mode name	17	#INTER	
5	LAN destination address	5	08002071CC8A	
16	Token-Ring Link Station profile name		SOLPRO	
17	CP name of adjacent node	3	SOLARPU	
18	LU 6.2 partner LU profile name		SOLLUPRO	
Connection to a Linux (x86 platform) system				
The values in this section of the table must match those used in Configuration worksheet for Communications Server for Linux, as indicated.				
10	Network name	4	NETID	
11	Remote LU name	5	LINUXLU	
12	Remote Transaction Program name	7	MQSERIES	
17	LU 6.2 CPI-C Side Information profile name		LXCPIC	
14	Mode name	6	#INTER	
5	LAN destination address	8	08005AC6DF33	
16	Token-Ring Link Station profile name		LXPRO	
17	CP name of adjacent node	2	LINUXPU	

Table 17. Configuration worksheet for Communications Server for AIX (continued)

ID	Parameter Name	Reference	Example	User Value
18	LU 6.2 partner LU profile name		LXLUPRO	
<i>Connection to an i5/OS system</i>				
The values in this section of the table must match those used in Table 35 on page 351, as indicated.				
10	Network name	1	NETID	
11	Remote LU name	3	AS400LU	
12	Remote Transaction Program name	8	MQSERIES	
13	LU 6.2 CPI-C Side Information profile name		AS4CPIC	
14	Mode name	17	#INTER	
15	LAN destination address	4	10005A5962EF	
16	Token-Ring Link Station profile name		AS4PRO	
17	CP name of adjacent node	2	AS400PU	
18	LU 6.2 partner LU profile name		AS4LUPRO	
<i>Connection to a z/OS system</i>				
The values in this section of the table must match those used in Table 27 on page 267, as indicated.				
10	Network name	2	NETID	
11	Remote LU name	4	MVSLU	
12	Remote Transaction Program name	7	MQSERIES	
13	LU 6.2 CPI-C Side Information profile name		MVSCPIC	
14	Mode name	10	#INTER	
15	LAN destination address	6	400074511092	
16	Token-Ring Link Station profile name		MVSPRO	
17	CP name of adjacent node	3	MVSPU	
18	LU 6.2 partner LU profile name		MVSLUPRO	
<i>Connection to a z/OS system using a generic interface</i>				
The values in this section of the table must match those used in Table 27 on page 267, as indicated.				
10	Network name	2	NETID	
11	Remote LU name	10	MVSGR	
12	Remote Transaction Program name	7	MQSERIES	
13	LU 6.2 CPI-C Side Information profile name		MVSCPIC	
14	Mode name	6	#INTER	
15	LAN destination address	8	400074511092	

Table 17. Configuration worksheet for Communications Server for AIX (continued)

ID	Parameter Name	Reference	Example	User Value
16	Token-Ring Link Station profile name		MVSPRO	
17	CP name of adjacent node	3	MVSPU	
18	LU 6.2 partner LU profile name		MVSLUPRO	
Connection to a VSE/ESA system				
The values in this section of the table must match those used in your VSE/ESA system.				
10	Network name	1	NETID	
11	Remote LU name	3	VSELU	
12	Remote Transaction Program name	4	MQ01	
13	LU 6.2 CPI-C Side Information profile name		VSECPIC	
14	Mode name		#INTER	
15	LAN destination address	5	400074511092	
16	Token-Ring Link Station profile name		VSEPRO	
17	CP name of adjacent node	2	VSEPU	
18	LU 6.2 partner LU profile name		VSELUPRO	

Explanation of terms

1 Network name

This is the unique ID of the network to which you are connected. Your network administrator will tell you this value.

2 Control Point name

This is a unique control point name for this workstation. Your network administrator will assign this to you.

3 XID node ID

This is a unique identifier for this workstation. On other platforms it is often referred to as the exchange ID (XID). Your network administrator will assign this to you.

4 Local LU name

A logical unit (LU) manages the exchange of data between systems. The local LU name is the name of the LU on your system. Your network administrator will assign this to you.

5 Local LU alias

The local LU alias is the name by which your local LU is known to your applications. You can choose this name yourself. It need be unique only on this machine.

6 TP Name

WebSphere MQ applications trying to converse with this workstation will specify a symbolic name for the program to be run at the receiving end. This will have been defined on the channel definition at the sender. It is recommended that when AIX is the receiver a Transaction Program Name

of MQSERIES is used, or in the case of a connection to VSE/ESA, where the length is limited to 4 bytes, use MQTP.

See Table 16 on page 154 for more information.

7 Full path to TP executable

This is the path and name of a shell script file that invokes the actual program to be run when a conversation is initiated with this workstation. You can choose the path and name of the script file. The contents of the file are illustrated in “WebSphere MQ for AIX TPN setup” on page 171.

8 Token-ring adapter address

This is the 12-character hex address of the token-ring card. It can be found by entering the AIX command:

```
lsfg -v -l tokn
```

where n is the number assigned to the token-ring adapter you are using. The **Network Address** field of the Token-Ring section indicates the adapter’s address.

9 Mode name

This is the name of a configuration profile used by Communications Server for AIX. The profile contains the set of parameters that control the APPC conversation. The mode name specified in the profile will be assigned to you by your network administrator. You supply the name to be used for the profile.

13 LU 6.2 CPI-C Side Information profile name

This is a name given to the Side Information profile defining a partner node. You supply the name. It needs to be unique only on this machine. You will later use the name in the WebSphere MQ sender channel definition.

16 Token-Ring Link Station profile name

This is the name of a configuration profile used by Communications Server for AIX. You supply the name to be used for the profile. The link station profile associates the link station with the SNA DLC profile, which has been used to define the hardware adapter and link characteristics, and the node control point.

17 CP name of adjacent node

This is the unique control point name of the partner system which which you are establishing communication. Your network administrator will assign this to you.

18 LU 6.2 partner LU profile name

This is the name of a configuration profile used by Communications Server for AIX. You supply the name to be used for the profile. It needs to be unique only on this machine. The profile defines parameters for establishing a session with a specific partner LU. In some scenarios, this profile may not be required but it is shown here to reduce the likelihood of error. See the *SNA Server for AIX Configuration Reference* manual for details.

Establishing a session using Communications Server for AIX

Verify the level of Communications Server software you have installed by entering the AIX command:

```
lslpp -h sna.rte
```

The level displayed in the response needs to be at least Version 5.0.

To update the SNA configuration profile, you need root authority. (Without root authority you can display options and appear to modify them, but cannot actually make any changes.) You can make configuration changes when SNA is either active or inactive.

The configuration scenario that follows was accomplished using the graphical interface.

Note: The setup used is APPN using independent LUs.

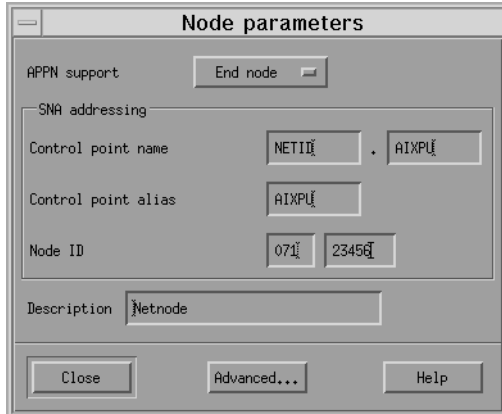
If you are an experienced user of AIX, you may choose to circumvent the panels and use the command-line interface. Refer to the *SNA Server for AIX Configuration Reference* manual to see the commands that correspond to the panels illustrated.

Throughout the following example, only the panels for profiles that must be added or updated are shown.

Configuring your node

This configuration uses a token ring setup. To define the end node to connect to the network node (assuming that a network node already exists), you need to:

1. Click on **Services** from the main menu on the main window.
2. Select **Configuration node parameters ...** from the drop-down list. A window entitled **Node parameters** appears:



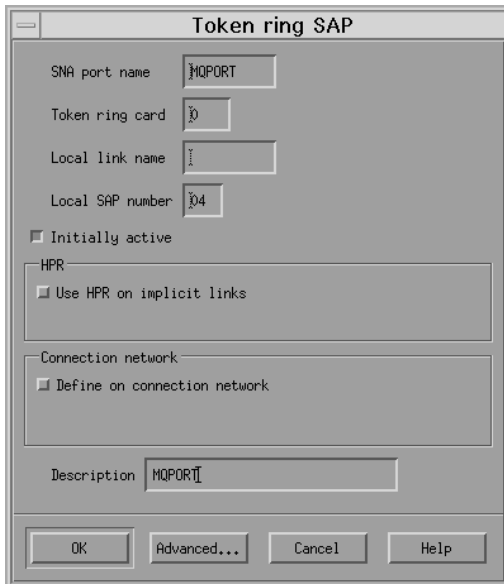
3. Click on **End node for APPN support**.
4. In the **SNA addressing** box, enter a name and alias for the Control point. The Control point name consists of a Network name (1) and a Control point name (2).
5. Enter the Node ID (3) of your local machine.
6. Click on **OK**.

You have now configured your node to connect to the network node.

Configuring connectivity to the network

1. Defining your port:
 - a. From the main menu of the main window, click on **Services, Connectivity, and New port ...** A window entitled **Add to machine name screen** appears.

- b. Select the default card for connecting to the network (**Token ring card**).
- c. Click on **OK**. A window entitled **Token ring SAP** appears:



- d. Enter a port name in the **SNA port name** box, for example, MQPORT.
 - e. Check **Initially Active**.
 - f. Click on **OK**.
2. Defining your connection to the network node:
- a. From the main menu on the main window, click on **Services, Connectivity, and New link station ...**
 - b. Click on **OK** to link your station to the chosen port (MQPORT). A window entitled **Token ring link station** appears:

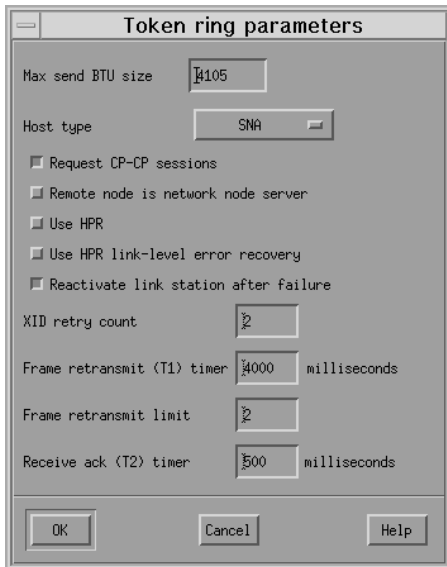
- c. Enter a name for your link station (4), for example, NETNODE.
- d. Enter the port name to which you want to connect the link station. In this case, the port name would be MQPORT.
- e. Check **Any** in the **LU traffic** box.
- f. Define where the remote node is by entering the control point on the network node in the **Independent LU traffic** box. The control point consists of a **Network name** (10) and a **CP name of adjacent node** (17).

Note: The network node does not have to be on the remote system that you are connecting to.

- g. Ensure the **Remote node type** is **Network node**.
- h. In the **Contact information**, enter the **MAC address** (15) of the token ring card on the network node.

Note: The network node does not have to be on the remote system that you are connecting to.

- i. Click on **Advanced ...**. A window entitled **Token ring parameters** appears:



- j. Check **Remote node is network node server**.
- k. Click on **OK**. The **Token ring link station** window remains on the screen.
- l. Click on **OK** on the **Token ring link station** window.

Defining a local LU

To define a local LU:

1. From the main menu on the main window, click on **Services, APPC, and New independent local LU ...** A window entitled **Local LU** appears:



Figure 33. Local LU window

2. Enter an LU name (4) and alias (5).
3. Click on **OK**.

You have now set up a basic SNA system.

To define the mode controlling the SNA session limits:

1. From the main menu in the main window, click on **Services, APPC, and Modes ...** A **Modes** window appears.
2. Select the **New ...** button. A window entitled **Mode** appears:

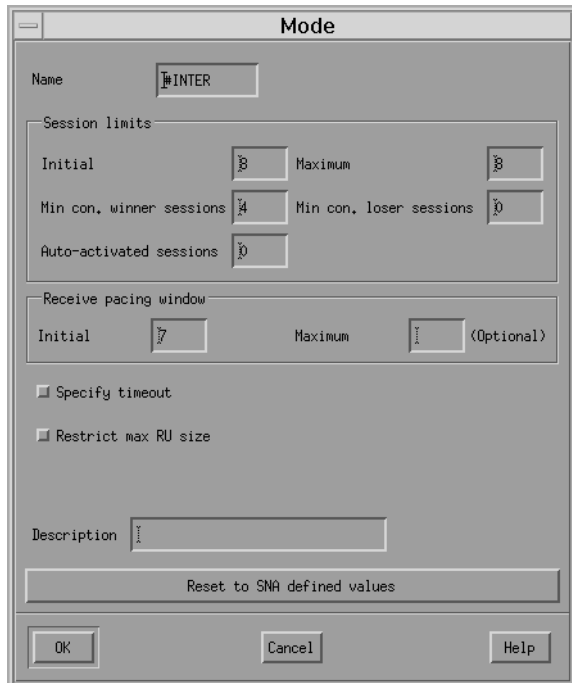


Figure 34. Mode window

3. Enter a **Name** (9) for your mode.
4. When you are happy with the session limits, click on **OK**. The **Modes** window remains on the screen.
5. Click on **Done** in the **Modes** window.

Defining a transaction program

WebSphere MQ allows you to use the Communications Server for AIX V5 graphical interface to configure transaction programs.

If you are migrating from a previous version of MQSeries, you should delete any existing Communications Server definitions of transaction programs that can be invoked by WebSphere MQ using the following commands:

1. Type


```
snaadmin delete_tp_load_info.tp_name=xxxxx
```
2. Then type


```
snaadmin delete_tp.tp_name=xxxxx
```

An attempt to invoke a previously defined transaction program results in a SNA sense code of 084B6031. In addition, error message AMQ9213 is returned. See WebSphere MQ Messages for more information about this and other WebSphere MQ messages.

You can then re-create the transaction program definition using the following instructions

From the main window, click **Services, APPC, and Transaction programs ...** The following panel is displayed:

1. Type **TP name** (6) in the **Application TP** field.
2. Clear the **Queue incoming Allocates** check box.
3. Type the **Full path to executable** (7).
4. Type **-m Local queue manager** in the **Arguments** field.
5. Type **mqm** in the **User ID** and **Group ID** fields.
6. Enter environment variables **APPCLU=local LU** (4) and **APPCTPN=Invokable TP** (6) separated by the pipe character in the **Environment** field.
7. Click **OK**.

Establishing a TCP connection

The WebSphere MQ command used to start the WebSphere MQ for TCP listener is:
`runmqtsr -t tcp`

The listener must be started explicitly before any channels are started. It enables receiving channels to start automatically in response to a request from an inbound sending channel.

Alternatively, if you want to use the UNIX supplied TCP/IP listener, complete the following steps:

1. Edit the file `/etc/services`.

Note: To edit the `/etc/services` file, you must be logged in as a superuser or root. If you do not have the following line in that file, add it as shown:

```
MQSeries      1414/tcp      # MQSeries channel listener
```

2. Edit the file `/etc/inetd.conf`. If you do not have the following line in that file, add it as shown:

```
MQSeries stream tcp nowait root /usr/mqm/bin/amqcrsta amqcrsta
[-m queue.manager.name]
```

3. Enter the command `refresh -s inetd`.

Note: You must add `root` to the `mqm` group. You need not have the primary group set to `mqm`. As long as `mqm` is in the set of groups, you can use the commands. If you are running only applications that use the queue manager you do not need `mqm` group authority.

What next?

The connection is now established. You are ready to complete the configuration. Go to “WebSphere MQ for AIX configuration.”

WebSphere MQ for AIX configuration

Note:

1. Before beginning the installation process ensure that you have first created the `mqm` user and group, and set the password.
2. If installation fails as a result of insufficient space in the file system you can increase the size as follows, using the command `smit C sna`. (Use `df` to display the current status of the file system. This will indicate the logical volume that is full.)
 - Physical and Logical Storage
 - File Systems
 - Add / Change / Show / Delete File Systems
 - Journaled File Systems
 - Change/Show Characteristics of a Journaled File System
3. Start any channel using the command:

```
runmqchl -c channel.name
```
4. Sample programs are installed in `/usr/mqm/samp`.
5. Error logs are stored in `/var/mqm/qmgrs/qmgrname/errors`.
6. On AIX you can start a trace of the WebSphere MQ components by using standard WebSphere MQ trace commands, or using AIX system trace. See *WebSphere MQ System Administration Guide* for more information on WebSphere MQ Trace and AIX system trace.
7. When you are using the command interpreter `runmqsc` to enter administration commands, a `+` at the end of a line indicates that the next line is a continuation. Ensure that there is a space between the last parameter and the continuation character.

Basic configuration

1. Create the queue manager from the AIX command line using the command:

```
crtmqm -u dlqname -q aix
```

where:

`aix` Is the name of the queue manager

`-q` Indicates that this is to become the default queue manager

`-u dlqname`

Specifies the name of the undeliverable message queue

This command creates a queue manager and a set of default objects.

2. Start the queue manager from the AIX command line using the command:

strmqm *aix*

where *aix* is the name given to the queue manager when it was created.

3. Start **runmqsc** from the AIX command line and use it to create the undeliverable message queue by entering the command:

```
def ql (dlqname)
```

where *dlqname* is the name given to the undeliverable message queue when the queue manager was created.

Channel configuration

The following section details the configuration to be performed on the AIX queue manager to implement the channel described in Figure 32 on page 101.

In each case the MQSC command is shown. Either start **runmqsc** from an AIX command line and enter each command in turn, or build the commands into a command file.

Examples are given for connecting WebSphere MQ for AIX and WebSphere MQ for Windows. If you wish to connect to WebSphere MQ on another platform use the appropriate set of values from the table in place of those for Windows.

Note: The words in **bold** are user-specified and reflect the names of WebSphere MQ objects used throughout these examples. If you change the names used here, ensure that you also change the other references made to these objects throughout this book. All others are keywords and should be entered as shown.

Table 18. Configuration worksheet for WebSphere MQ for AIX

ID	Parameter Name	Reference	Example Used	User Value
<i>Definition for local node</i>				
A	Queue Manager Name		AIX	
B	Local queue name		AIX.LOCALQ	
<i>Connection to WebSphere MQ for Windows</i>				
The values in this section of the table must match those used in Table 14 on page 146, as indicated.				
C	Remote queue manager name	A	WINNT	
D	Remote queue name		WINNT.REMOTEQ	
E	Queue name at remote system	B	WINNT.LOCALQ	
F	Transmission queue name		WINNT	
G	Sender (SNA) channel name		AIX.WINNT.SNA	
H	Sender (TCP/IP) channel name		AIX.WINNT.TCP	
I	Receiver (SNA) channel name	G	WINNT.AIX.SNA	
J	Receiver (TCP) channel name	H	WINNT.AIX.TCP	
<i>Connection to MQSeries for HP Tru64 UNIX</i>				
The values in this section of the table must match those used in your HP Tru64 UNIX system.				
C	Remote queue manager name	A	DECUX	
D	Remote queue name		DECUX.REMOTEQ	
E	Queue name at remote system	B	DECUX.LOCALQ	
F	Transmission queue name		DECUX	

Table 18. Configuration worksheet for WebSphere MQ for AIX (continued)

ID	Parameter Name	Reference	Example Used	User Value
H	Sender (TCP) channel name		DECUX.AIX.TCP	
J	Receiver (TCP) channel name	H	AIX.DECUX.TCP	
Connection to WebSphere MQ for HP-UX				
The values in this section of the table must match those used in Table 20 on page 191, as indicated.				
C	Remote queue manager name	A	HPUX	
D	Remote queue name		HPUX.REMOTEQ	
E	Queue name at remote system	B	HPUX.LOCALQ	
F	Transmission queue name		HPUX	
G	Sender (SNA) channel name		AIX.HPUX.SNA	
H	Sender (TCP) channel name		AIX.HPUX.TCP	
I	Receiver (SNA) channel name	G	HPUX.AIX.SNA	
J	Receiver (TCP) channel name	H	HPUX.AIX.TCP	
Connection to WebSphere MQ for Solaris				
The values in this section of the table must match those used in Table 22 on page 212, as indicated.				
C	Remote queue manager name	A	SOLARIS	
D	Remote queue name		SOLARIS.REMOTEQ	
E	Queue name at remote system	B	SOLARIS.LOCALQ	
F	Transmission queue name		SOLARIS	
G	Sender (SNA) channel name		AIX.SOLARIS.SNA	
H	Sender (TCP/IP) channel name		AIX.SOLARIS.TCP	
I	Receiver (SNA) channel name	G	SOLARIS.AIX.SNA	
J	Receiver (TCP/IP) channel name	H	SOLARIS.AIX.TCP	
Connection to WebSphere MQ for Linux				
The values in this section of the table must match those used in Table 24 on page 235, as indicated.				
C	Remote queue manager name	A	LINUX	
D	Remote queue name		LINUX.REMOTEQ	
E	Queue name at remote system	B	LINUX.LOCALQ	
F	Transmission queue name		LINUX	
G	Sender (SNA) channel name		AIX.LINUX.SNA	
H	Sender (TCP/IP) channel name		AIX.LINUX.TCP	
I	Receiver (SNA) channel name	G	LINUX.AIX.SNA	
J	Receiver (TCP/IP) channel name	H	LINUX.AIX.TCP	
Connection to WebSphere MQ for i5/OS				
The values in this section of the table must match those used in Table 36 on page 364, as indicated.				
C	Remote queue manager name	A	AS400	
D	Remote queue name		AS400.REMOTEQ	
E	Queue name at remote system	B	AS400.LOCALQ	
F	Transmission queue name		AS400	
G	Sender (SNA) channel name		AIX.AS400.SNA	

Table 18. Configuration worksheet for WebSphere MQ for AIX (continued)

ID	Parameter Name	Reference	Example Used	User Value
H	Sender (TCP) channel name		AIX.AS400.TCP	
I	Receiver (SNA) channel name	G	AS400.AIX.SNA	
J	Receiver (TCP) channel name	H	AS400.AIX.TCP	
Connection to WebSphere MQ for z/OS				
The values in this section of the table must match those used in Table 28 on page 272, as indicated.				
C	Remote queue manager name	A	MVS	
D	Remote queue name		MVS.REMOTEQ	
E	Queue name at remote system	B	MVS.LOCALQ	
F	Transmission queue name		MVS	
G	Sender (SNA) channel name		AIX.MVS.SNA	
H	Sender (TCP) channel name		AIX.MVS.TCP	
I	Receiver (SNA) channel name	G	MVS.AIX.SNA	
J	Receiver (TCP) channel name	H	MVS.AIX.TCP	
Connection to WebSphere MQ for z/OS using queue-sharing groups				
The values in this section of the table must match those used in Table 30 on page 292, as indicated.				
C	Remote queue manager name	A	QSG	
D	Remote queue name		QSG.REMOTEQ	
E	Queue name at remote system	B	QSG.SHAREDQ	
F	Transmission queue name		QSG	
G	Sender (SNA) channel name		AIX.QSG.SNA	
H	Sender (TCP) channel name		AIX.QSG.TCP	
I	Receiver (SNA) channel name	G	QSG.AIX.SNA	
J	Receiver (TCP) channel name	H	QSG.AIX.TCP	
Connection to MQSeries for VSE/ESA				
The values in this section of the table must match those used in your VSE/ESA system.				
C	Remote queue manager name	A	VSE	
D	Remote queue name		VSE.REMOTEQ	
E	Queue name at remote system	B	VSE.LOCALQ	
F	Transmission queue name		VSE	
G	Sender channel name		AIX.VSE.SNA	
I	Receiver channel name	G	VSE.AIX.SNA	

WebSphere MQ for AIX sender-channel definitions using SNA:

```
def ql (WINNT) +                               F
  usage(xmitq) +
  replace

def qr (WINNT.REMOTEQ) +                       D
  rname(WINNT.LOCALQ) +                       E
  rqmname(WINNT) +                             C
  xmitq(WINNT) +                               F
  replace
```

```
def chl (AIX.WINNT.SNA) chltype(sdr) +          G
    trptype(lu62) +
    conname('WINNTCPIC') +                    17
    xmitq(WINNT) +                             F
    replace
```

WebSphere MQ for AIX receiver-channel definitions using SNA:

```
def ql (AIX.LOCALQ) replace                    B

def chl (WINNT.AIX.SNA) chltype(rcvr) +       I
    trptype(lu62) +
    replace
```

WebSphere MQ for AIX TPN setup:

During the AIX Communications Server configuration process, an LU 6.2 TPN profile was created, which contained the full path to a TP executable. In the example the file was called `u/interop/AIX.crs6a`. You can choose a name, but you are recommended to include the name of your queue manager in it. The contents of the executable file must be:

```
#!/bin/sh
/opt/mqm/bin/amqcrs6a -m aix
```

where *aix* is the queue manager name (A). After creating this file, enable it for execution by running the command:

```
chmod 755 /u/interop/AIX.crs6a
```

As an alternative to creating an executable file, you can specify the path on the Add LU 6.2 TPN Profile panel, using command line parameters.

Specifying a path in one of these two ways ensures that SNA receiver channels activate correctly when a sender channel initiates a conversation.

WebSphere MQ for AIX sender-channel definitions using TCP:

```
def ql (WINNT) +                              F
    usage(xmitq) +
    replace

def qr (WINNT.REMOTEQ) +                     D
    rname(WINNT.LOCALQ) +                   E
    rqmname(WINNT) +                       C
    xmitq(WINNT) +                         F
    replace

def chl (AIX.WINNT.TCP) chltype(sdr) +       H
    trptype(tcp) +
    conname(remote_tcpip_hostname) +
    xmitq(WINNT) +                         F
    replace
```

WebSphere MQ for AIX receiver-channel definitions using TCP:

```
def ql (AIX.LOCALQ) replace                    B

def chl (WINNT.AIX.TCP) chltype(rcvr) +       J
    trptype(tcp) +
    replace
```

Example configuration - IBM WebSphere MQ for HP-UX

This chapter gives an example of how to set up communication links from WebSphere MQ for HP-UX to WebSphere MQ products on the following platforms:

- Windows
- AIX
- HP Tru64 UNIX
- Solaris
- Linux
- i5/OS
- z/OS
- VSE/ESA

First it describes the parameters needed for an LU 6.2 connection, then it describes “Establishing a session using HP SNAplus2” on page 176 and “Establishing a TCP connection” on page 189.

Once the connection is established, you need to define some channels to complete the configuration. This is described in “WebSphere MQ for HP-UX configuration” on page 190.

See “Example configuration chapters in this book” on page 101 for background information about this chapter and how to use it.

Configuration parameters for an LU 6.2 connection

Table 19 presents a worksheet listing all the parameters needed to set up communication from HP-UX to one of the other WebSphere MQ platforms. The worksheet shows examples of the parameters, which have been tested in a working environment, and leaves space for you to fill in your own values. An explanation of the parameter names follows the worksheet. Use the worksheet in this chapter in conjunction with the worksheet in the chapter for the platform to which you are connecting.

Configuration worksheet

Use this worksheet to record the values you use for your configuration. Where numbers appear in the Reference column they indicate that the value must match that in the appropriate worksheet elsewhere in this book. The examples that follow in this chapter refer back to the values in the ID column. The entries in the Parameter Name column are explained in “Explanation of terms” on page 175.

Table 19. Configuration worksheet for HP SNAplus2

ID	Parameter Name	Reference	Example	User Value
<i>Parameters for local node</i>				
1	Configuration file name		sna_node.cfg	
2	Control point name		HPUXPU	
3	Node ID to send		05D 54321	
4	Network name		NETID	
5	Local APPC LU		HPUXLU	
6	APPC mode		#INTER	

Table 19. Configuration worksheet for HP SNAplus2 (continued)

ID	Parameter Name	Reference	Example	User Value
7	Invokable TP		MQSERIES	
8	Token-Ring adapter address		100090DC2C7C	
9	Port name		MQPORT	
10	Full path to executable		/opt/mqm/bin/amqcrs6a	
11	Local queue manager		HPUX	
Connection to a Windows system				
The values in this section of the table must match those used in Table 13 on page 130, as indicated.				
12	Link station name		NTCONN	
13	Network name	2	NETID	
14	CP name	3	WINNTCP	
15	Remote LU	5	WINNTLU	
16	Application TP	7	MQSERIES	
17	Mode name	17	#INTER	
18	CPI-C symbolic destination name		NTCPIC	
19	Remote network address	9	08005AA5FAB9	
20	Node ID to receive	4	05D 30F65	
Connection to an AIX system				
The values in this section of the table must match those used in Table 17 on page 156, as indicated.				
12	Link station name		AIXCONN	
13	Network name	1	NETID	
14	CP name	2	AIXPU	
15	Remote LU	4	AIXLU	
16	Application TP	6	MQSERIES	
17	Mode name	14	#INTER	
18	CPI-C symbolic destination name		AIXCPIC	
19	Remote network address	8	123456789012	
20	Node ID to receive	3	071 23456	
Connection to a Solaris system				
The values in this section of the table must match those used in Table 21 on page 195, as indicated.				
12	Link station name		SOLCONN	
13	Network name	2	NETID	
14	CP name	3	SOLARPU	
15	Remote LU	7	SOLARLU	
16	Application TP	8	MQSERIES	
17	Mode name	17	#INTER	
18	CPI-C symbolic destination name		SOLCPIC	
19	Remote network address	5	08002071CC8A	
20	node ID to receive	6	05D 310D6	

Table 19. Configuration worksheet for HP SNAplus2 (continued)

ID	Parameter Name	Reference	Example	User Value
Connection to a Linux (x86 platform) system				
The values in this section of the table must match those used in Configuration worksheet for Communications Server for Linux, as indicated.				
12	Link station name		LXCONN	
13	Network name	4	NETID	
14	CP name	2	LINUXPU	
15	Remote LU	5	LINUXLU	
16	Application TP	7	MQSERIES	
17	Mode name	6	#INTER	
18	CPI-C symbolic destination name		LXCPIC	
19	Remote network address	8	08005AC6DF33	
20	node ID to receive	3	05D 30A55	
Connection to an i5/OS system				
The values in this section of the table must match those used in Table 35 on page 351, as indicated.				
12	Link station name		AS4CONN	
13	Network name	1	NETID	
14	CP name	2	AS400PU	
15	Remote LU	3	AS400LU	
16	Application TP	8	MQSERIES	
17	Mode name	17	#INTER	
18	CPI-C symbolic destination name		AS4CPIC	
19	Remote network address	4	10005A5962EF	
Connection to a z/OS system				
The values in this section of the table must match those used in Table 27 on page 267, as indicated.				
12	Link station name		MVSCONN	
13	Network name	2	NETID	
14	CP name	3	MVSPU	
15	Remote LU	4	MVSLU	
16	Application TP	7	MQSERIES	
17	Mode name	10	#INTER	
18	CPI-C symbolic destination name		MVSCPIC	
19	Remote network address	8	400074511092	
Connection to a VSE/ESA system				
The values in this section of the table must match those used in your VSE/ESA system.				
12	Link station name		VSECONN	
13	Network name	1	NETID	
14	CP name	2	VSEPU	
15	Remote LU	3	VSELU	
16	Application TP	4	MQ01	MQ01

Table 19. Configuration worksheet for HP SNAplus2 (continued)

ID	Parameter Name	Reference	Example	User Value
17	Mode name		#INTER	
18	CPI-C symbolic destination name		VSECPIC	
19	Remote network address	5	400074511092	

Explanation of terms

1 Configuration file name

This is the unique name of the SNAplus2 configuration file. The default for this name is `sna_node.cfg`.

Although it is possible to edit this file it is strongly recommended that configuration is done using xsnapadmin.

2 Control point name

This is the unique Control point name for this workstation. In the SNA network, the Control point is an addressable location (PU type 2.1). Your network administrator will assign this to you.

3 Node ID to send

This is the unique ID of this workstation. On other platforms this is often referred to as the *Exchange ID* or *XID*. Your network administrator will assign this ID for you.

4 Network name

This is the unique ID of the network to which you are connected. It is an alphanumeric value and can be 1-8 characters long. The network name works with the Control point name to uniquely identify a system. Your network administrator will tell you the value.

5 Local APPC LU

An LU manages the exchange of data between transactions. The local APPC LU name is the name of the LU on your system. Your network administrator will assign this to you.

6 APPC mode

This is the name given to the set of parameters that control the APPC conversation. This name must be defined at each partner system. Your network administrator will assign this to you.

7 Invokable TP

WebSphere MQ applications trying to converse with this workstation will specify a symbolic name for the program to be run at the receiving end. This will have been defined on the channel definition at the sender. For simplicity, wherever possible use a transaction program name of MQSERIES, or in the case of a connection to VSE/ESA, where the length is limited to 4 bytes, use MQTP.

See Table 16 on page 154 for more information.

8 Token-ring adapter address

Use the HP-UX System Administration Manager (SAM) to discover the adapter address for this workstation. You need root authority to use SAM. From the initial menu, select **Networking and Communications**, then select **Network Interface cards** followed by **LAN 0** (or whichever LAN you are using). The adapter address is displayed under the heading Station

Address (hex). The card name represents the appropriate card type. If you do not have root level authority, your HP-UX system administrator can tell you the value.

9 Port name

This is a meaningful symbolic name that is used to associate the definitions with a network interface (in this case, a Token-Ring adapter). A separate Port must be defined for each physical device attached to the workstation.

10 Full path to executable

On HP SNAplus2 Release 5, this is the path and name of a shell script file that invokes the actual program to be run when a conversation is initiated with this workstation. You can choose the path and name of the script file. The contents of the file are illustrated in “WebSphere MQ for HP-UX invocable TP setup” on page 193. On HP SNAplus2 Release 6, this is the path and name of the program to be run when a conversation is initiated with this workstation. You enter the path in the **TP invocation** screen (see “Adding a TP definition using HP SNAplus2 Release 6” on page 187).

11 Local queue manager

This is the name of the queue manager on your local system.

10 Link station name

This is a meaningful symbolic name by which the connection to a peer or host node is known. It defines a logical path to the remote system. Its name is used only inside SNAplus2 and is specified by you. The connection must be associated with an existing Link and owned by one local node. You must define one connection for each partner or host system.

18 CPI-C symbolic destination name

This is a name given to the definition of a partner node. You choose the name. It need be unique only on this machine. Later you can use this name in the WebSphere MQ sender channel definition.

20 Node ID to receive

This is the unique ID of the partner workstation with which you will be communicating. On other platforms this is often referred to as the *Exchange ID* or *XID*. For a connection to a host system any values except 000 FFFFF and FFF FFFFF may be specified. Your network administrator will assign this ID for you.

Establishing a session using HP SNAplus2

The following information guides you through the tasks you must perform to create the SNA infrastructure that WebSphere MQ requires. This example creates the definitions for a partner node and LU on OS/2®.

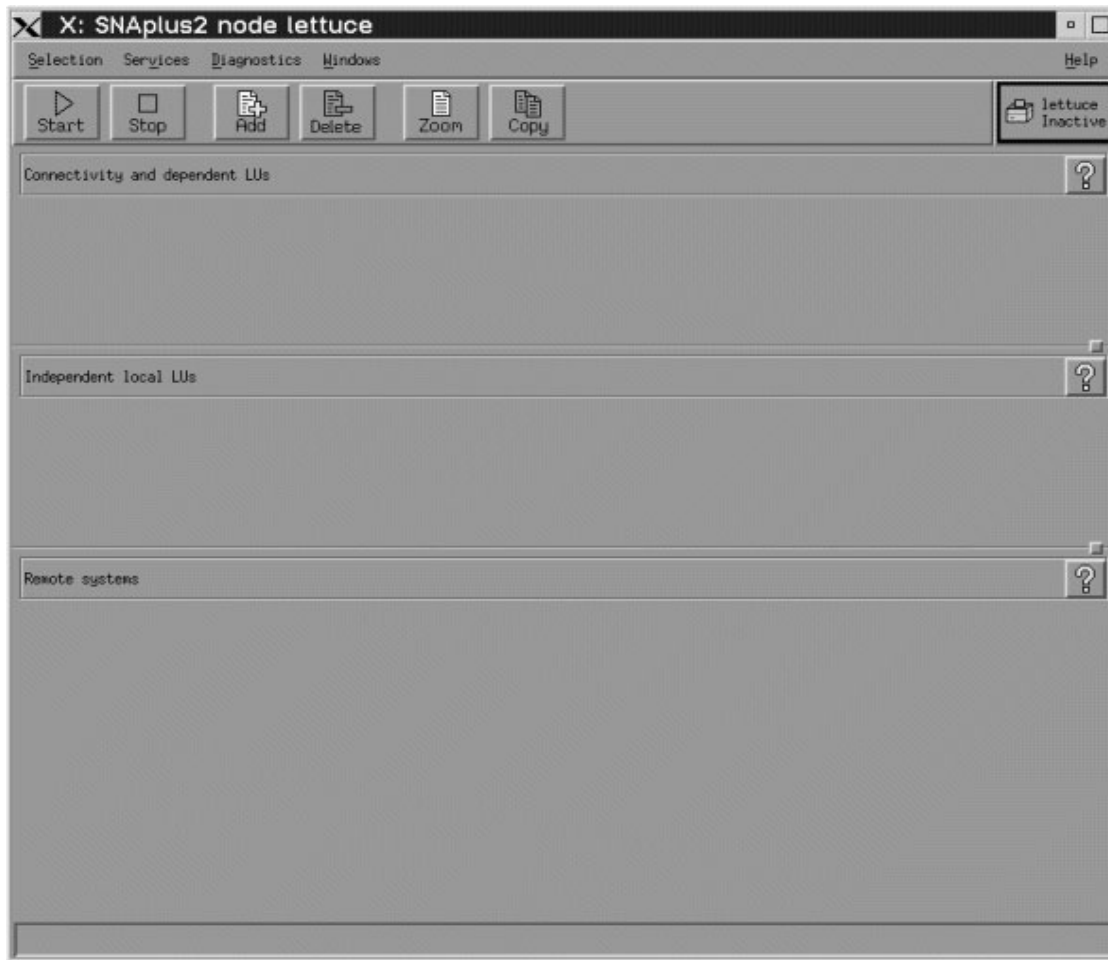
Use **snap start** followed by **xsnapadmin** to enter the HP SNAplus2 configuration panels. You need root authority to use **xsnapadmin**.

SNAplus2 configuration

SNAplus2 configuration involves the following steps:

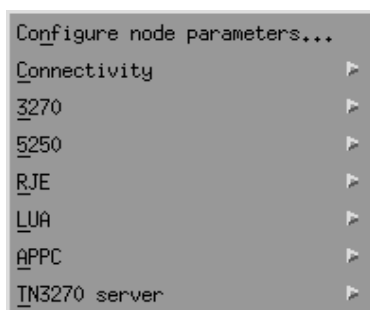
1. Defining a local node
2. Adding a Token Ring Port
3. Defining a local LU

The SNAplus2 main menu, from which you will start, is shown below:

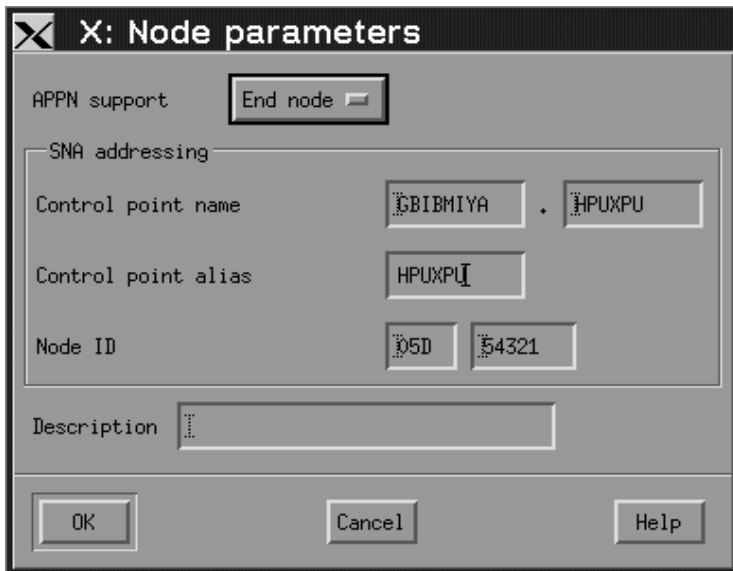


Defining a local node:

1. From the SNAplus2 main menu, select the **Services** pull-down:



2. Select **Configure node parameters....** The following panel is displayed:



3. Complete the **Control point name** with the values **Network name** (4) and **Control point name** (2).
4. Enter the **Control point name** (2) in the **Control point alias** field.
5. Enter the **Node ID** (3).
6. Select **End node**.
7. Press **OK**.

A default independent local LU is defined.

Adding a Token Ring Port:

1. From the main SNAPplus2 menu, select the Connectivity and dependent LUs panel.
2. Press **Add**. The following panel is displayed:



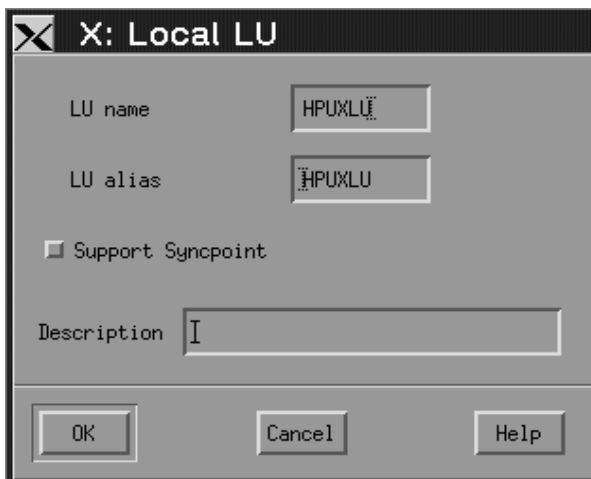
3. Select a Token Ring Card port and press **OK**. The following panel is displayed:



4. Enter the **SNA port name** (9).
5. Enter a **Description** and press **OK** to take the default values.

Defining a local LU:

1. From the main SNAPplus2 menu, select the Independent local LUs panel.
2. Press **Add**. The following panel is displayed:



3. Enter the **LU name** (5) and press **OK**.

APPC configuration

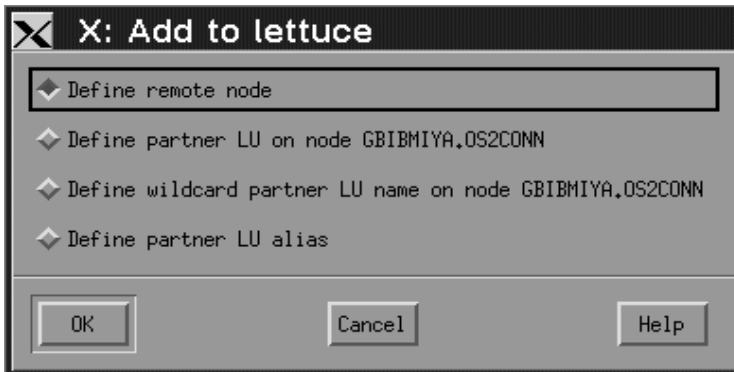
APPC configuration involves the following steps:

1. Defining a remote node
2. Defining a partner LU
3. Defining a link station

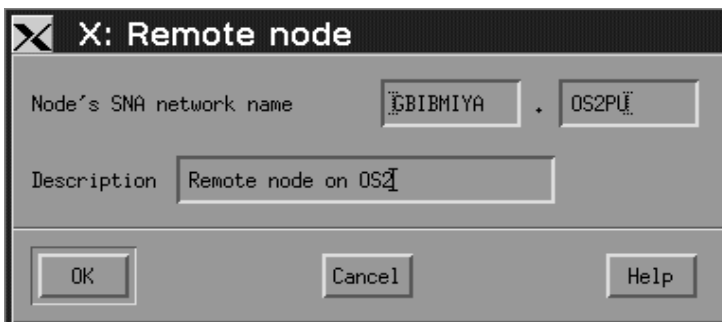
4. Defining a mode
5. Adding CPI-C information
6. Adding a TP definition

Defining a remote node:

1. From the main SNAplus2 menu, select the Remote systems panel.
2. Press **Add**. The following panel is displayed:



3. Select **Define remote node** and press **OK**. The following panel is displayed:



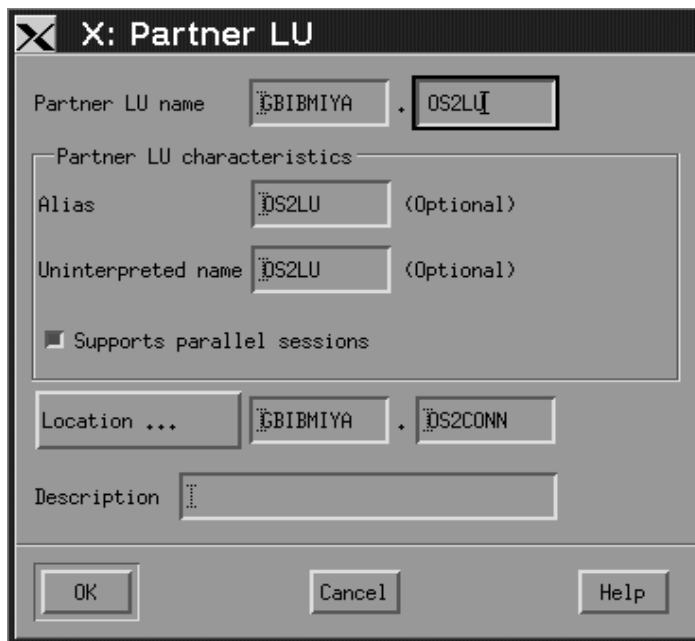
4. Enter the **Node's SNA network name** (13) and a **Description**.
5. Press **OK**.
6. A default partner LU with the same name is generated and a message is displayed.
7. Press **OK**.

Defining a partner LU:

1. From the main SNAplus2 menu, select the remote node in the Remote systems panel.
2. Press **Add**. The following panel is displayed:



3. Select **Define partner LU on node node name**.
4. Press **OK**. The following panel is displayed:



5. Enter the **partner LU name** (15) and press **OK**.

Defining a link station:

1. From the main SNAplus2 menu, select the Connectivity and dependent LUs panel.
2. Select the MQPORT port.
3. Press **Add**. The following panel is displayed:



4. Select **Add link station to port MQPORT**.
5. Press **OK**. The following panel is displayed:



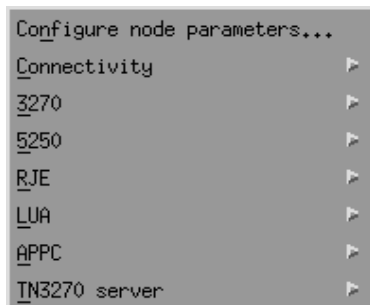
6. Enter the **Name** of the link station (12).
7. Set the value of **Activation** to “On demand”.
8. Select **Independent only**.
9. Press **Remote node...** and select the value of the remote node (14).
10. Press **OK**.
11. Set the value of **Remote node type** to “End or LEN node”.
12. Enter the value for **MAC address** (19) and press **Advanced...** The following panel is displayed:



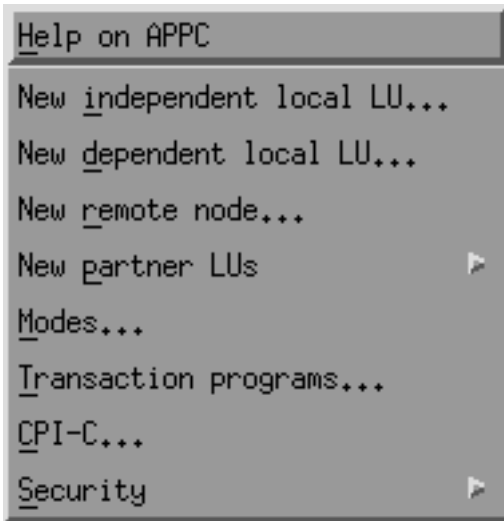
13. Select **Reactivate link station after failure**.
14. Press **OK** to exit the Advanced... panel.
15. Press **OK** again.

Defining a mode:

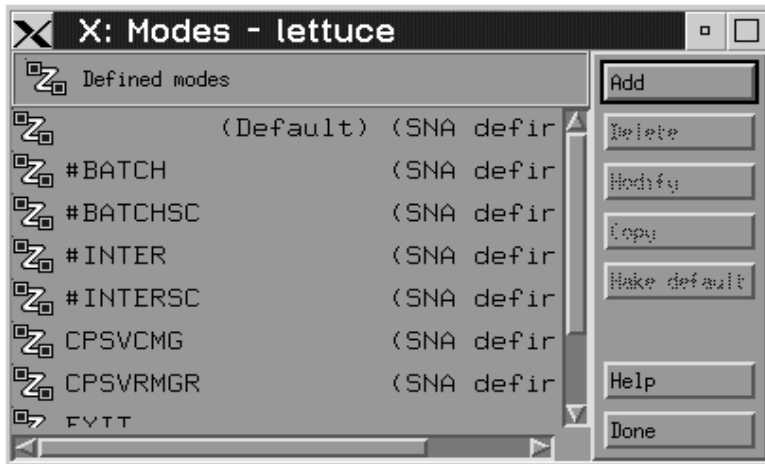
1. From the SNAplus2 main menu, select the **Services** pull-down: The following panel is displayed:



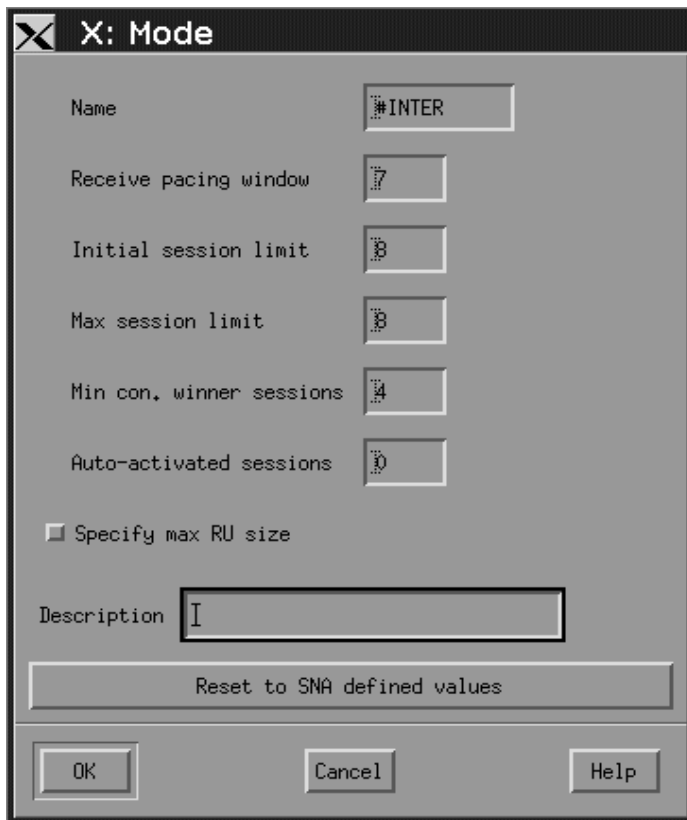
2. Select **APPC**. The following panel is displayed:



3. Select **Modes...**. The following panel is displayed:



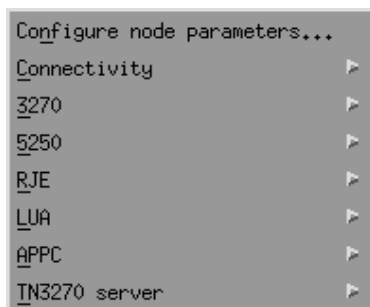
4. Press **Add**. The following panel is displayed:



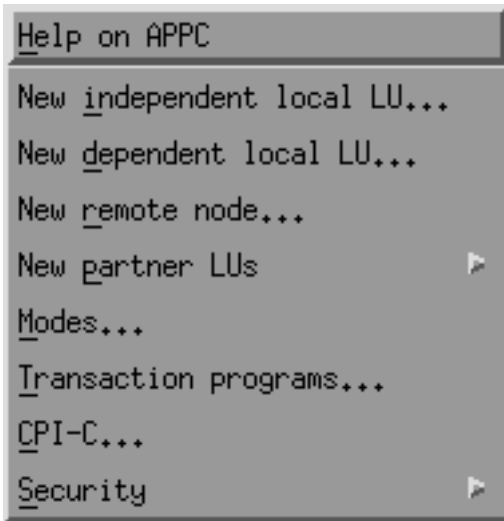
5. Enter the **Name** to be given to the mode (17).
6. Set the values of **Initial session limit** to 8, **Min con. winner sessions** to 4, and **Auto-activated sessions** to 0.
7. Press **OK**.
8. Press **Done**.

Adding CPI-C information:

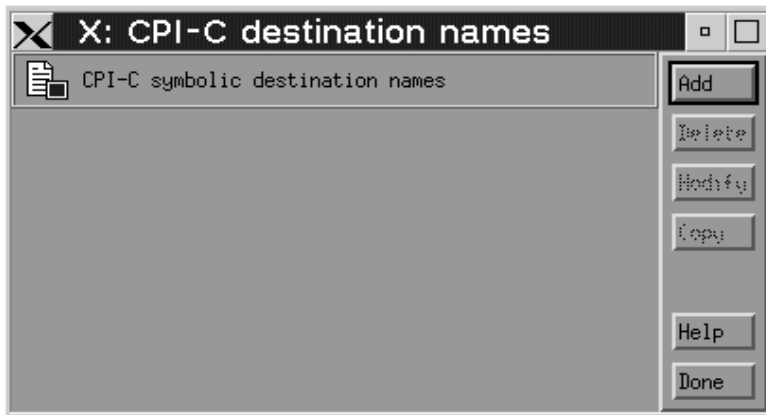
1. From the SNAplus2 main menu, select the **Services** pull-down:



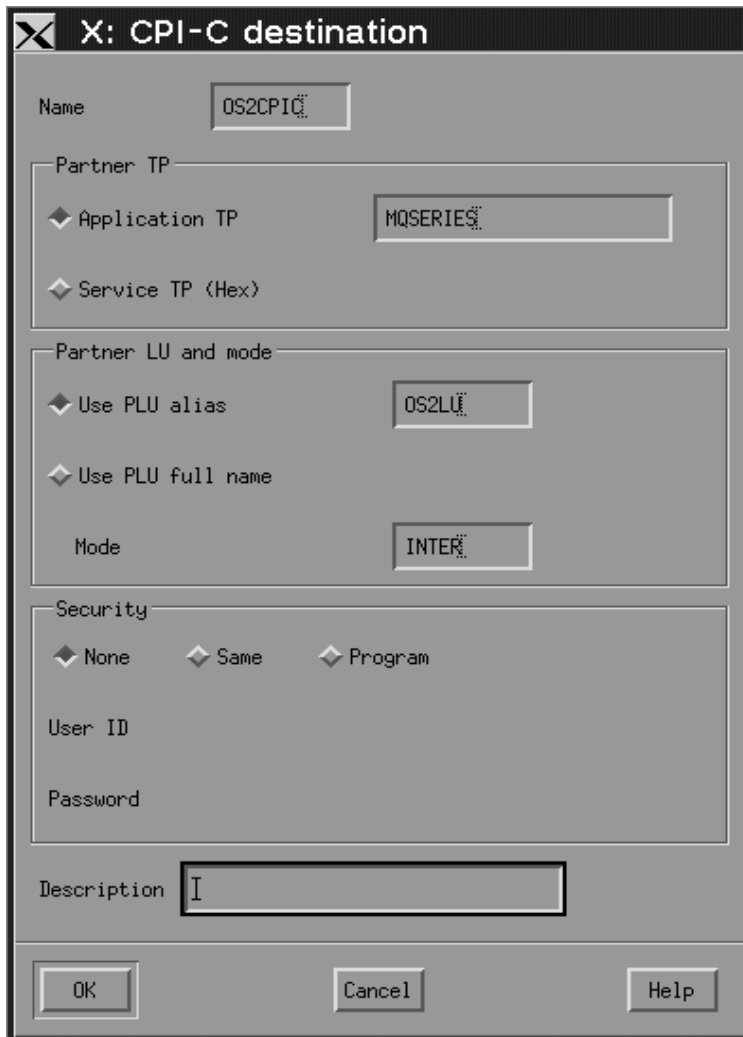
2. Select **APPC**. The following panel is displayed:



3. Select **CPI-C...**. The following panel is displayed:



4. Press **Add**. The following panel is displayed:



5. Enter the **Name** (18). Select **Application TP** and enter the value (16). Select **Use PLU alias** and enter the name (15). Enter the **Mode** name (17).
6. Press **OK**.

Adding a TP definition using HP SNAplus2 Release 5:

Invokable TP definitions are kept in the file `/etc/opt/sna/sna_tps`. This should be edited to add a TP definition. Add the following lines:

```
[MQSERIES]
PATH = /users/interops/HPUX.crs6a
TYPE = NON-QUEUED
USERID = mqm
ENV = APPCLU=HPUXLU
ENV = APPCTPN=MQSERIES
```

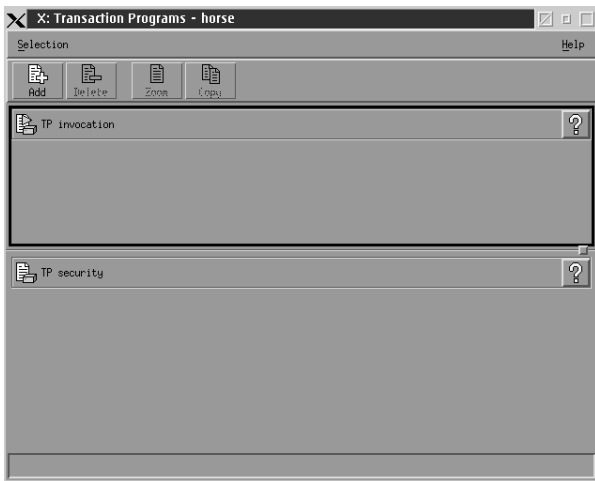
See “WebSphere MQ for HP-UX invokable TP setup” on page 193 for more information about TP definitions.

Adding a TP definition using HP SNAplus2 Release 6:

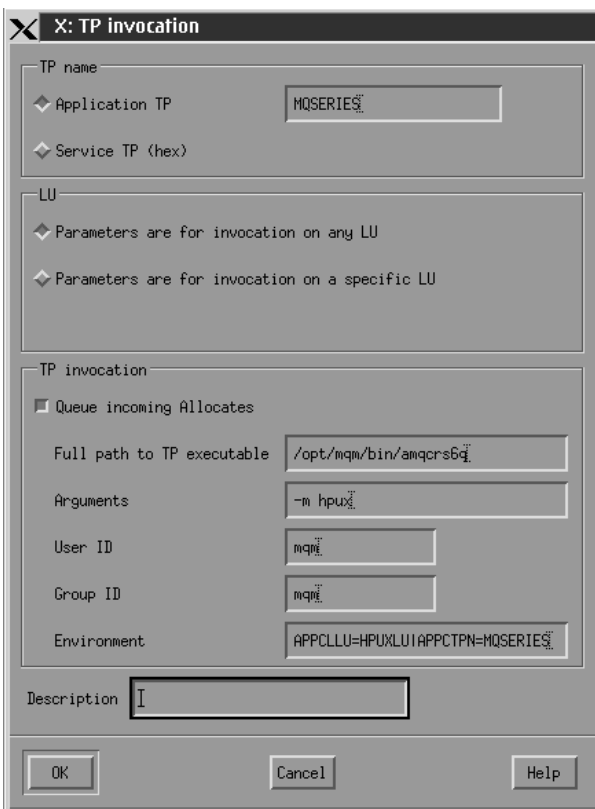
To add a TP definition:

1. Select **Services** pulldown and select **APPC** as for CPI-C information.

2. Select **Transaction Programs**. The following panel is displayed:



3. Select **Add**. The following panel is displayed:



4. Enter **TP name** (7) in the **Application TP** field.
5. Mark **Incoming Allocates** as non-queued.
6. Enter **Full path to executable** (10).
7. Enter **-m Local queue manager** (11) in the **Arguments** field.
8. Enter **mqm** in the **User ID** and **Group ID** fields.

9. Enter environment variables **APPCLLU=local LU (5)** and **APPCTPN=Invokable TP (7)** separated by the pipe character in the **Environment** field.
10. Press **OK** to save your definition.

HP-UX operation

The SNAplus2 control daemon is started with the **snap start** command. Depending on the options selected at installation, it may already be running.

The **xsnapadmin** utility controls SNAplus2 resources.

Logging and tracing are controlled from here. Log and trace files can be found in the `/var/opt/sna` directory. The logging files `sna.aud` and `sna.err` can be read using a standard editor such as `vi`.

In order to read the trace files **sna1.trc** and **sna2.trc** they must first be formatted by running the command **snaptrcfmt -f sna1.trc -o sna1** which produces a `sna1.dmp` file which can be read using a normal editor.

The configuration file itself is editable but this is not a recommended method of configuring SNAplus2.

The APPCLU environment variables must be set before starting a sender channel from the HP-UX system. The command can be either entered interactively or added to the logon profile. Depending on the level of BOURNE shell or KORN shell program being used, the command will be:

```
export APPCLLU=HPUXLU    5    newer level
```

or

```
APPCLLU=HPUXLU          5    older level
export
```

What next?

The connection is now established. You are ready to complete the configuration. Go to “WebSphere MQ for HP-UX configuration” on page 190.

Establishing a TCP connection

The listener must be started explicitly before any channels are started. It enables receiving channels to start automatically in response to a request from an inbound sending channel.

Alternatively, if you want to use the UNIX supplied TCP/IP listener, complete the following steps:

1. Edit the file `/etc/services`.

Note: To edit the `/etc/services` file, you must be logged in as a superuser or root. If you do not have the following line in that file, add it as shown:

```
MQSeries      1414/tcp      # MQSeries channel listener
```

2. Edit the file `/etc/inetd.conf`. If you do not have the following line in that file, add it as shown:

```
MQSeries stream tcp nowait root /opt/mqm/bin/amqcrsta amqcrsta
[-m queue.manager.name]
```

3. Find the process ID of the inetd with the command:

```
ps -ef | grep inetd
```

4. Run the command:

```
kill -1 inetd processid
```

Note: You must add **root** to the mqm group. You do not need not have the primary group set to mqm. As long as mqm is in the set of groups, you can use the commands. If you are running only applications that use the queue manager you do not need to have mqm group authority.

What next?

The connection is now established. You are ready to complete the configuration. Go to “WebSphere MQ for HP-UX configuration.”

WebSphere MQ for HP-UX configuration

Before beginning the installation process ensure that you have first created the *mqm* user and group, and set the password.

Start any channel using the command:

```
runmqchl -c channel.name
```

Note:

1. Sample programs are installed in */opt/mqm/samp*.
2. Error logs are stored in */var/mqm/qmgrs/qmgrname/errors*.
3. When you are using the command interpreter **runmqsc** to enter administration commands, a + at the end of a line indicates that the next line is a continuation. Ensure that there is a space between the last parameter and the continuation character.

Basic configuration

1. Create the queue manager from the UNIX prompt using the command:

```
crtmqm -u dlqname -q hpux
```

where:

hpux Is the name of the queue manager

-q Indicates that this is to become the default queue manager

-u *dlqname*

Specifies the name of the undeliverable message queue

This command creates a queue manager and a set of default objects. It sets the DEADQ attribute of the queue manager but does not create the undeliverable message queue.

2. Start the queue manager from the UNIX prompt using the command:

```
strmqm hpux
```

where *hpux* is the name given to the queue manager when it was created.

Channel configuration

The following section details the configuration to be performed on the HP-UX queue manager to implement the channel described in Figure 32 on page 101.

In each case the MQSC command is shown. Either start **runmqsc** from a UNIX prompt and enter each command in turn, or build the commands into a command file.

Examples are given for connecting WebSphere MQ for HP-UX and WebSphere MQ for Windows. If you wish connect to WebSphere MQ on another platform use the appropriate set of values from the table in place of those for Windows.

Note: The words in **bold** are user-specified and reflect the names of WebSphere MQ objects used throughout these examples. If you change the names used here, ensure that you also change the other references made to these objects throughout this book. All others are keywords and should be entered as shown.

Table 20. Configuration worksheet for WebSphere MQ for HP-UX

ID	Parameter Name	Reference	Example Used	User Value
<i>Definition for local node</i>				
A	Queue Manager Name		HPUX	
B	Local queue name		HPUX.LOCALQ	
<i>Connection to WebSphere MQ for Windows</i>				
The values in this section of the table must match those used in Table 14 on page 146, as indicated.				
C	Remote queue manager name	A	WINNT	
D	Remote queue name		WINNT.REMOTEQ	
E	Queue name at remote system	B	WINNT.LOCALQ	
F	Transmission queue name		WINNT	
G	Sender (SNA) channel name		HPUX.WINNT.SNA	
H	Sender (TCP/IP) channel name		HPUX.WINNT.TCP	
I	Receiver (SNA) channel name	G	WINNT.HPUX.SNA	
J	Receiver (TCP) channel name	H	WINNT.HPUX.TCP	
<i>Connection to WebSphere MQ for AIX</i>				
The values in this section of the table must match those used in Table 18 on page 168, as indicated.				
C	Remote queue manager name	A	AIX	
D	Remote queue name		AIX.REMOTEQ	
E	Queue name at remote system	B	AIX.LOCALQ	
F	Transmission queue name		AIX	
G	Sender (SNA) channel name		HPUX.AIX.SNA	
H	Sender (TCP) channel name		HPUX.AIX.TCP	
I	Receiver (SNA) channel name	G	AIX.HPUX.SNA	
J	Receiver (TCP) channel name	H	AIX.HPUX.TCP	
<i>Connection to WebSphere MQ for HP Tru64 UNIX</i>				
The values in this section of the table must match those used in your HP Tru64 UNIX system.				
C	Remote queue manager name	A	DECUX	
D	Remote queue name		DECUX.REMOTEQ	
E	Queue name at remote system	B	DECUX.LOCALQ	
F	Transmission queue name		DECUX	
H	Sender (TCP) channel name		DECUX.HPUX.TCP	

Table 20. Configuration worksheet for WebSphere MQ for HP-UX (continued)

ID	Parameter Name	Reference	Example Used	User Value
J	Receiver (TCP) channel name	H	HPUX.DECUX.TCP	
Connection to WebSphere MQ for Solaris				
The values in this section of the table must match those used in Table 22 on page 212, as indicated.				
C	Remote queue manager name	A	SOLARIS	
D	Remote queue name		SOLARIS.REMOTEQ	
E	Queue name at remote system	B	SOLARIS.LOCALQ	
F	Transmission queue name		SOLARIS	
G	Sender (SNA) channel name		HPUX.SOLARIS.SNA	
H	Sender (TCP/IP) channel name		HPUX.SOLARIS.TCP	
I	Receiver (SNA) channel name	G	SOLARIS.HPUX.SNA	
J	Receiver (TCP/IP) channel name	H	SOLARIS.HPUX.TCP	
Connection to WebSphere MQ for Linux				
The values in this section of the table must match those used in Table 24 on page 235, as indicated.				
C	Remote queue manager name	A	LINUX	
D	Remote queue name		LINUX.REMOTEQ	
E	Queue name at remote system	B	LINUX.LOCALQ	
F	Transmission queue name		LINUX	
G	Sender (SNA) channel name		HPUX.LINUX.SNA	
H	Sender (TCP/IP) channel name		HPUX.LINUX.TCP	
I	Receiver (SNA) channel name	G	LINUX.HPUX.SNA	
J	Receiver (TCP/IP) channel name	H	LINUX.HPUX.TCP	
Connection to WebSphere MQ for i5/OS				
The values in this section of the table must match those used in Table 36 on page 364, as indicated.				
C	Remote queue manager name	A	AS400	
D	Remote queue name		AS400.REMOTEQ	
E	Queue name at remote system	B	AS400.LOCALQ	
F	Transmission queue name		AS400	
G	Sender (SNA) channel name		HPUX.AS400.SNA	
H	Sender (TCP/IP) channel name		HPUX.AS400.TCP	
I	Receiver (SNA) channel name	G	AS400.HPUX.SNA	
J	Receiver (TCP) channel name	H	AS400.HPUX.TCP	
Connection to WebSphere MQ for z/OS				
The values in this section of the table must match those used in Table 28 on page 272, as indicated.				
C	Remote queue manager name	A	MVS	
D	Remote queue name		MVS.REMOTEQ	
E	Queue name at remote system	B	MVS.LOCALQ	
F	Transmission queue name		MVS	
G	Sender (SNA) channel name		HPUX.MVS.SNA	
H	Sender (TCP) channel name		HPUX.MVS.TCP	

Table 20. Configuration worksheet for WebSphere MQ for HP-UX (continued)

ID	Parameter Name	Reference	Example Used	User Value
I	Receiver (SNA) channel name	G	MVS.HPUX.SNA	
J	Receiver (TCP) channel name	H	MVS.HPUX.TCP	
Connection to MQSeries for VSE/ESA				
The values in this section of the table must match those used in your VSE/ESA system.				
C	Remote queue manager name	A	VSE	
D	Remote queue name		VSE.REMOTEQ	
E	Queue name at remote system	B	VSE.LOCALQ	
F	Transmission queue name		VSE	
G	Sender channel name		HPUX.VSE.SNA	
I	Receiver channel name	G	VSE.HPUX.SNA	

WebSphere MQ for HP-UX sender-channel definitions using SNA:

```
def ql (WINNT) +                                     F
    usage(xmitq) +
    replace

def qr (WINNT.REMOTEQ) +                             D
    rname(WINNT.LOCALQ) +                             E
    rqmname(WINNT) +                                   C
    xmitq(WINNT) +                                     F
    replace

def chl (HPUX.WINNT.SNA) chltype(sdr) +              G
    trptype(lu62) +
    conname('WINNTCPIC') +                             16
    xmitq(WINNT) +                                     F
    replace
```

WebSphere MQ for HP-UX receiver-channel definitions using SNA:

```
def ql (HPUX.LOCALQ) replace                           B

def chl (WINNT.HPUX.SNA) chltype(rcvr) +              I
    trptype(lu62) +
    replace
```

WebSphere MQ for HP-UX invocable TP setup:

This is not required for HP SNAplus2 Release 6.

During the HP SNAplus2 configuration process, you created an invocable TP definition, which points to an executable file. In the example, the file was called /users/interop/HPUX.crs6a. You can choose what you call this file, but you are recommended to include the name of your queue manager in the name. The contents of the executable file must be:

```
#!/bin/sh
/opt/mqm/bin/amqcrs6a -m hpux
```

where *hpux* is the name of your queue manager A.

This ensures that SNA receiver channels activate correctly when a sender channel initiates a conversation.

WebSphere MQ for HP-UX sender-channel definitions using TCP:

```
def ql (WINNT) +                               F
  usage(xmitq) +
  replace

def qr (WINNT.REMOTEQ) +                       D
  rname(WINNT.LOCALQ) +                       E
  rqmname(WINNT) +                             C
  xmitq(WINNT) +                               F
  replace

def chl (HPUX.WINNT.TCP) chltype(sdr) +       H
  trptype(tcp) +
  conname(remote_tcpip_hostname) +
  xmitq(WINNT) +                               F
  replace
```

WebSphere MQ for HP-UX receiver-channel definitions using TCP/IP:

```
def ql (HPUX.LOCALQ) replace                   B

def chl (WINNT.HPUX.TCP) chltype(rcvr) +      J
  trptype(tcp) +
  replace
```

Example configuration - IBM WebSphere MQ for Solaris

This chapter gives an example of how to set up communication links from WebSphere MQ for Solaris to WebSphere MQ products on the following platforms:

- Windows
- AIX
- HP Tru64 UNIX
- HP-UX
- Linux
- i5/OS
- z/OS
- VSE/ESA

WebSphere MQ allows you to set up communication links from WebSphere MQ for Solaris using SNAP-IX V6.2 or later. See “Configuration parameters for an LU 6.2 connection using SNAP-IX” on page 195, and “Establishing a session using SNAP-IX” on page 199.

To establish a TCP connection, see “Establishing a TCP connection” on page 210.

Once the connection is established, you need to define some channels to complete the configuration. This is described in “WebSphere MQ for Solaris configuration” on page 211.

See “Example configuration chapters in this book” on page 101 for background information about this chapter and how to use it.

Configuration parameters for an LU 6.2 connection using SNAP-IX

Table 21 presents a worksheet listing all the parameters needed to set up communication from Solaris using SNAP-IX to one of the other WebSphere MQ platforms. The worksheet shows examples of the parameters, which have been tested in a working environment, and leaves space for you to fill in your own values. An explanation of the parameter names follows the worksheet. Use the worksheet for the platform to which you are connecting.

Configuration worksheet

Use this worksheet to record the values you use for your configuration. Where numbers appear in the Reference column they indicate that the value must match that in the appropriate worksheet elsewhere in this book. The examples that follow in this chapter refer to the values in the ID column. The entries in the Parameter Name column are explained in "Explanation of terms" on page 197.

Table 21. Configuration worksheet for SNAP-IX

ID	Parameter Name	Ref.	Example	User Value
<i>Parameters for local node</i>				
1	Configuration file name		sna_node.cfg	
2	Control point name		SOLARXPU	
3	Node ID to send		05D 23456	
4	Network name		NETID	
5	Local APPC LU		SOLARXLU	
6	APPC mode		#INTER	
7	Invokable TP		MQSERIES	
8	Local MAC address		08002071CC8A	
9	Port name		MQPORT	
10	Command path		/opt/mqm/bin/amqcrs6a	
11	Local queue manager		SOLARIS	
<i>Connection to a Windows system</i>				
The values in this section of the table must match those used in the Table for Windows and LU6.2, as indicated.				
12	Link station name		NTCONN	
13	Network name	2	NETID	
14	CP name	3	WINNTCP	
15	Remote LU	5	WINNTLU	
16	Application TP	7	MQSERIES	
17	Mode name	17	#INTER	
18	CPI-C symbolic destination name		NTCPIC	
19	Remote network address	9	08005AA5FAB9	
20	Node ID to receive	4	05D 30F65	
<i>Connection to an AIX system</i>				
The values in this section of the table must match those used in the Table for AIX and LU6.2, as indicated.				
12	Link station name		AIXCONN	

Table 21. Configuration worksheet for SNAP-IX (continued)

ID	Parameter Name	Ref.	Example	User Value
13	Network name	1	NETID	
14	CP name	2	AIXPU	
15	Remote LU	4	AIXLU	
16	Application TP	6	MQSERIES	
17	Mode name	14	#INTER	
18	CPI-C symbolic destination name		AIXCPIC	
19	Remote network address	8	123456789012	
20	Node ID to receive	3	071 23456	
Connection to an HP-UX system				
The values in this section of the table must match those used in the Table for HP-UX and LU6.2, as indicated.				
12	Link station name		HPUXCONN	
13	Network name	2	NETID	
14	CP name	3	HPUXPU	
15	Remote LU	7	HPUXLU	
16	Application TP	8	MQSERIES	
17	Mode name	17	#INTER	
18	CPI-C symbolic destination name		HPUXCPIC	
19	Remote network address	5	10005FC5D83	
20	node ID to receive	6	05D 54321	
Connection to a Linux (x86 platform) system				
The values in this section of the table must match those used in the table the Table for Linux (x86 platform) and LU6.2, as indicated.				
12	Link station name		LXCONN	
13	Network name	4	NETID	
14	CP name	2	LINUXPU	
15	Remote LU	5	LINUXLU	
16	Application TP	7	MQSERIES	
17	Mode name	6	#INTER	
18	CPI-C symbolic destination name		LXCPIC	
19	Remote network address	8	08005AC6DF33	
20	Node ID to receive	3	05D 30A55	
Connection to an i5/OS system				
The values in this section of the table must match those used in the Table for i5/OS and LU6.2, as indicated.				
12	Link station name		AS4CONN	
13	Network name	1	NETID	
14	CP name	2	AS400PU	
15	Remote LU	3	AS400LU	
16	Application TP	8	MQSERIES	
17	Mode name	17	#INTER	

Table 21. Configuration worksheet for SNAP-IX (continued)

ID	Parameter Name	Ref.	Example	User Value
18	CPI-C symbolic destination name		AS4CPIC	
19	Remote network address	4	10005A5962EF	
<i>Connection to a z/OS system</i>				
The values in this section of the table must match those used in the Table for z/OS and LU6.2, as indicated.				
12	Link station name		MVSCONN	
13	Network name	2	NETID	
14	CP name	3	MVSPU	
15	Remote LU	4	MVSLU	
16	Application TP	7	MQSERIES	
17	Mode name	10	#INTER	
18	CPI-C symbolic destination name		MVSCPIC	
19	Remote network address	8	400074511092	
<i>Connection to a VSE/ESA system</i>				
The values in this section of the table must match those used in the Table for VSE/ESA and LU6.2, as indicated.				
12	Link station name		VSECONN	
13	Network name	1	NETID	
14	CP name	2	VSEPU	
15	Remote LU	3	VSELU	
16	Application TP	4	MQ01	MQ01
17	Mode name		#INTER	
18	CPI-C symbolic destination name		VSECPIC	
19	Remote network address	5	400074511092	

Explanation of terms

1 Configuration file name

This is the unique name of the SNAP-IX configuration file. The default for this name is `sna_node.cfg`.

Although it is possible to edit this file, it is strongly recommended that configuration is done using `xsnadmin`.

2 Control point name

This is the unique Control point name for this workstation. In the SNA network, the Control point is an addressable location (PU type 2.1). Your network administrator will assign this to you.

3 Node ID to send

This is the unique ID of this workstation. On other platforms this is often referred to as the *Exchange ID* or *XID*. Your network administrator will assign this ID for you.

4 Network name

This is the unique ID of the network to which you are connected. It is an alphanumeric value and can be 1-8 characters long. The network name works with the Control point name to uniquely identify a system. Your network administrator will tell you the value.

5 Local APPC LU

An LU manages the exchange of data between transactions. The local APPC LU name is the name of the LU on your system. Your network administrator will assign this to you.

6 APPC mode

This is the name given to the set of parameters that control the APPC conversation. This name must be defined at each partner system. Your network administrator will assign this to you.

7 Invokable TP

WebSphere MQ applications trying to converse with this workstation will specify a symbolic name for the program to be run at the receiving end. This will have been defined on the channel definition at the sender. For simplicity, wherever possible use a transaction program name of MQSERIES, or in the case of a connection to VSE/ESA, where the length is limited to 4 bytes, use MQTP.

8 Local MAC address

This is the network address of the token-ring card. The address to be specified is found in the ether value displayed in response to the `ifconfig tr0` command issued at a root level of authority. (Tr0 is the name of the machine's token-ring interface.) If you do not have the necessary level of authority, your Sun Solaris system administrator can tell you the value.

9 Port name

This is a meaningful symbolic name that is used to associate the definitions with a network interface (in this case, a Token-Ring adapter). A separate Port must be defined for each physical device attached to the workstation.

10 Full path to executable

This is the path and name of the script file that invokes the WebSphere MQ program to run.

11 Local queue manager

This is the name of the queue manager on your local system.

10 Link station name

This is a meaningful symbolic name by which the connection to a peer or host node is known. It defines a logical path to the remote system. Its name is used only inside SNAP-IX and is specified by you. The connection must be associated with an existing Link and owned by one local node. You must define one connection for each partner or host system.

18 CPI-C symbolic destination name

This is a name given to the definition of a partner node. You choose the name. It need be unique only on this machine. Later you can use this name in the WebSphere MQ sender channel definition.

20 Node ID to receive

This is the unique ID of the partner workstation with which you will be communicating. On other platforms this is often referred to as the *Exchange ID* or *XID*. For a connection to a host system any values except 000 FFFFF and FFF FFFFF may be specified. Your network administrator will assign this ID for you.

Establishing a session using SNAP-IX

The following information guides you through the tasks you must perform to create the SNA infrastructure that WebSphere MQ requires. This example creates the definitions for a partner node and LU on OS/2.

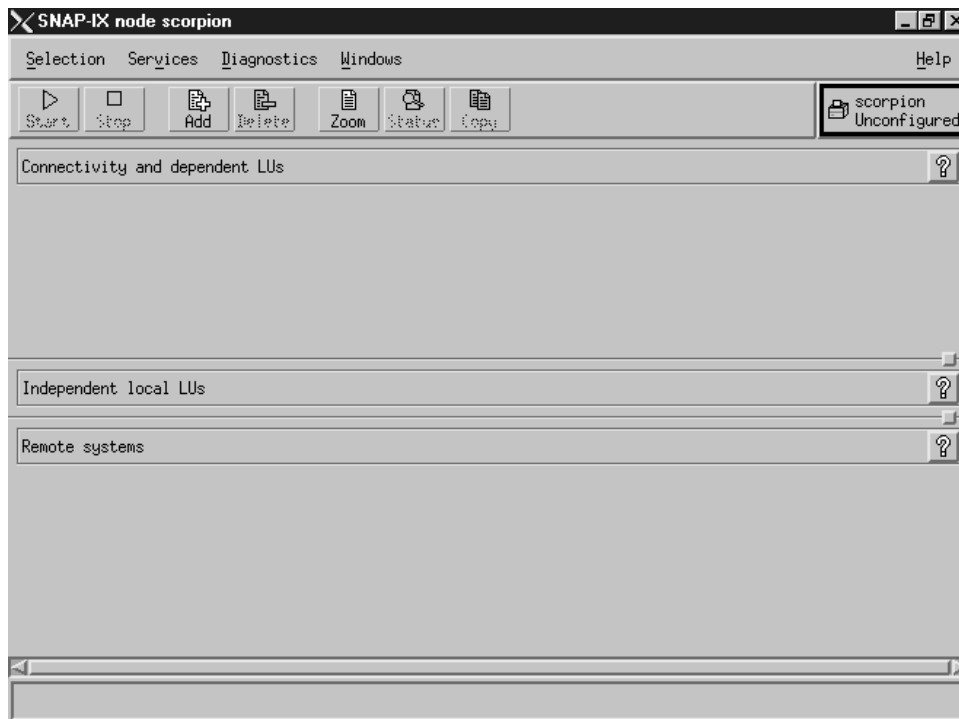
Use **sna start** followed by **x snaadmin** to type the SNAP-IX configuration panels. You need root authority to use **x snaadmin**.

SNAP-IX configuration

SNAP-IX configuration involves the following steps:

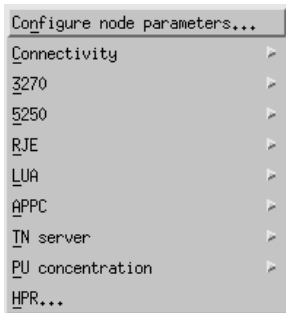
1. Defining a local node
2. Adding a Token Ring Port
3. Defining a local LU

The SNAP-IX main menu, from which you start, is shown here:

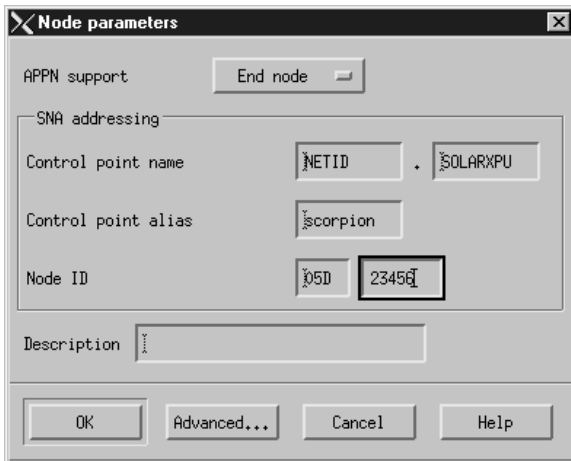


Defining a local node:

1. From the SNAP-IX main menu, click the **Services** pull-down. The **Services** pull-down appears::



2. Click **Configure node parameters**. The following panel is displayed:



3. Complete the **Control point name** with the values **Network name** (4) and **Control point name** (2).
4. Type the **Control point name** (2) in the **Control point alias** field.
5. Type the **Node ID** (3).
6. Click **End node**.
7. Click **OK**.

A default independent local LU is defined.

Adding a Token Ring Port:

1. From the main SNAP-IX menu, click **Connectivity and dependent LUs**.
2. Click **Add**. The following panel is displayed:



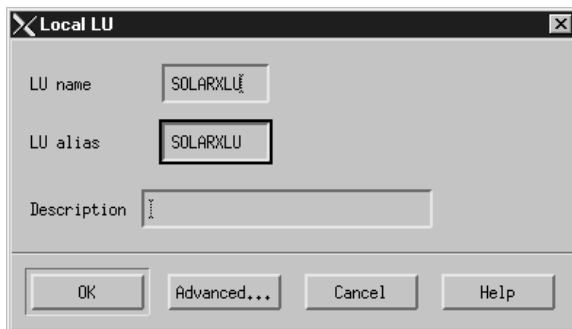
3. Click **Token Ring Card** and click **OK**. The following panel is displayed:



4. Type the **SNA port name** (9).
5. Type a **Description** and click **OK** to take the default values.

Defining a local LU:

1. From the main SNAP-IX menu, click **Independent local LUs**.
2. Click **Add**. The following panel is displayed:



3. Type the **LU name** (5) and click **OK**.

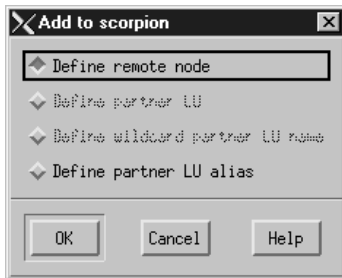
APPC configuration

APPC configuration involves the following steps:

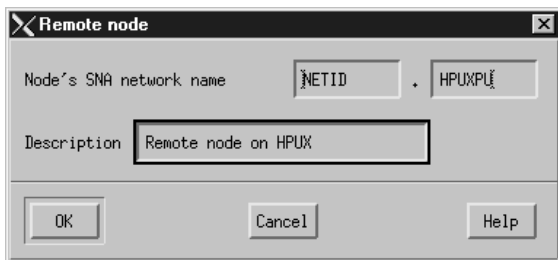
1. Defining a remote node
2. Defining a partner LU
3. Defining a link station
4. Defining a mode
5. Adding CPI-C information
6. Adding a TP definition

Defining a remote node:

1. From the main SNAP-IX menu, click **Remote systems**.
2. Click **Add**. The following panel is displayed:



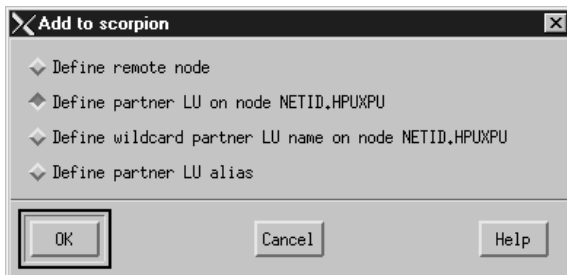
3. Select the **Define remote node** check box and click **OK**. The following panel is displayed:



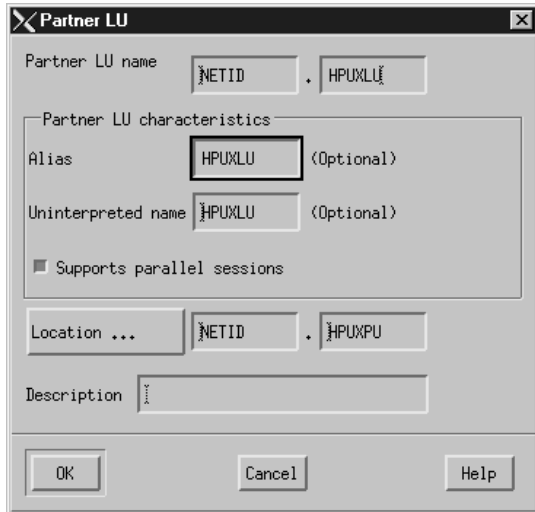
4. Type the **Node's SNA network name** (13) and a **Description**.
5. Click **OK**.
6. A default partner LU with the same name is generated and a message is displayed.
7. Click **OK**.

Defining a partner LU:

1. From the main SNAP-IX menu, click **Remote systems** and click the remote node.
2. Click **Add**. The following panel is displayed:



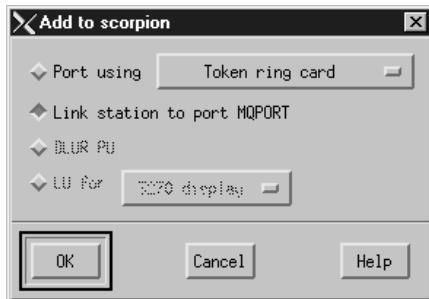
3. Select the **Define partner LU on node node name** check box.
4. Click **OK**. The following panel is displayed:



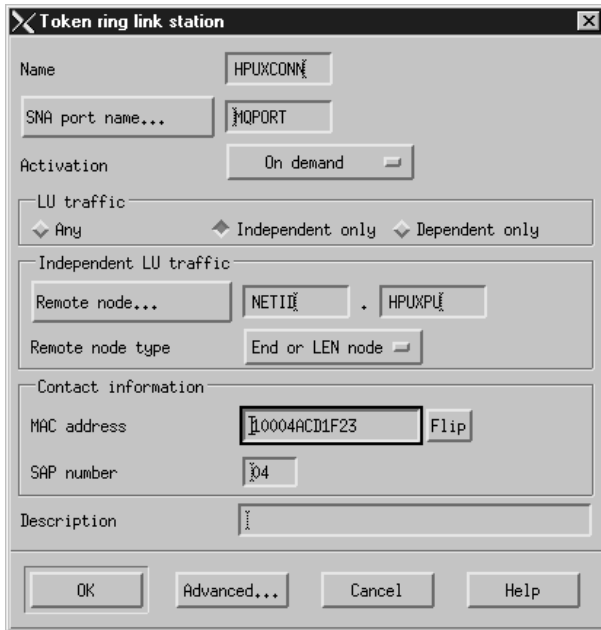
5. Type the **partner LU name** (15) and click **OK**.

Defining a link station:

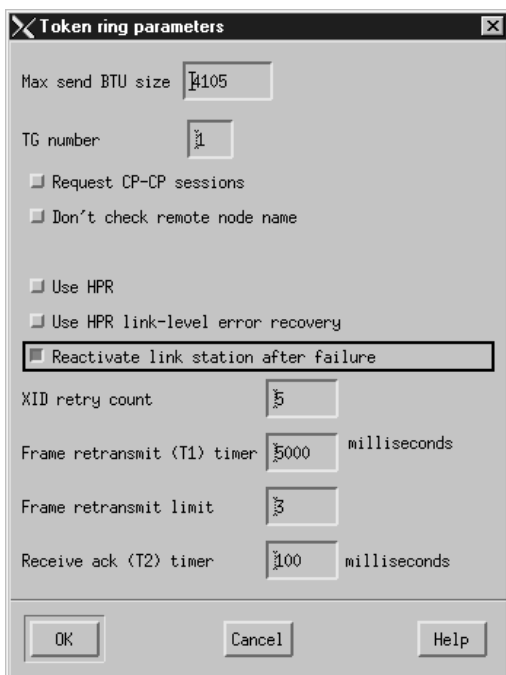
1. From the main SNAP-IX menu, click **Connectivity and dependent LUs**.
2. Click the MQPORT port.
3. Click **Add**. The following panel is displayed:



4. Select the **Add link station to port MQPORT** check box.
5. Click **OK**. The following panel is displayed:



6. Type the **Name** of the link station (12).
7. Set the value of **Activation** to “On demand”.
8. Select the **Independent only** check box.
9. Click **Remote node** and select the value of the remote node (14).
10. Click **OK**.
11. Set the value of **Remote node type** to “End or LEN node”.
12. Type the value for **MAC address** (19) and click **Advanced**. The following panel is displayed:

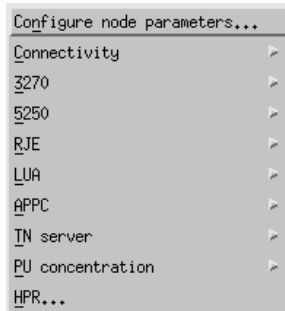


13. Select the **Request CP-CP sessions**. check box

14. Select the **Reactivate link station after failure**. check box
15. Click **OK** to exit the Advanced panel.
16. Click **OK** again.

Defining a mode:

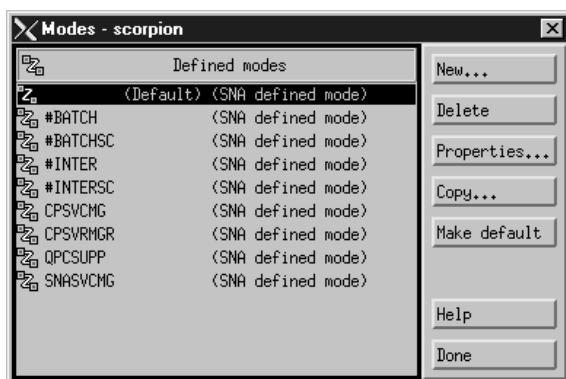
1. From the SNAP-IX main menu, click the **Services** pull-down: The following panel is displayed:



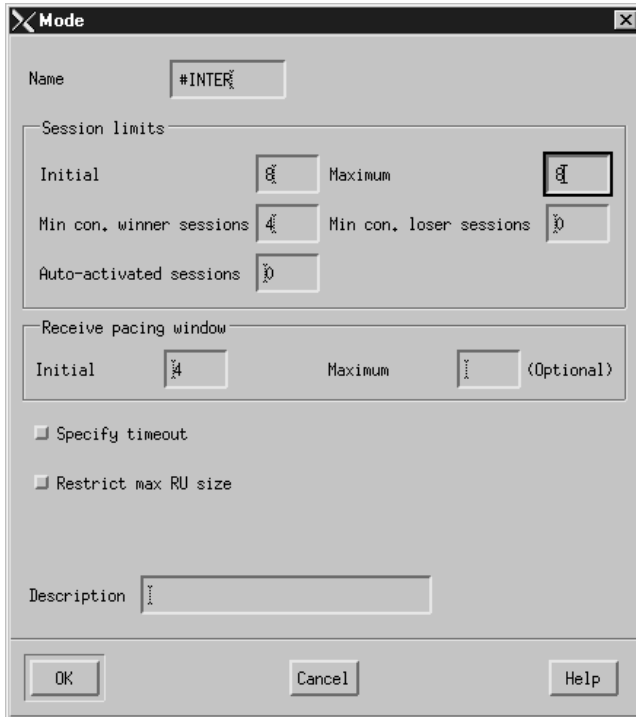
2. Click **APPC**. The following panel is displayed:



3. Click **Modes**. The following panel is displayed:



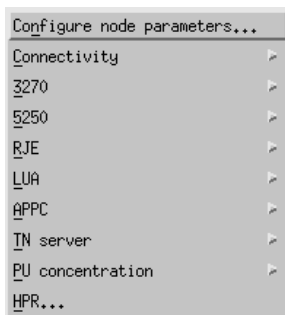
4. Click **Add**. The following panel is displayed:



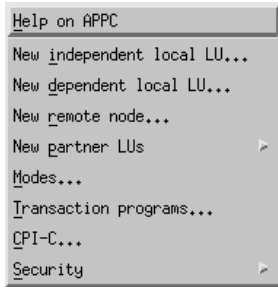
5. Type the **Name** to be given to the mode (17).
6. Set the values of **Initial session limit** to 8, **Min con. winner sessions** to 4, and **Auto-activated sessions** to 0.
7. Click **OK**.
8. Click **Done**.

Adding CPI-C information:

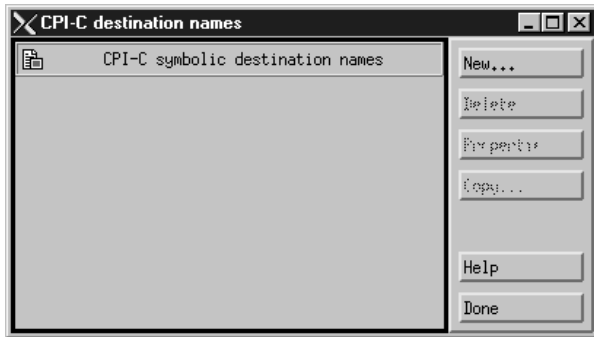
1. From the SNAP-IX main menu, click the **Services** pull-down:



2. Click **APPC**. The following panel is displayed:



3. Click **CPI-C**. The following panel is displayed:



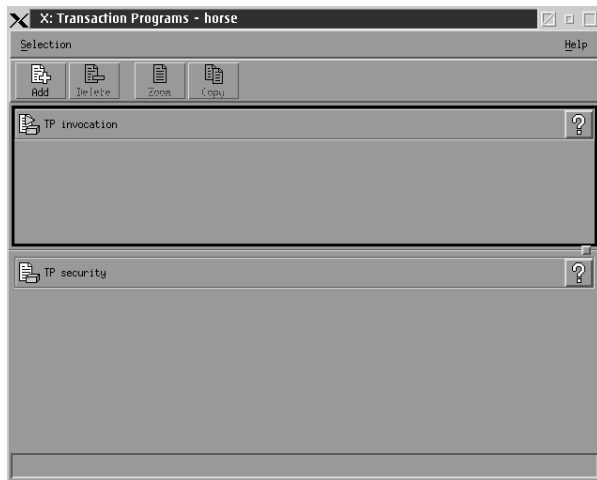
4. Click **Add**. The following panel is displayed:

5. Type the **Name** (18). Select the **Application TP** check box and type the value (16). Select the **Use PLU alias** check box and type the name (15). Type the **Mode** name (17).
6. Click **OK**.

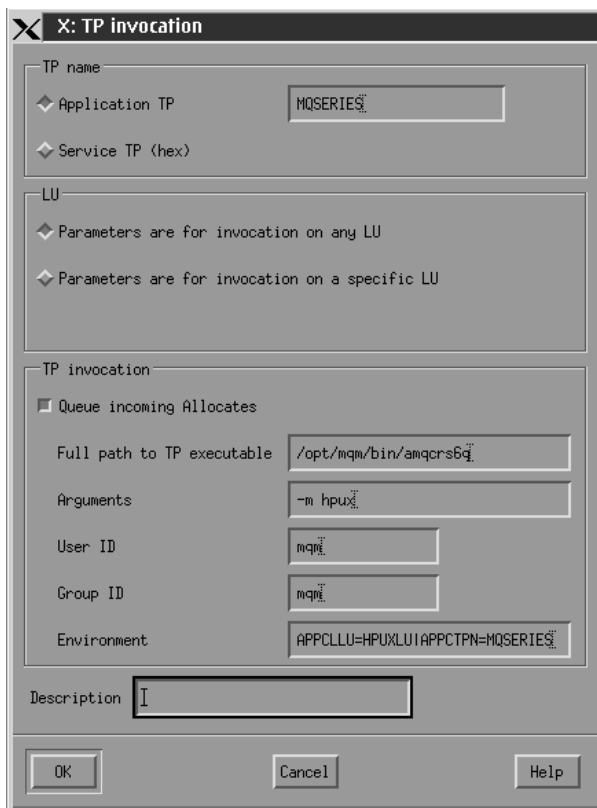
Adding a TP definition using SNAP-IX Release 6:

To add a TP definition:

1. Click the **Services** pull-down and click **APPC** as for CPI-C information.
2. Click **Transaction Programs**. The following panel is displayed:



3. Click **Add**. The following panel is displayed:



4. Type **TP name** (7) in the **Application TP** field.
5. Clear the **Queue incoming Allocates** check box.
6. Type **Full path to executable** (10).
7. Type **-m Local queue manager** (11) in the **Arguments** field.
8. Type **mqm** in the **User ID** and **Group ID** fields.
9. Type environment variables **APPCLU=local LU** (5) and **APPCTPN=Invokable TP** (7) separated by the pipe character in the **Environment** field.
10. Click **OK** to save your definition.

SNAP-IX operation

The SNAP-IX control daemon is started with the **sna start** command. Depending on the options selected at installation, it may already be running.

The **xsnaadmin** utility controls SNAP-IX resources.

Logging and tracing are controlled from here. Log and trace files can be found in the `/var/opt/sna` directory. The logging files `sna.aud` and `sna.err` can be read using a standard editor such as `vi`.

In order to read the trace files **sna1.trc** and **sna2.trc** you must first format them by running the command **sna1.trc -f sna1.trc -o sna1**. This produces a `sna1.dmp` file that can be read using a normal editor.

It is possible to edit the configuration file, but this is not a recommended method of configuring SNAP-IX.

The APPCLLU environment variables must be set before starting a sender channel from the Solaris system. The command can be either entered interactively or added to the logon profile. Depending on the level of BOURNE shell or KORN shell program being used, the command will be:

```
export APPCLLU=SOLARXLU 5 newer level
```

or

```
APPCLLU=SOLARXLU 5 older level  
export
```

What next?

The connection is now established. You are ready to complete the configuration. Go to “WebSphere MQ for Solaris configuration” on page 211.

Establishing a TCP connection

To establish a TCP connection, follow these steps.

1. Edit the file `/etc/services`.

Note: To edit the `/etc/services` file, you must be logged in as a superuser or root. If you do not have the following line in that file, add it as shown:

```
MQSeries 1414/tcp # MQSeries channel listener
```

2. Edit the file `/etc/inetd.conf`. If you do not have the following line in that file, add it as shown:

```
MQSeries stream tcp nowait mqm /opt/mqm/bin/amqcrsta amqcrsta  
[-m queue.manager.name]
```

3. Find the process ID of the `inetd` with the command:

```
ps -ef | grep inetd
```

4. Run the appropriate command, as follows:

- For Solaris 9:

```
kill -1 inetd processid
```

- For Solaris 10 or later:

```
inetconv
```

What next?

The TCP/IP connection is now established. You are ready to complete the configuration. Go to “WebSphere MQ for Solaris configuration.”

WebSphere MQ for Solaris configuration

Before beginning the installation process ensure that you have first created the *mqm* user and group, and set the password.

Start any channel using the command:

```
runmqchl -c channel.name
```

Note:

1. Sample programs are installed in `/opt/mqm/samp`.
2. Error logs are stored in `/var/mqm/qmgrs/qmgrname/errors`.
3. When you are using the command interpreter **runmqsc** to enter administration commands, a + at the end of a line indicates that the next line is a continuation. Ensure that there is a space between the last parameter and the continuation character.
4. For an SNA or LU6.2 channel, if you experience an error when you try to load the communications library, probably file `liblu62.so` cannot be found. A likely solution to this problem is to add its location, which is probably `/opt/SUNWlu62`, to `LD_LIBRARY_PATH`.

Basic configuration

1. Create the queue manager from the UNIX prompt using the command:

```
crtmqm -u dlqname -q solaris
```

where:

solaris

Is the name of the queue manager

-q Indicates that this is to become the default queue manager

-u *dlqname*

Specifies the name of the undeliverable message queue

This command creates a queue manager and a set of default objects.

2. Start the queue manager from the UNIX prompt using the command:

```
strmqm solaris
```

where *solaris* is the name given to the queue manager when it was created.

Channel configuration

The following section details the configuration to be performed on the Solaris queue manager to implement the channel described in Figure 32 on page 101.

The MQSC command to create each object is shown. Either start **runmqsc** from a UNIX prompt and enter each command in turn, or build the commands into a command file.

Examples are given for connecting WebSphere MQ for Solaris and WebSphere MQ for Windows. If you wish to connect to WebSphere MQ on another platform use the appropriate set of values from the table in place of those for Windows.

Note: The words in **bold** are user-specified and reflect the names of WebSphere MQ objects used throughout these examples. If you change the names used here, ensure that you also change the other references made to these objects throughout this book. All others are keywords and should be entered as shown.

Table 22. Configuration worksheet for WebSphere MQ for Solaris

ID	Parameter Name	Reference	Example Used	User Value
<i>Definition for local node</i>				
A	Queue Manager Name		SOLARIS	
B	Local queue name		SOLARIS.LOCALQ	
<i>Connection to WebSphere MQ for Windows</i>				
The values in this section of the table must match those used in Table 14 on page 146, as indicated.				
C	Remote queue manager name	A	WINNT	
D	Remote queue name		WINNT.REMOTEQ	
E	Queue name at remote system	B	WINNT.LOCALQ	
F	Transmission queue name		WINNT	
G	Sender (SNA) channel name		SOLARIS.WINNT.SNA	
H	Sender (TCP/IP) channel name		SOLARIS.WINNT.TCP	
I	Receiver (SNA) channel name	G	WINNT.SOLARIS.SNA	
J	Receiver (TCP) channel name	H	WINNT.SOLARIS.TCP	
<i>Connection to WebSphere MQ for AIX</i>				
The values in this section of the table must match those used in Table 18 on page 168, as indicated.				
C	Remote queue manager name	A	AIX	
D	Remote queue name		AIX.REMOTEQ	
E	Queue name at remote system	B	AIX.LOCALQ	
F	Transmission queue name		AIX	
G	Sender (SNA) channel name		SOLARIS.AIX.SNA	
H	Sender (TCP) channel name		SOLARIS.AIX.TCP	
I	Receiver (SNA) channel name	G	AIX.SOLARIS.SNA	
J	Receiver (TCP) channel name	H	AIX.SOLARIS.TCP	
<i>Connection to MQSeries for Compaq Tru64 UNIX</i>				
The values in this section of the table must match those used in your Compaq Tru64 UNIX system.				
C	Remote queue manager name	A	DECUX	
D	Remote queue name		DECUX.REMOTEQ	
E	Queue name at remote system	B	DECUX.LOCALQ	
F	Transmission queue name		DECUX	
H	Sender (TCP) channel name		DECUX.SOLARIS.TCP	
J	Receiver (TCP) channel name	H	SOLARIS.DECUX.TCP	

Table 22. Configuration worksheet for WebSphere MQ for Solaris (continued)

ID	Parameter Name	Reference	Example Used	User Value
Connection to WebSphere MQ for HP-UX				
The values in this section of the table must match those used in Table 20 on page 191, as indicated.				
C	Remote queue manager name	A	HPUX	
D	Remote queue name		HPUX.REMOTEQ	
E	Queue name at remote system	B	HPUX.LOCALQ	
F	Transmission queue name		HPUX	
G	Sender (SNA) channel name		SOLARIS.HPUX.SNA	
H	Sender (TCP) channel name		SOLARIS.HPUX.TCP	
I	Receiver (SNA) channel name	G	HPUX.SOLARIS.SNA	
J	Receiver (TCP/IP) channel name	H	HPUX.SOLARIS.TCP	
Connection to WebSphere MQ for Linux				
The values in this section of the table must match those used in Table 24 on page 235, as indicated.				
C	Remote queue manager name	A	LINUX	
D	Remote queue name		LINUX.REMOTEQ	
E	Queue name at remote system	B	LINUX.LOCALQ	
F	Transmission queue name		LINUX	
G	Sender (SNA) channel name		SOLARIS.LINUX.SNA	
H	Sender (TCP/IP) channel name		SOLARIS.LINUX.TCP	
I	Receiver (SNA) channel name	G	LINUX.SOLARIS.SNA	
J	Receiver (TCP/IP) channel name	H	LINUX.SOLARIS.TCP	
Connection to WebSphere MQ for i5/OS				
The values in this section of the table must match those used in Table 36 on page 364, as indicated.				
C	Remote queue manager name	A	AS400	
D	Remote queue name		AS400.REMOTEQ	
E	Queue name at remote system	B	AS400.LOCALQ	
F	Transmission queue name		AS400	
G	Sender (SNA) channel name		SOLARIS.AS400.SNA	
H	Sender (TCP) channel name		SOLARIS.AS400.TCP	
I	Receiver (SNA) channel name	G	AS400.SOLARIS.SNA	
J	Receiver (TCP) channel name	H	AS400.SOLARIS.TCP	
Connection to WebSphere MQ for z/OS				
The values in this section of the table must match those used in Table 28 on page 272, as indicated.				
C	Remote queue manager name	A	MVS	
D	Remote queue name		MVS.REMOTEQ	
E	Queue name at remote system	B	MVS.LOCALQ	
F	Transmission queue name		MVS	
G	Sender (SNA) channel name		SOLARIS.MVS.SNA	
H	Sender (TCP) channel name		SOLARIS.MVS.TCP	
I	Receiver (SNA) channel name	G	MVS.SOLARIS.SNA	

Table 22. Configuration worksheet for WebSphere MQ for Solaris (continued)

ID	Parameter Name	Reference	Example Used	User Value
J	Receiver (TCP) channel name	H	MVS.SOLARIS.TCP	
<i>Connection to MQSeries for VSE/ESA</i>				
The values in this section of the table must match those used in your VSE/ESA system.				
C	Remote queue manager name	A	VSE	
D	Remote queue name		VSE.REMOTEQ	
E	Queue name at remote system	B	VSE.LOCALQ	
F	Transmission queue name		VSE	
G	Sender channel name		SOLARIS.VSE.SNA	
I	Receiver channel name	G	VSE.SOLARIS.SNA	

WebSphere MQ for Solaris sender-channel definitions using SNAP-IX SNA:

```
def ql (WINNT) +                                     F
  usage(xmitq) +
  replace

def qr (WINNT.REMOTEQ) +                             D
  rname(WINNT.LOCALQ) +                             E
  rqmname(WINNT) +                                  C
  xmitq(WINNT) +                                     F
  replace

def chl (SOLARIS.WINNT.SNA) chltype(sdr) +          G
  trptype(lu62) +
  conname('NTCPIC') +                               14
  xmitq(WINNT) +                                     F
  replace
```

WebSphere MQ for Solaris receiver-channel definitions using SNA:

```
def ql (SOLARIS.LOCALQ) replace                       B

def chl (WINNT.SOLARIS.SNA) chltype(rcvr) +         I
  trptype(lu62) +
  replace
```

WebSphere MQ for Solaris sender-channel definitions using TCP:

```
def ql (WINNT) +                                     F
  usage(xmitq) +
  replace

def qr (WINNT.REMOTEQ) +                             D
  rname(WINNT.LOCALQ) +                             E
  rqmname(WINNT) +                                  C
  xmitq(WINNT) +                                     F
  replace

def chl (SOLARIS.WINNT.TCP) chltype(sdr) +          H
  trptype(tcp) +
  conname(remote_tcpip_hostname) +
  xmitq(WINNT) +                                     F
  replace
```

WebSphere MQ for Solaris receiver-channel definitions using TCP/IP:

```
def q1 (SOLARIS.LOCALQ) replace B
def chl (WINNT.SOLARIS.TCP) chltype(rcvr) + J
  trptype(tcp) +
  replace
```

Example configuration - IBM WebSphere MQ for Linux

This chapter gives an example of how to set up communication links from WebSphere MQ for Linux to WebSphere MQ products on the following platforms:

- Windows
- AIX
- Compaq Tru64 UNIX
- HP-UX
- Solaris
- i5/OS
- z/OS
- VSE/ESA

This chapter contains the following sections:

- “Configuration parameters for an LU 6.2 connection”
- “Establishing a session using Communications Server for Linux” on page 219
- “Establishing a TCP connection” on page 232
- “WebSphere MQ for Linux configuration” on page 234

See “Example configuration chapters in this book” on page 101 for background information about this chapter and how to use it.

Configuration parameters for an LU 6.2 connection

Use this worksheet to record the values you use for your configuration

Note: The information in this section applies only to WebSphere MQ for Linux (x86 platform). It does not apply to WebSphere MQ for Linux (x86-64 platform), WebSphere MQ for Linux (zSeries® s390x platform), or WebSphere MQ for Linux (POWER™ platform).

Table 23 on page 216 presents a worksheet listing all the parameters needed to set up communication from Linux to one of the other WebSphere MQ platforms using Communications Server for Linux. The worksheet shows examples of the parameters, which have been tested in a working environment, and leaves space for you to fill in your own values. An explanation of the parameter names follows the worksheet. Use the worksheet for the platform to which you are connecting.

Use this worksheet to record the values you use for your configuration. Where numbers appear in the Reference column they indicate that the value must match that in the appropriate worksheet elsewhere in this information centre. The examples that follow in this section refer to the values in the ID column. The entries in the Parameter Name column are explained in “Explanation of terms” on page 218.

Table 23. Configuration worksheet for Communications Server for Linux

ID	Parameter Name	Reference	Example	User Value
<i>Parameters for local node</i>				
1	Configuration file name		sna_node.cfg	
2	Control point name		LINUXPU	
3	Node ID to send		05D 30A55	
4	Network name		NETID	
5	Local APPC LU		LINUXLU	
6	APPC mode		#INTER	
7	Invokable TP		MQSERIES	
8	Local MAC address		08005AC6DF33	
9	Port name		MQPORT	
10	Command path		/opt/mqm/bin/ amqcrs6a	
11	Local queue manager		LINUX	
<i>Connection to a Windows system</i>				
The values in this section of the table must match those used in Table 13 on page 130, as indicated.				
12	Link station name		NTCONN	
13	Network name	2	NETID	
14	CP name	3	WINNTCP	
15	Remote LU	5	WINNTLU	
16	Application TP	7	MQSERIES	
17	Mode name	17	#INTER	
18	CPI-C symbolic destination name		NTCPIC	
19	Remote network address	9	08005AA5FAB9	
20	Node ID to receive	4	05D 30F65	
<i>Connection to an AIX system</i>				
The values in this section of the table must match those used in Table 17 on page 156, as indicated.				
12	Link station name		AIXCONN	
13	Network name	1	NETID	
14	CP name	2	AIXPU	
15	Remote LU	4	AIXLU	
16	Application TP	6	MQSERIES	
17	Mode name	9	#INTER	
18	CPI-C symbolic destination name		AIXCPIC	
19	Remote network address	8	123456789012	
20	Node ID to receive	3	071 23456	

Table 23. Configuration worksheet for Communications Server for Linux (continued)

ID	Parameter Name	Reference	Example	User Value
Connection to an HP-UX system				
The values in this section of the table must match those used in Table 19 on page 172, as indicated.				
12	Link station name		HPUXCONN	
13	Network name	4	NETID	
14	CP name	2	HPUXPU	
15	Remote LU	5	HPUXLU	
16	Application TP	7	MQSERIES	
17	Mode name	6	#INTER	
18	CPI-C symbolic destination name		HPUXCPIC	
19	Remote network address	8	100090DC2C7C	
20	node ID to receive	3	05D 54321	
Connection to a Solaris system				
The values in this section of the table must match those used in Table 21 on page 195, as indicated.				
12	Link station name		SOLCONN	
13	Network name	2	NETID	
14	CP name	3	SOLARPU	
15	Remote LU	7	SOLARLU	
16	Application TP	8	MQSERIES	
17	Mode name	17	#INTER	
18	CPI-C symbolic destination name		SOLCPIC	
19	Remote network address	5	08002071CC8A	
20	Node ID to receive	6	05D 310D6	
Connection to an i5/OS system				
The values in this section of the table must match those used in Table 35 on page 351, as indicated.				
12	Link station name		AS4CONN	
13	Network name	1	NETID	
14	CP name	2	AS400PU	
15	Remote LU	3	AS400LU	
16	Application TP	8	MQSERIES	
17	Mode name	17	#INTER	
18	CPI-C symbolic destination name		AS4CPIC	
19	Remote network address	4	10005A5962EF	
Connection to a z/OS system				
The values in this section of the table must match those used in Table 27 on page 267, as indicated.				

Table 23. Configuration worksheet for Communications Server for Linux (continued)

ID	Parameter Name	Reference	Example	User Value
12	Link station name		MVSCONN	
13	Network name	2	NETID	
14	CP name	3	MVSPU	
15	Remote LU	4	MVSLU	
16	Application TP	7	MQSERIES	
17	Mode name	6	#INTER	
18	CPI-C symbolic destination name		MVSCPIC	
19	Remote network address	8	400074511092	
Connection to a VSE/ESA system				
The values in this section of the table must match those used in your VSE/ESA system.				
12	Link station name		VSECONN	
13	Network name	1	NETID	
14	CP name	2	VSEPU	
15	Remote LU	3	VSELU	
16	Application TP	4	MQ01	
17	Mode name		#INTER	
18	CPI-C symbolic destination name		VSECPIC	
19	Remote network address	5	400074511092	

Explanation of terms

1 Configuration file name

This is the unique name of the Communications Server for Linux configuration file. The default for this name is `sna_node.cfg`.

Although it is possible to edit this file, it is strongly recommended that configuration is done using `xsnadmin`.

2 Control point name

This is the unique control point name for this workstation. In the SNA network, the control point is an addressable location (PU type 2.1). Your network administrator will assign this to you.

3 Node ID to send

This is the unique ID of this workstation. On other platforms this is often referred to as the *Exchange ID* or *XID*. Your network administrator will assign this ID for you.

4 Network name

This is the unique ID of the network to which you are connected. It is an alphanumeric value and can be 1-8 characters long. The network name works with the control point name to uniquely identify a system. Your network administrator will tell you the value.

5 Local APPC LU

An LU manages the exchange of data between transactions. The local APPC LU name is the name of the LU on your system. Your network administrator will assign this to you.

6 APPC mode

This is the name given to the set of parameters that control the APPC conversation. This name must be defined at each partner system. Your network administrator will assign this to you.

7 Invokable TP

WebSphere MQ applications trying to converse with this workstation will specify a symbolic name for the program to be run at the receiving end. This will have been defined on the channel definition at the sender. For simplicity, wherever possible use a transaction program name of MQSERIES, or in the case of a connection to VSE/ESA, where the length is limited to 4 bytes, use MQ01.

8 Local MAC address

This is the network address of the token-ring card. The address to be specified is found in the ether value displayed in response to the `ifconfig tr0` command issued at a root level of authority. (Tr0 is the name of the machine's token-ring interface.) If you do not have the necessary level of authority, your Linux system administrator can tell you the value.

9 Port name

This is a meaningful symbolic name that is used to associate the definitions with a network interface (in this case, a Token-Ring adapter). A separate Port must be defined for each physical device attached to the workstation.

10 Full path to executable

This is the path and name of the script file that invokes the WebSphere MQ program to run.

11 Local queue manager

This is the name of the queue manager on your local system.

10 Link station name

This is a meaningful symbolic name by which the connection to a peer or host node is known. It defines a logical path to the remote system. Its name is used only inside Communications Server for Linux and is specified by you. The connection must be associated with an existing Link and owned by one local node. You must define one connection for each partner or host system.

18 CPI-C symbolic destination name

This is a name given to the definition of a partner node. You choose the name. It need be unique only on this machine. Later you can use this name in the WebSphere MQ sender channel definition.

20 Node ID to receive

This is the unique ID of the partner workstation with which you will be communicating. On other platforms this is often referred to as the *Exchange ID* or *XID*. For a connection to a host system any values except 000 FFFFF and FFF FFFFF may be specified. Your network administrator will assign this ID for you.

Establishing a session using Communications Server for Linux

The following information guides you through the tasks you must perform to create the SNA infrastructure that WebSphere MQ requires. This example creates the definitions for a partner node and LU on HP-UX.

Note: The information in this section applies only to WebSphere MQ for Linux (x86 platform). It does not apply to WebSphere MQ for Linux (x86-64 platform), WebSphere MQ for Linux (zSeries s390x platform), or WebSphere MQ for Linux (POWER platform).

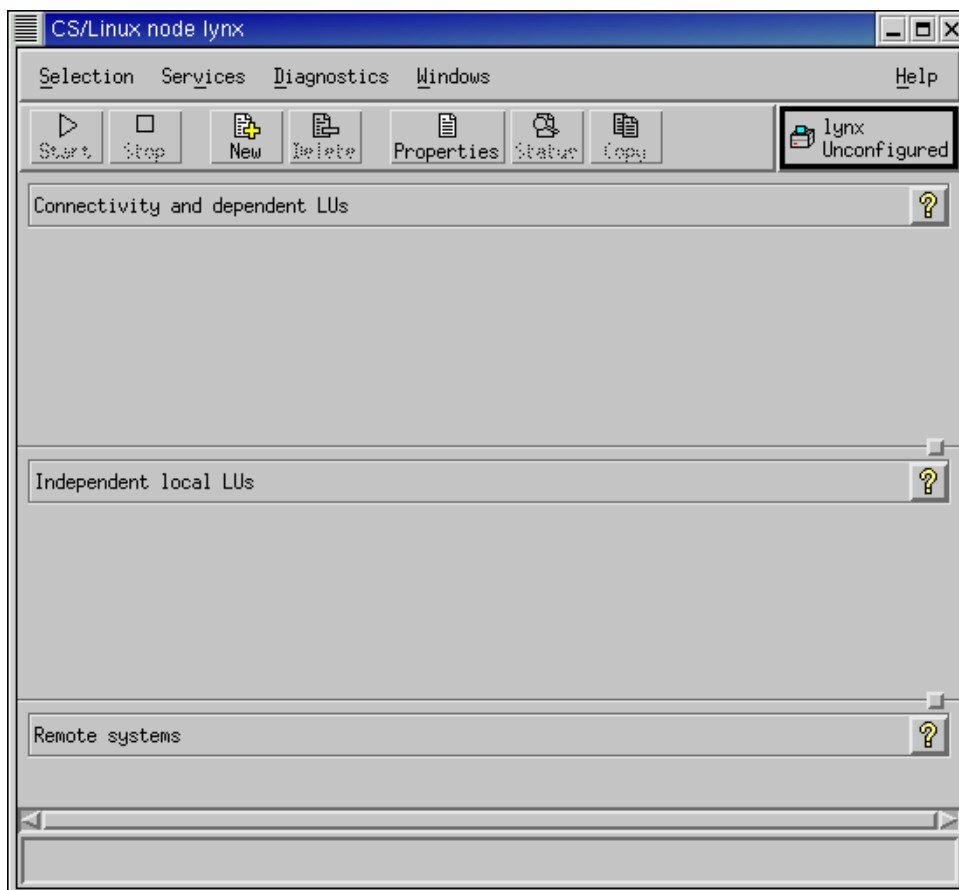
Use **sna start** followed by **xsnaadmin** to access the Communications Server for Linux main window. You need root authority to use **xsnaadmin**.

Communications Server for Linux configuration

Communications Server for Linux configuration involves the following steps:

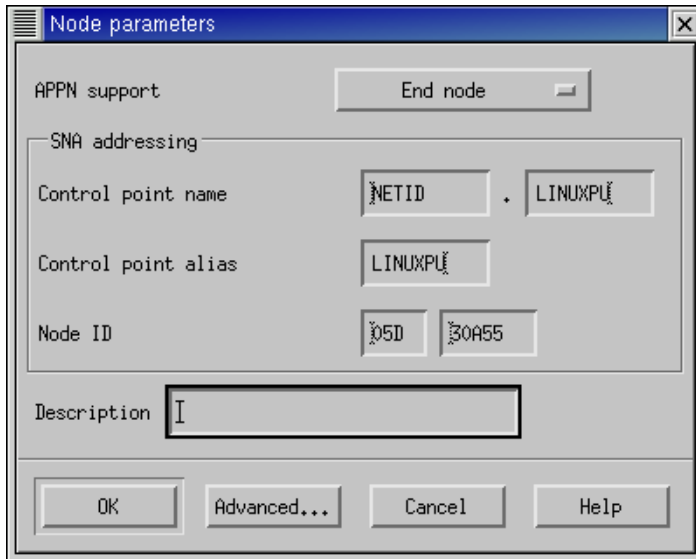
1. Defining a local node
2. Adding a Token-Ring port
3. Defining a local LU

The Communications Server for Linux main window, from which you start, is shown here:



Defining a local node:

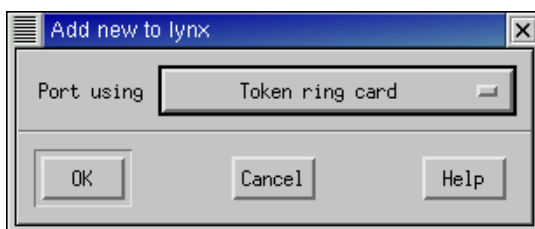
1. From the Communications Server for Linux main menu, click **Services** —> **Configure node parameters**. The “Node parameters” window opens.



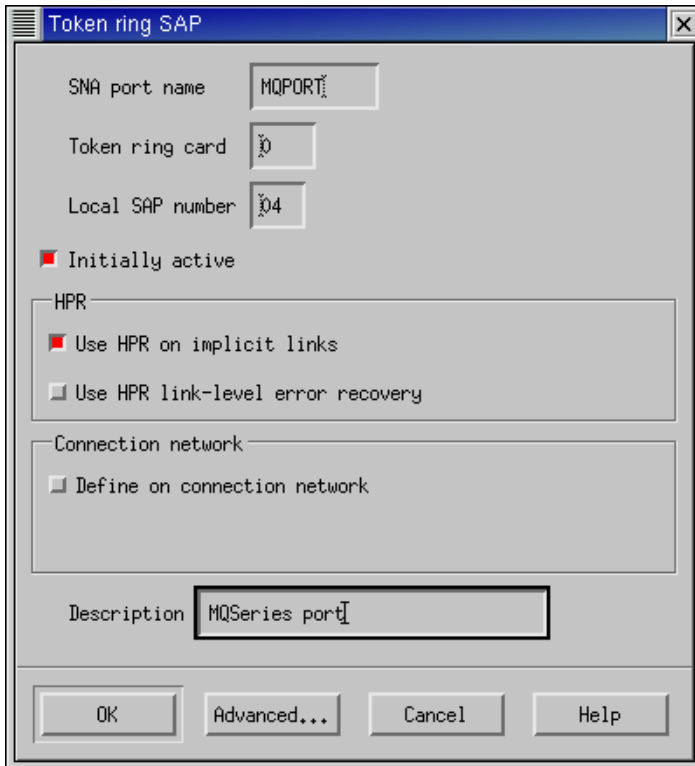
2. Set **APPN support** to **End node**.
3. In the **Control point name** fields, specify the network qualified name of the control point at the local node. In the first field, type the network name (4). In the second field, type the control point name (2).
4. In the **Control point alias** field, type the control point name (2) again.
5. In the **Node ID** fields, type the two components of the node ID (3).
6. Click **OK**. A message is displayed informing you that a default LU has been defined automatically for the local node.
7. Click **OK**.

Adding a Token-Ring port:

1. From the Communications Server for Linux main menu, click **Services** → **Connectivity** → **New port**. The following window opens.



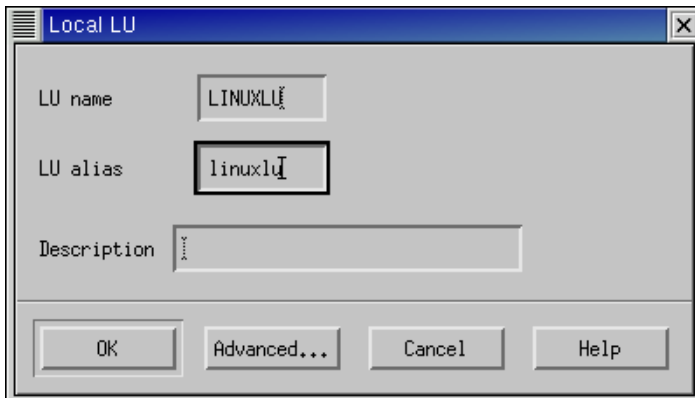
2. Set **Port using** to **Token Ring Card**, or to the type of network adapter card you are using.
3. Click **OK**. The “Token-Ring SAP” window opens.



4. In the **SNA port name** field, type the port name (9).
5. Click **OK**.

Defining a local LU:

1. From the Communications Server for Linux main menu, click **Services** → **APPC** → **New independent local LU**. The “Local LU” window opens.



2. In the **LU name** field, type the name of the local LU (5).
3. Click the **LU alias** field. Communications Server for Linux suggests an LU alias that is the same as the name of the local LU (5).
4. Click **OK**.

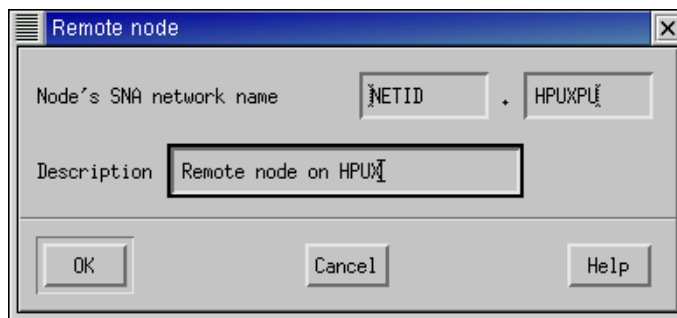
APPC configuration

APPC configuration involves the following steps:

1. Defining a remote node
2. Defining a partner LU
3. Defining a link station
4. Defining a mode
5. Adding CPI-C information
6. Adding a TP definition

Defining a remote node:

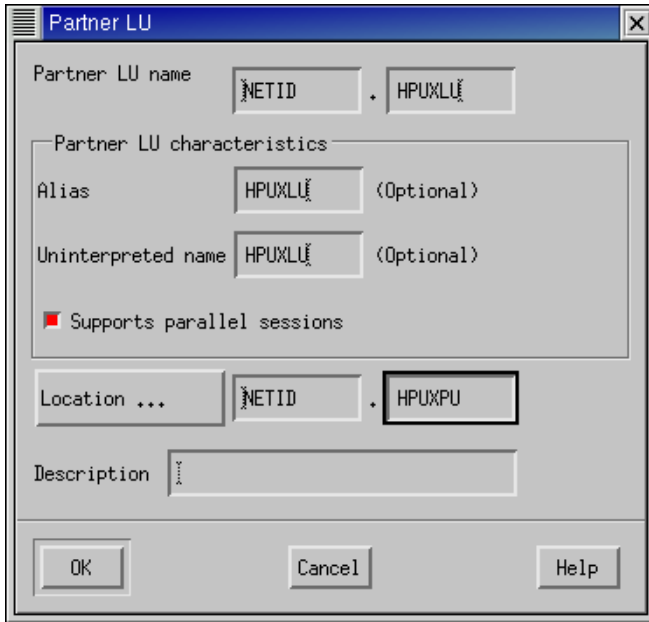
1. From the Communications Server for Linux main menu, click **Services** → **APPC** → **New remote node**. The “Remote node” window opens.



2. In the **Node's SNA network name** fields, specify the network qualified name of the control point at the remote node. Communications Server for Linux completes the first field for you by setting it to the network name (4 and 13) you entered earlier. In the second field, type the control point name (14).
3. Click **OK**. A message is displayed informing you that a default LU has been defined automatically for the remote node.
4. Click **OK**.

Defining a partner LU:

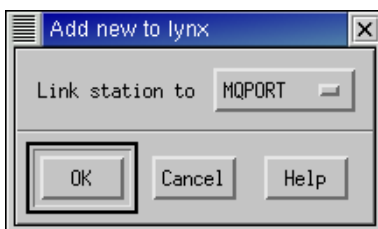
1. From the Communications Server for Linux main menu, click **Services** → **APPC** → **New partner LUs** → **Partner LU on remote node**. The “Partner LU” window opens.



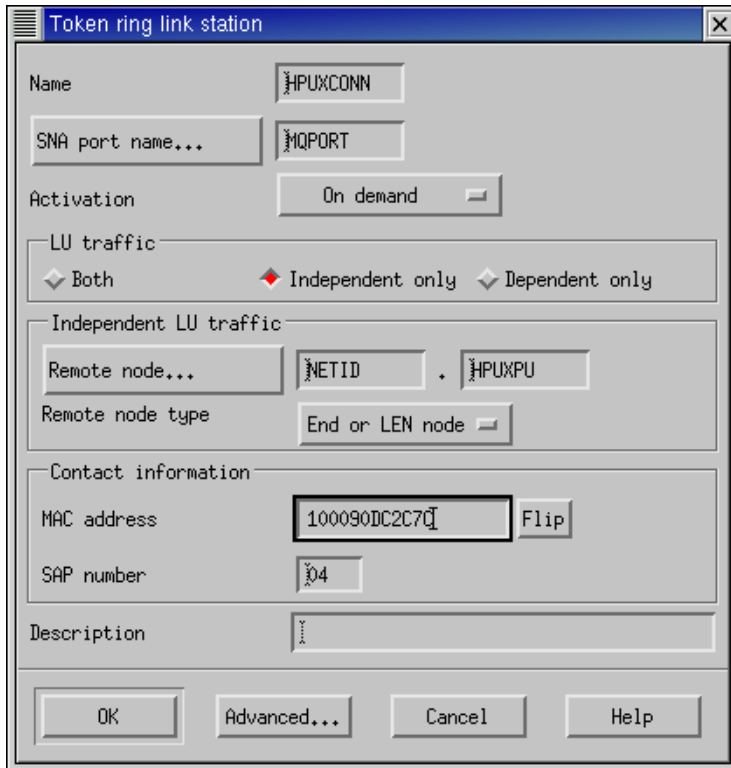
2. In the **Partner LU name** fields, specify the network qualified name of the partner LU at the remote node. Communications Server for Linux completes the first field for you by setting it to the network name (4 and 13) you entered earlier. In the second field, type the name of the partner LU (15).
3. In the **Alias** and **Uninterpreted name** fields, type the name of the partner LU (15) again.
4. Click **Location**, select the network qualified name of the control point at the remote node (13.14) from the list that is displayed, and click **OK**.
5. Click **OK**.

Defining a link station:

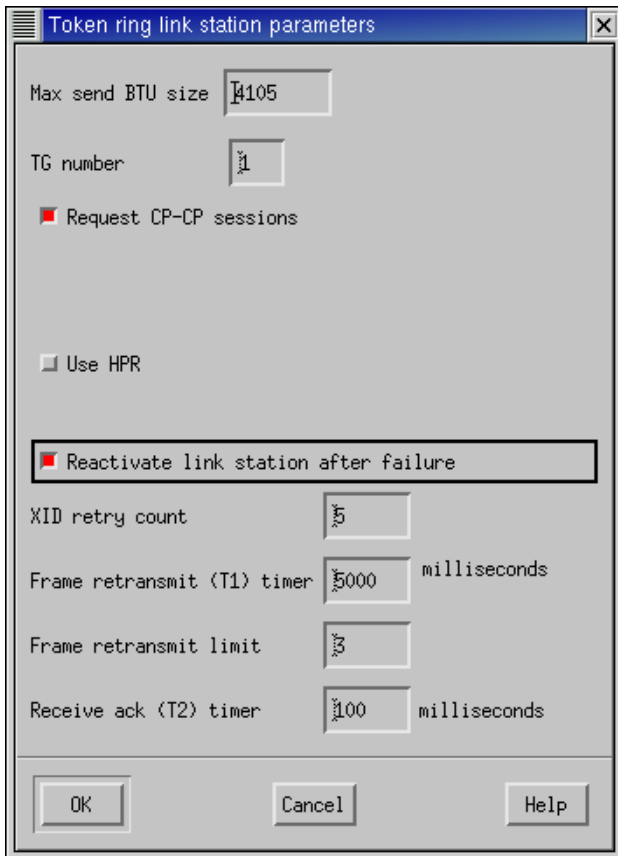
1. From the Communications Server for Linux main menu, click **Services** → **Connectivity** → **New link station**. The following window opens.



2. Set **Link station to** to **MQPORT**.
3. Click **OK**. The “Token ring link station” window opens.



4. In the **Name** field, type the name of the link station (12).
5. Set **Activation** to **On demand**.
6. Select the **Independent only** check box.
7. Click **Remote node**, select the network qualified name of the control point at the remote node (13.14) from the list that is displayed, and click **OK**.
8. Set **Remote node type** to **End or LEN node**.
9. In the **MAC address** field, type the MAC address (19) of the network adapter card at the remote node.
10. Click **Advanced**. The “Token ring link station parameters” window opens.

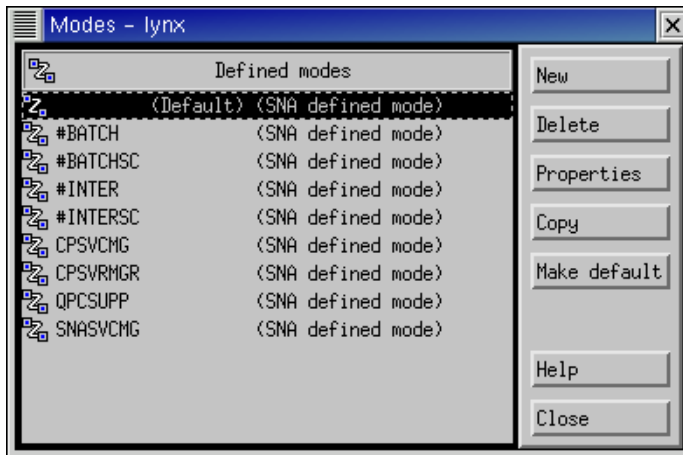


11. Select the **Request CP-CP sessions** check box.
12. Select the **Reactivate link station after failure** check box.
13. Click **OK** to exit the “Token ring link station parameters” window.
14. Click **OK** to exit the “Token ring link station” window.

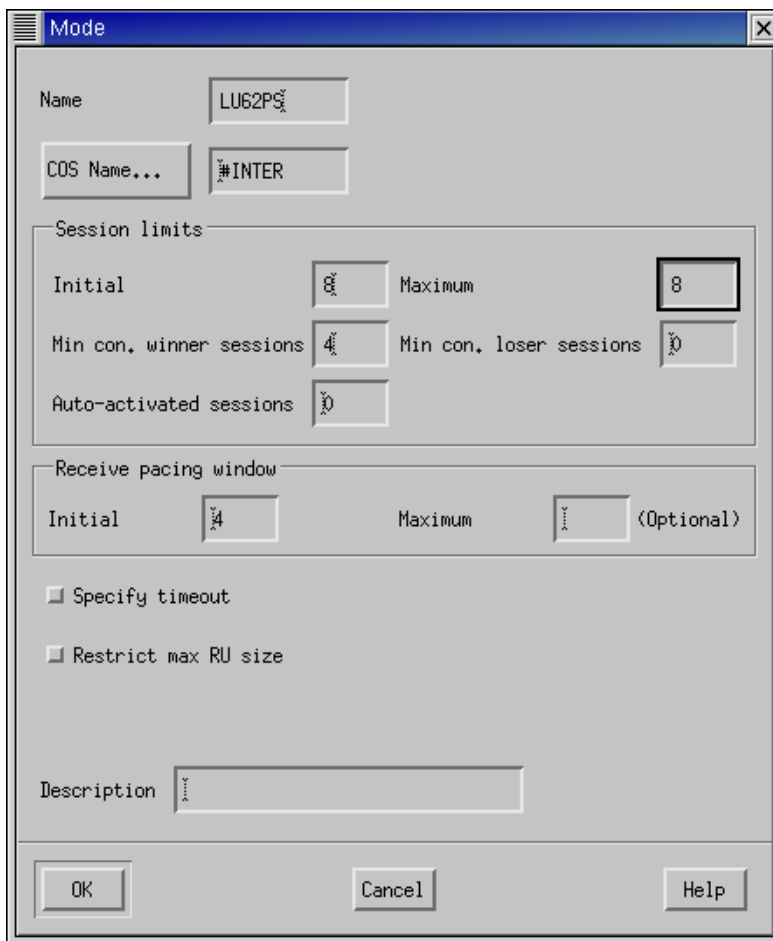
Defining a mode:

This purpose of the section is to explain how to define a new mode with the name LU62PS. The example continues subsequently, however, by using the mode #INTER (17), which is a standard mode supplied by Communications Server for Linux.

1. From the Communications Server for Linux main menu, click **Services** → **APPC** → **Modes**. The “Modes” window opens.



2. Click **New**. The “Mode” window opens.

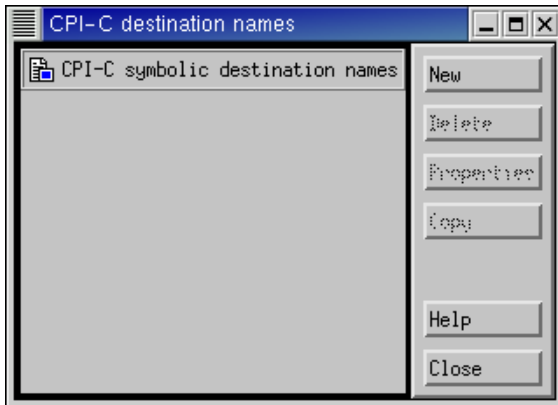


3. In the **Name** field, type the name of the new mode, LU62PS.
4. Click **COS Name**, select the class of service **#INTER** from the list that is displayed, and click **OK**.
5. For the **Session limits**:
 - Type 8 in the **Initial** field.
 - Type 8 in the **Maximum** field.

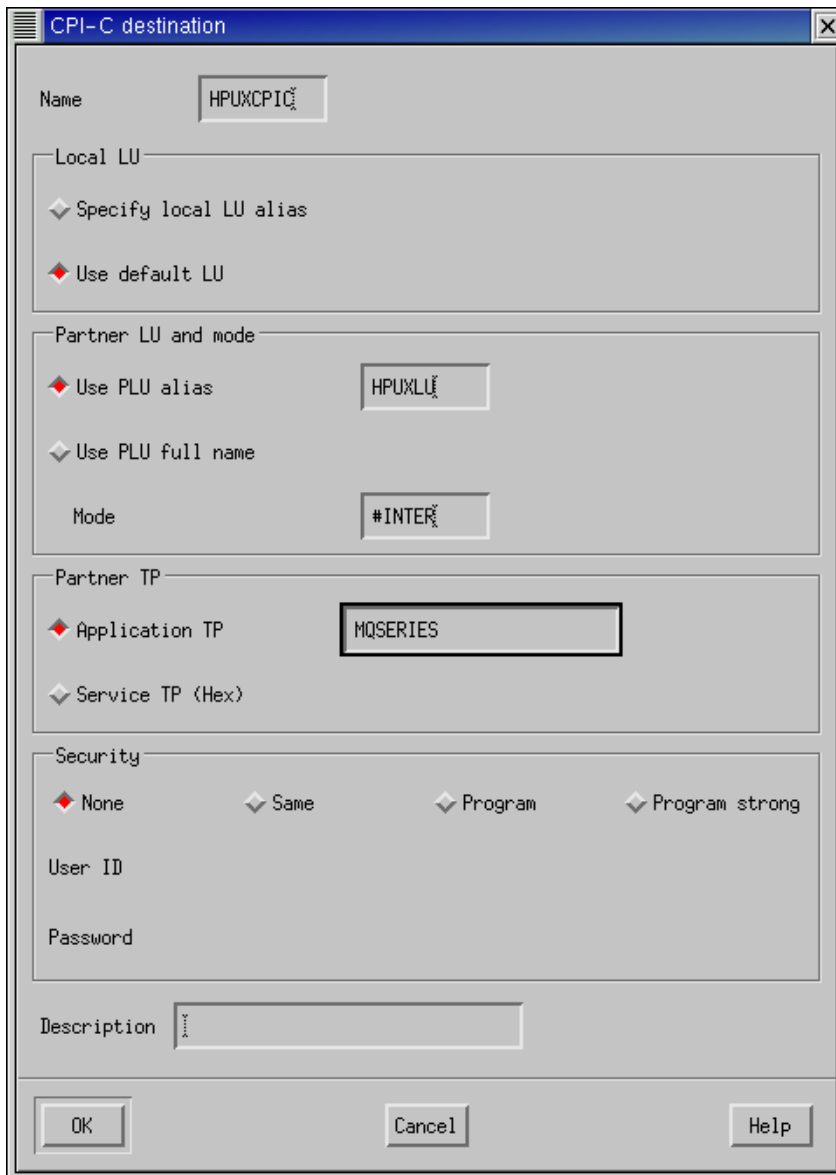
- Type 4 in the **Min con. winner sessions** field.
6. Click **OK** to exit the “Mode” window.
 7. Click **Close** to exit the “Modes” window.

Adding CPI-C information:

1. From the Communications Server for Linux main menu, click **Services** —> **APPC** —> **CPI-C**. The “CPI-C destination names” window opens.



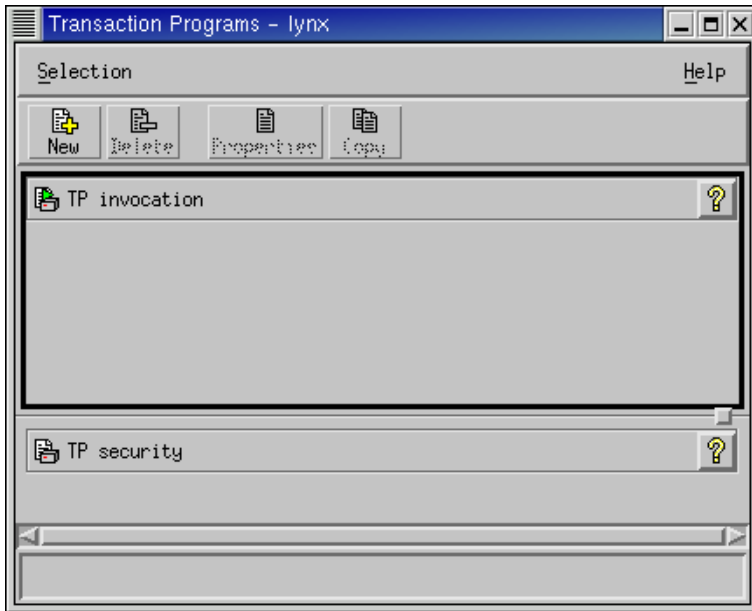
2. Click **New**. The “CPI-C destination” window opens.



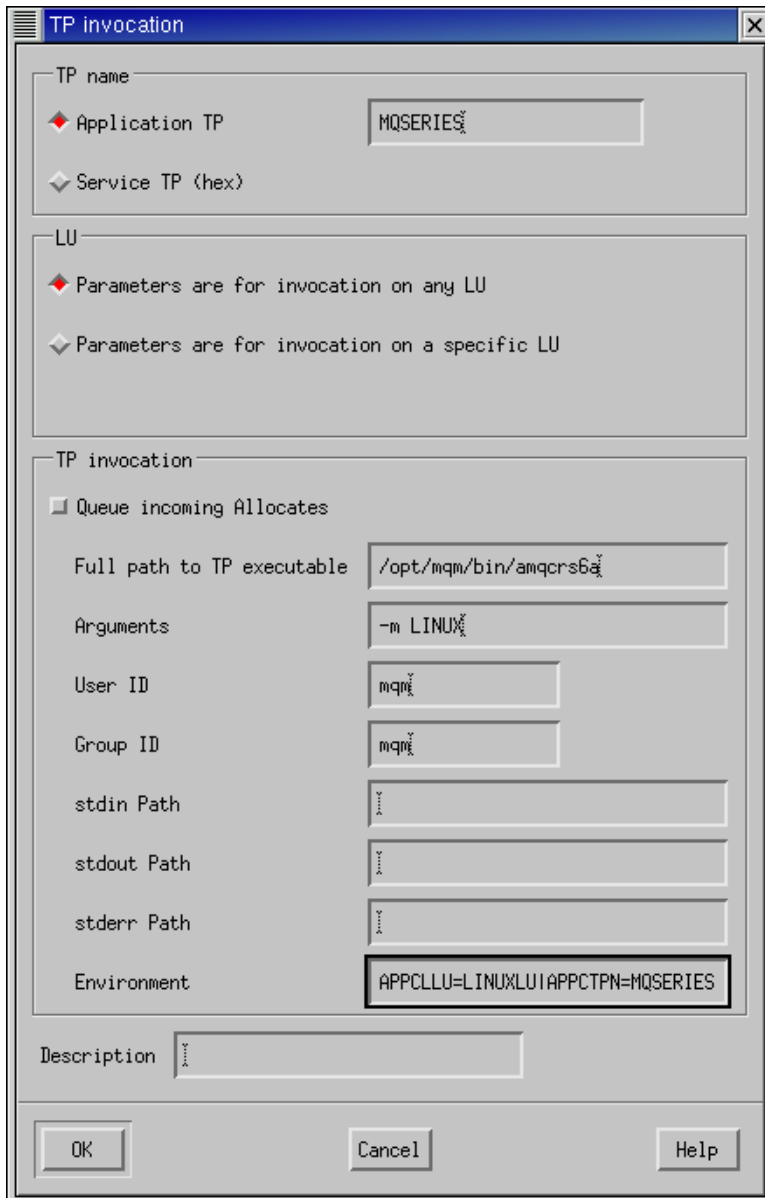
3. In the **Name** field, type the CPI-C symbolic destination name (18).
4. Select the **Use PLU alias** check box, and type the name of the partner LU (15), which you specified earlier as the partner LU alias.
5. In the **Mode** field, type the mode name (17).
6. Select the **Application TP** check box, and type the TP name (16).
7. Click **OK** to exit the “CPI-C destination” window.
8. Click **Close** to exit the “CPI-C destination names” window.

Adding a TP definition:

1. From the Communications Server for Linux main menu, click **Services** → **APPC** → **Transaction programs**. The “Transaction Programs” window opens.



2. Click **New**. The “TP invocation” window opens.



3. Select the **Application TP** check box, and type the TP name (7).
4. Clear the **Queue incoming Allocates** check box.
5. In the **Full path to TP executable** field, type the full path to the executable program (10).
6. In the **Arguments** field, type `-m local queue manager` (11).
7. In the **User ID** and **Group ID** fields, type `mqm`.
8. In the **Environment** field, type `APPCLU=local LU name (5) and APPCTPN=TP name (7) separated by the pipe character`.
9. Click **OK** to exit the “TP invocation” window.
10. Click **Selection** → **Close TP window** to exit the “Transaction Programs” window.

Communications Server for Linux operation

The Communications Server for Linux control daemon is started with the **sna start** command. Depending on the options selected at installation, it may already be running.

The **xsnaadmin** utility controls Communications Server for Linux resources.

Logging and tracing are controlled from here. Log and trace files can be found in the `/var/opt/sna` directory. The logging files `sna.aud` and `sna.err` can be read using a standard editor such as `vi`.

In order to read the `sna1.trc` trace file, you must first format it by running the command:

```
snatrcfmt -f sna1.trc -o sna1
```

This produces an `sna1.dmp` file, which can be read using a normal editor. The `sna2.trc` trace file can be processed in the same way.

It is possible to edit the configuration file, but this is not a recommended method of configuring Communications Server for Linux.

The `APPCLLU` environment variable must be set before starting a sender channel from the Linux system. The command can be either entered interactively or added to the logon profile. Depending on the level of Bourne shell or Korn shell program being used, the command is:

```
export APPCLLU=Local LU name 5 newer level
```

or

```
APPCLLU=Local LU name 5 older level  
export
```

What next?

The connection is now established. You are ready to complete the configuration. Go to “WebSphere MQ for Linux configuration” on page 234.

Establishing a TCP connection

Some Linux distributions now use the extended inet daemon (XINETD) instead of the inet daemon (INETD). The following instructions tell you how to establish a TCP connection using either the inet daemon or the extended inet daemon.

Using the inet daemon (INETD)

To establish a TCP connection, follow these steps.

1. Edit the file `/etc/services`. If you do not have the following line in the file, add it as shown:

```
MQSeries 1414/tcp # MQSeries channel listener
```

Note: To edit this file, you must be logged in as a superuser or root.

2. Edit the file `/etc/inetd.conf`. If you do not have the following line in that file, add it as shown:

```
MQSeries stream tcp nowait mqm /opt/mqm/bin/amqcrsta amqcrsta  
[-m queue.manager.name]
```

3. Find the process ID of the inetd with the command:

```
ps -ef | grep inetd
```

4. Run the command:

```
kill -1 inetd processid
```

If you have more than one queue manager on your system, and therefore require more than one service, you must add a line for each additional queue manager to both `/etc/services` and `inetd.conf`.

For example:

```
MQSeries1    1414/tcp
MQSeries2    1822/tcp
MQSeries1 stream tcp nowait mqm /mqmtop/bin/amqcrsta amqcrsta -m QM1
MQSeries2 stream tcp nowait mqm /mqmtop/bin/amqcrsta amqcrsta -m QM2
```

This avoids error messages being generated if there is a limitation on the number of outstanding connection requests queued at a single TCP port. For information about the number of outstanding connection requests, see “Using the TCP listener backlog option” on page 152.

The `inetd` process on Linux can limit the rate of inbound connections on a TCP port. The default is 40 connections in a 60 second interval. If you need a higher rate, specify a new limit on the number of inbound connections in a 60 second interval by appending a period (.) followed by the new limit to the `nowait` parameter of the appropriate service in `inetd.conf`. For example, for a limit of 500 connections in a 60 second interval use:

```
MQSeries stream tcp nowait.500 mqm /mqmtop/bin/amqcrsta amqcrsta -m QM1
```

Using the extended inet daemon (XINETD)

The following instructions describe how the extended inet daemon is implemented on Red Hat Linux. If you are using a different Linux distribution, you might have to adapt these instructions.

To establish a TCP connection, follow these steps.

1. Edit the file `/etc/services`. If you do not have the following line in the file, add it as shown:

```
MQSeries      1414/tcp      # MQSeries channel listener
```

Note: To edit this file, you must be logged in as a superuser or root.

2. Create a file called `MQSeries` in the XINETD configuration directory, `/etc/xinetd.d`. Add the following stanza to the file:

```
# WebSphere MQ service for XINETD
service MQSeries
{
    disable          = no
    flags            = REUSE
    socket_type      = stream
    wait             = no
    user             = mqm
    server           = /opt/mqm/bin/amqcrsta
    server_args      = -m queue.manager.name
    log_on_failure += USERID
}
```

3. Restart the extended inet daemon by issuing the following command:
`/etc/rc.d/init.d/xinetd restart`

If you have more than one queue manager on your system, and therefore require more than one service, you must add a line to `/etc/services` for each additional queue manager. You can create a file in the `/etc/xinetd.d` directory for each service, or you can add additional stanzas to the `MQSeries` file you created previously.

The `xinetd` process on Linux can limit the rate of inbound connections on a TCP port. The default is 50 connections in a 10 second interval. If you need a higher rate, specify a new limit on the rate of inbound connections by specifying the 'cps' attribute in the `xinetd` configuration file. For example, for a limit of 500 connections in a 60 second interval use:

```
cps = 500 60
```

What next?

The TCP/IP connection is now established. You are ready to complete the configuration. Go to "WebSphere MQ for Linux configuration."

WebSphere MQ for Linux configuration

Before beginning the installation process ensure that you have first created the `mqm` user ID and the `mqm` group, and set the password.

Start any channel using the command:

```
runmqchl -c channel.name
```

Note:

1. Sample programs are installed in `/opt/mqm/samp`.
2. Error logs are stored in `/var/mqm/qmgrs/qmgrname/errors`.
3. When you are using the command interpreter `runmqsc` to enter administration commands, a `+` at the end of a line indicates that the next line is a continuation. Ensure that there is a space between the last parameter and the continuation character.

Basic configuration

1. Create the queue manager from the UNIX prompt using the command:

```
crtmqm -u dlqname -q linux
```

where:

linux Is the name of the queue manager

`-q` Indicates that this is to become the default queue manager

`-u dlqname`

Specifies the name of the dead letter queue

This command creates a queue manager and a set of default objects.

2. Start the queue manager from the UNIX prompt using the command:

```
strmqm linux
```

where *linux* is the name given to the queue manager when it was created.

Channel configuration

The following section details the configuration to be performed on the Linux queue manager to implement the channel described in Figure 32 on page 101.

The MQSC command to create each object is shown. Either start **runmqsc** from a UNIX prompt and enter each command in turn, or build the commands into a command file.

Examples are given for connecting WebSphere MQ for Linux and WebSphere MQ for HP-UX. If you wish to connect to WebSphere MQ on another platform use the appropriate set of values from the table in place of those for HP-UX.

Note: The words in **bold** are user-specified and reflect the names of WebSphere MQ objects used throughout these examples. If you change the names used here, ensure that you also change the other references made to these objects throughout this book. All others are keywords and should be entered as shown.

Table 24. Configuration worksheet for WebSphere MQ for Linux

ID	Parameter Name	Reference	Example Used	User Value
<i>Definition for local node</i>				
A	Queue Manager Name		LINUX	
B	Local queue name		LINUX.LOCALQ	
<i>Connection to WebSphere MQ for Windows</i>				
The values in this section of the table must match those used in Table 14 on page 146, as indicated.				
C	Remote queue manager name	A	WINNT	
D	Remote queue name		WINNT.REMOTEQ	
E	Queue name at remote system	B	WINNT.LOCALQ	
F	Transmission queue name		WINNT	
G	Sender (SNA) channel name		LINUX.WINNT.SNA	
H	Sender (TCP/IP) channel name		LINUX.WINNT.TCP	
I	Receiver (SNA) channel name	G	WINNT.LINUX.SNA	
J	Receiver (TCP) channel name	H	WINNT.LINUX.TCP	
<i>Connection to WebSphere MQ for AIX</i>				
The values in this section of the table must match those used in Table 18 on page 168, as indicated.				
C	Remote queue manager name	A	AIX	
D	Remote queue name		AIX.REMOTEQ	
E	Queue name at remote system	B	AIX.LOCALQ	
F	Transmission queue name		AIX	
G	Sender (SNA) channel name		LINUX.AIX.SNA	
H	Sender (TCP) channel name		LINUX.AIX.TCP	
I	Receiver (SNA) channel name	G	AIX.LINUX.SNA	
J	Receiver (TCP) channel name	H	AIX.LINUX.TCP	
<i>Connection to MQSeries for Compaq Tru64 UNIX</i>				
The values in this section of the table must match those used in your Compaq Tru64 UNIX system.				
C	Remote queue manager name	A	DECUX	
D	Remote queue name		DECUX.REMOTEQ	
E	Queue name at remote system	B	DECUX.LOCALQ	
F	Transmission queue name		DECUX	
H	Sender (TCP) channel name		DECUX.LINUX.TCP	

Table 24. Configuration worksheet for WebSphere MQ for Linux (continued)

ID	Parameter Name	Reference	Example Used	User Value
J	Receiver (TCP) channel name	H	LINUX.DECUX.TCP	
Connection to WebSphere MQ for HP-UX				
The values in this section of the table must match those used in Table 20 on page 191, as indicated.				
C	Remote queue manager name	A	HPUX	
D	Remote queue name		HPUX.REMOTEQ	
E	Queue name at remote system	B	HPUX.LOCALQ	
F	Transmission queue name		HPUX	
G	Sender (SNA) channel name		LINUX.HPUX.SNA	
H	Sender (TCP) channel name		LINUX.HPUX.TCP	
I	Receiver (SNA) channel name	G	HPUX.LINUX.SNA	
J	Receiver (TCP/IP) channel name	H	HPUX.LINUX.TCP	
Connection to WebSphere MQ for Solaris				
The values in this section of the table must match those used in Table 22 on page 212, as indicated.				
C	Remote queue manager name	A	SOLARIS	
D	Remote queue name		SOLARIS.REMOTEQ	
E	Queue name at remote system	B	SOLARIS.LOCALQ	
F	Transmission queue name		GIS	
G	Sender (SNA) channel name		LINUX.SOLARIS.SNA	
H	Sender (TCP/IP) channel name		LINUX.SOLARIS.TCP	
I	Receiver (SNA) channel name	G	SOLARIS.LINUX.SNA	
J	Receiver (TCP/IP) channel name	H	SOLARIS.LINUX.TCP	
Connection to WebSphere MQ for i5/OS				
The values in this section of the table must match those used in Table 36 on page 364, as indicated.				
C	Remote queue manager name	A	AS400	
D	Remote queue name		AS400.REMOTEQ	
E	Queue name at remote system	B	AS400.LOCALQ	
F	Transmission queue name		AS400	
G	Sender (SNA) channel name		LINUX.AS400.SNA	
H	Sender (TCP) channel name		LINUX.AS400.TCP	
I	Receiver (SNA) channel name	G	AS400.LINUX.SNA	
J	Receiver (TCP) channel name	H	AS400.LINUX.TCP	
Connection to WebSphere MQ for z/OS				
The values in this section of the table must match those used in Table 28 on page 272, as indicated.				
C	Remote queue manager name	A	MVS	
D	Remote queue name		MVS.REMOTEQ	
E	Queue name at remote system	B	MVS.LOCALQ	
F	Transmission queue name		MVS	
G	Sender (SNA) channel name		LINUX.MVS.SNA	
H	Sender (TCP) channel name		LINUX.MVS.TCP	

Table 24. Configuration worksheet for WebSphere MQ for Linux (continued)

ID	Parameter Name	Reference	Example Used	User Value
I	Receiver (SNA) channel name	G	MVS.LINUX.SNA	
J	Receiver (TCP) channel name	H	MVS.LINUX.TCP	
Connection to MQSeries for VSE/ESA (WebSphere MQ for Linux (x86 platform) only)				
The values in this section of the table must match those used in your VSE/ESA system.				
C	Remote queue manager name	A	VSE	
D	Remote queue name		VSE.REMOTEQ	
E	Queue name at remote system	B	VSE.LOCALQ	
F	Transmission queue name		VSE	
G	Sender channel name		LINUX.VSE.SNA	
I	Receiver channel name	G	VSE.LINUX.SNA	

WebSphere MQ for Linux (x86 platform) sender-channel definitions using SNA:

```
def ql (HPUX) +                                     F
    usage(xmitq) +
    replace

def qr (HPUX.REMOTEQ) +                             D
    rname(HPUX.LOCALQ) +                             E
    rqmname(HPUX) +                                  C
    xmitq(HPUX) +                                     F
    replace

def chl (LINUX.HPUX.SNA) chltype(sdr) +             G
    trptype(lu62) +
    conname('HPUXCPIC') +                            14
    xmitq(HPUX) +                                     F
    replace
```

WebSphere MQ for Linux (x86 platform) receiver-channel definitions using SNA:

```
def ql (LINUX.LOCALQ) replace                       B

def chl (HPUX.LINUX.SNA) chltype(rcvr) +           I
    trptype(lu62) +
    replace
```

WebSphere MQ for Linux sender-channel definitions using TCP:

```
def ql (HPUX) +                                     F
    usage(xmitq) +
    replace

def qr (HPUX.REMOTEQ) +                             D
    rname(HPUX.LOCALQ) +                             E
    rqmname(HPUX) +                                  C
    xmitq(HPUX) +                                     F
    replace

def chl (LINUX.HPUX.TCP) chltype(sdr) +           H
    trptype(tcp) +
    conname(remote_tcpip_hostname) +
    xmitq(HPUX) +                                     F
    replace
```

WebSphere MQ for Linux receiver-channel definitions using TCP/IP:

```
def ql (LINUX.LOCALQ) replace B
def chl (HPUX.LINUX.TCP) chltype(rcvr) + J
  trptype(tcp) +
  replace
```

Message channel planning example for distributed platforms

This chapter provides a detailed example of how to connect two queue managers together so that messages can be sent between them. The example illustrates the preparations needed to allow an application using queue manager QM1 to put messages on a queue at queue manager QM2. An application running on QM2 can retrieve these messages, and send responses to a reply queue on QM1.

The example illustrates the use of TCP/IP connections. The example assumes that channels are to be triggered to start when the first message arrives on the transmission queue they are servicing. You must start the channel initiator in order for triggering to work.

This example uses SYSTEM.CHANNEL.INITQ as the initiation queue. This queue is already defined by WebSphere MQ. You can use a different initiation queue, but you will have to define it yourself and specify the name of the queue when you start the channel initiator.

What the example shows

The example shows the WebSphere MQ commands (MQSC) that you can use.

In all the examples, the MQSC commands are shown as they would appear in a file of commands, and as they would be typed at the command line. The two methods look identical, but, to issue a command at the command line, you must first type `runmqsc`, for the default queue manager, or `runmqsc qmname` where *qmname* is the name of the required queue manager. Then type any number of commands, as shown in the examples.

An alternative method is to create a file containing these commands. Any errors in the commands are then easy to correct. If you called your file `mqsc.in` then to run it on queue manager QMNAME use:

```
runmqsc QMNAME < mqsc.in > mqsc.out
```

You could verify the commands in your file before running it using:

```
runmqsc -v QMNAME < mqsc.in > mqsc.out
```

For portability, you should restrict the line length of your commands to 72 characters. Use a concatenation character to continue over more than one line. On Windows use Ctrl-z to end the input at the command line. On UNIX systems use Ctrl-d. Alternatively, use the **end** command.

Figure 35 on page 239 shows the example scenario.

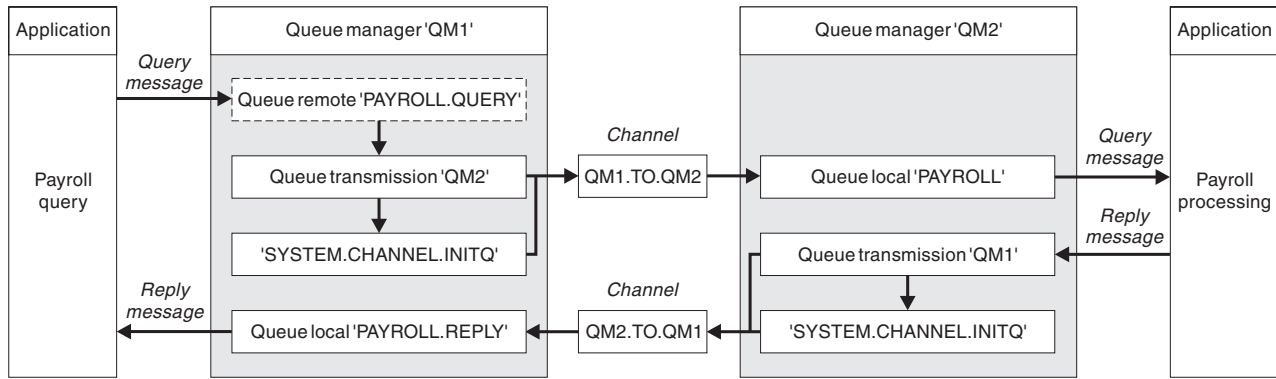


Figure 35. The message channel example for Windows, and UNIX systems

The example involves a payroll query application connected to queue manager QM1 that sends payroll query messages to a payroll processing application running on queue manager QM2. The payroll query application needs the replies to its queries sent back to QM1. The payroll query messages are sent from QM1 to QM2 on a sender-receiver channel called QM1.TO.QM2, and the reply messages are sent back from QM2 to QM1 on another sender-receiver channel called QM2.TO.QM1. Both of these channels are triggered to start as soon as they have a message to send to the other queue manager.

The payroll query application puts a query message to the remote queue "PAYROLL.QUERY" defined on QM1. This remote queue definition resolves to the local queue "PAYROLL" on QM2. In addition, the payroll query application specifies that the reply to the query is sent to the local queue "PAYROLL.REPLY" on QM1. The payroll processing application gets messages from the local queue "PAYROLL" on QM2, and sends the replies to wherever they are required; in this case, local queue "PAYROLL.REPLY" on QM1.

In the example definitions for TCP/IP, QM1 has a host address of 9.20.9.31 and is listening on port 1411, and QM2 has a host address of 9.20.9.32 and is listening on port 1412. The example assumes that these are already defined on your system and available for use.

The object definitions that need to be created on QM1 are:

- Remote queue definition, PAYROLL.QUERY
- Transmission queue definition, QM2 (default=remote queue manager name)
- Sender channel definition, QM1.TO.QM2
- Receiver channel definition, QM2.TO.QM1
- Reply-to queue definition, PAYROLL.REPLY

The object definitions that need to be created on QM2 are:

- Local queue definition, PAYROLL
- Transmission queue definition, QM1 (default=remote queue manager name)
- Sender channel definition, QM2.TO.QM1
- Receiver channel definition, QM1.TO.QM2

The connection details are supplied in the CONNAME attribute of the sender channel definitions.

You can see a diagram of the arrangement in Figure 35.

Queue manager QM1 example

The following object definitions allow applications connected to queue manager QM1 to send request messages to a queue called PAYROLL on QM2, and to receive replies on a queue called PAYROLL.REPLY on QM1.

All the object definitions have been provided with the DESCR and REPLACE attributes. The other attributes supplied are the minimum required to make the example work. The attributes that are not supplied take the default values for queue manager QM1.

Run the following commands on queue manager QM1.

Remote queue definition

```
DEFINE QREMOTE(PAYROLL.QUERY) DESCR('Remote queue for QM2') REPLACE +  
PUT(ENABLED) XMITQ(QM2) RNAME(PAYROLL) RQMNAME(QM2)
```

Note: The remote queue definition is not a physical queue, but a means of directing messages to the transmission queue, QM2, so that they can be sent to queue manager QM2.

Transmission queue definition

```
DEFINE QLOCAL(QM2) DESCR('Transmission queue to QM2') REPLACE +  
USAGE(XMITQ) PUT(ENABLED) GET(ENABLED) TRIGGER TRIGTYPE(FIRST) +  
INITQ(SYSTEM.CHANNEL.INITQ) PROCESS(QM1.TO.QM2.PROCESS)
```

When the first message is put on this transmission queue, a trigger message is sent to the initiation queue, SYSTEM.CHANNEL.INITQ. The channel initiator gets the message from the initiation queue and starts the channel identified in the named process.

Sender channel definition

```
DEFINE CHANNEL(QM1.TO.QM2) CHLTYPE(SDR) TRPTYPE(TCP) +  
REPLACE DESCR('Sender channel to QM2') XMITQ(QM2) +  
CONNNAME('9.20.9.32(1412)')
```

Receiver channel definition

```
DEFINE CHANNEL(QM2.TO.QM1) CHLTYPE(RCVR) TRPTYPE(TCP) +  
REPLACE DESCR('Receiver channel from QM2')
```

Reply-to queue definition

```
DEFINE QLOCAL(PAYROLL.REPLY) REPLACE PUT(ENABLED) GET(ENABLED) +  
DESCR('Reply queue for replies to query messages sent to QM2')
```

The reply-to queue is defined as PUT(ENABLED). This ensures that reply messages can be put to the queue. If the replies cannot be put to the reply-to queue, they are sent to the dead-letter queue on QM1 or, if this queue is not available, remain on transmission queue QM1 on queue manager QM2. The queue has been defined as GET(ENABLED) to allow the reply messages to be retrieved.

Queue manager QM2 example

The following object definitions allow applications connected to queue manager QM2 to retrieve request messages from a local queue called PAYROLL, and to put replies to these request messages to a queue called PAYROLL.REPLY on queue manager QM1.

You do not need to provide a remote queue definition to enable the replies to be returned to QM1. The message descriptor of the message retrieved from local queue PAYROLL contains both the reply-to queue and the reply-to queue manager names. Therefore, as long as QM2 can resolve the reply-to queue manager name to that of a transmission queue on queue manager QM2, the reply message can be sent. In this example, the reply-to queue manager name is QM1 and so queue manager QM2 simply requires a transmission queue of the same name.

All the object definitions have been provided with the DESCR and REPLACE attributes and are the minimum required to make the example work. The attributes that are not supplied take the default values for queue manager QM2.

Run the following commands on queue manager QM2.

Local queue definition

```
DEFINE QLOCAL(PAYROLL) REPLACE PUT(ENABLED) GET(ENABLED) +  
DESCR('Local queue for QM1 payroll details')
```

This queue is defined as PUT(ENABLED) and GET(ENABLED) for the same reason as the reply-to queue definition on queue manager QM1.

Transmission queue definition

```
DEFINE QLOCAL(QM1) DESCR('Transmission queue to QM1') REPLACE +  
USAGE(XMITQ) PUT(ENABLED) GET(ENABLED) TRIGGER TRIGTYPE(FIRST) +  
INITQ(SYSTEM.CHANNEL.INITQ) PROCESS(QM2.TO.QM1.PROCESS)
```

When the first message is put on this transmission queue, a trigger message is sent to the initiation queue, SYSTEM.CHANNEL.INITQ. The channel initiator gets the message from the initiation queue and starts the channel identified in the named process.

Sender channel definition

```
DEFINE CHANNEL(QM2.TO.QM1) CHLTYPE(SDR) TRPTYPE(TCP) +  
REPLACE DESCR('Sender channel to QM1') XMITQ(QM1) +  
CONNNAME('9.20.9.31(1411)')
```

Receiver channel definition

```
DEFINE CHANNEL(QM1.TO.QM2) CHLTYPE(RCVR) TRPTYPE(TCP) +  
REPLACE DESCR('Receiver channel from QM1')
```

Running the example

Once these definitions have been created, you need to:

- Start the channel initiator on each queue manager.
- Start the listener for each queue manager.

For information about starting the channel initiator and listener, see “Setting up communication for Windows” on page 120 and “Setting up communication on UNIX systems” on page 150.

Expanding this example

This simple example could be expanded with:

- The use of LU 6.2 communications for interconnection with CICS systems, and transaction processing.
- Adding more queue, process, and channel definitions to allow other applications to send messages between the two queue managers.

- Adding user-exit programs on the channels to allow for link encryption, security checking, or additional message processing.
- Using queue-manager aliases and reply-to queue aliases to understand more about how these can be used in the organization of your queue manager network.

Chapter 4. DQM in WebSphere MQ for z/OS

Monitoring and controlling channels on z/OS

Use the DQM commands and panels to create, monitor, and control the channels to remote queue managers. Each z/OS queue manager has a DQM program (the *channel initiator*) for controlling interconnections to remote queue managers using native z/OS facilities.

The implementation of these panels and commands on z/OS is integrated into the operations and control panels and the MQSC commands. No differentiation is made in the organization of these two sets of panels and commands.

Commands may also be entered using Programmable Command Format (PCF) commands. See the WebSphere MQ Programmable Command Formats and Administration Interface book for information about using these commands.

The information in this chapter applies in all cases where the channel initiator is used for distributed queuing. It applies whether or not you are using queue-sharing groups, or intra-group queuing.

The DQM channel control function

The channel control function provides the administration and control of message channels between WebSphere MQ for z/OS and remote systems. See Figure 28 on page 52 for a conceptual picture.

The channel control function consists of panels, commands and programs, two synchronization queues, channel command queues, and the channel definitions. The following is a brief description of the components of the channel control function.

- The channel definitions are held as objects in page set zero or in DB2[®], like other WebSphere MQ objects in z/OS.
- You use the operations and control panels, MQSC commands, or PCF commands to:
 - Create, copy, display, alter, and delete channel definitions
 - Start and stop channel initiators and listeners
 - Start, stop, and ping channels, reset channel sequence numbers, and resolve in-doubt messages when links cannot be re-established
 - Display status information about channels
 - Display information about DQM

In particular, you can use the CSQINPX initialization input data set to issue your MQSC commands. This can be processed every time you start the channel initiator. See the WebSphere MQ for z/OS Concepts and Planning Guide for information about this.

- There are two queues (SYSTEM.CHANNEL.SYNCQ and SYSTEM.QSG.CHANNEL.SYNCQ) used for channel re-synchronization purposes. You should define these with INDXTYPE(MSGID) for performance reasons.

- The channel command queue (SYSTEM.CHANNEL.INITQ) is used to hold commands for channel initiators, channels, and listeners.
- The channel control function program runs in its own address space, separate from the queue manager, and comprises the channel initiator, listeners, MCAs, trigger monitor, and command handler.
- For queue-sharing groups and shared channels, see “Preparing WebSphere MQ for z/OS for DQM with queue-sharing groups” on page 280.
- For intra-group queuing, see “Intra-group queuing” on page 298

Using the panels and the commands

You can use either the MQSC commands, the PCF commands, or the operations and control panels to manage DQM. For information about the syntax of the MQSC commands, see the WebSphere MQ Script (MQSC) Command Reference book. For information about PCF commands, see WebSphere MQ Programmable Command Formats and Administration Interface.

Using the initial panel

For an introduction to invoking the operations and control panels, using the function keys, and getting help, see the WebSphere MQ for z/OS System Administration Guide.

Note: To use the operations and control panels, you must have the correct security authorization; see the WebSphere MQ for z/OS System Setup Guide for information. Figure 36 shows the panel that is displayed when you start a panel session. The text after the panel explains the actions you should perform in this panel.

```

IBM WebSphere MQ for z/OS - Main Menu

Complete fields. Then press Enter.

Action . . . . . 1      0. List with filter  4. Manage
                          1. List or Display  5. Perform
                          2. Define like    6. Start
                          3. Alter          7. Stop

Object type . . . . . CHANNEL      +
Name . . . . . *
Disposition . . . . . A  Q=Qmgr, C=Copy, P=Private, G=Group,
                          S=Shared, A=All

Connect name . . . . . MQ25 - local queue manager or group
Target queue manager . . . MQ25
                          - connected or remote queue manager for command input
Action queue manager . . . MQ25 - command scope in group
Response wait time . . . . 10    5 - 999 seconds

(C) Copyright IBM Corporation 1993,2005. All rights reserved.

Command ==>
F1=Help      F2=Split      F3=Exit      F4=Prompt      F9=SwapNext  F10=Messages
F12=Cancel

```

Figure 36. The operations and controls initial panel

From this panel you can:

- Select the action you want to perform by typing in the appropriate number in the **Action** field.

- Specify the object type that you want to work with. Press F4 for a list of object types if you are not sure what they are.
- Display a list of objects of the type specified. Type in an asterisk (*) in the **Name** field and press Enter to display a list of objects (of the type specified) that have already been defined on this subsystem. You can then select one or more objects to work with in sequence. Figure 37 shows a list of channels produced in this way.
- Specify the disposition in the queue-sharing group of the objects you want to work with in the **Disposition** field. The disposition determines where the object is kept and how the object behaves.
- Choose the local queue manager, or queue-sharing group to which you want to connect in the **Connect name** field. If you want the commands to be issued on a remote queue manager, choose either the **Target queue manager** field or the **Action queue manager** field, depending upon whether the remote queue manager is not or is a member of a queue-sharing group. If the remote queue manager is *not* a member of a queue-sharing group, choose the **Target queue manager** field. If the remote queue manager *is* a member of a queue-sharing group, choose the **Action queue manager** field.
- Choose the wait time for responses to be received in the **Response wait time** field.

```

List Channels - MQ25                               Row 1 of 8

Type action codes, then press Enter. Press F11 to display connection status.
1=Display  2=Define like  3=Alter  4=Manage  5=Perform
6=Start    7=Stop

  Name                Type          Disposition  Status
<> *                  CHANNEL      ALL          MQ25
- SYSTEM.DEF.CLNTCONN CLNTCONN    QMGR         MQ25
- SYSTEM.DEF.CLUSRCVR CLUSRCVR    QMGR         MQ25  INACTIVE
- SYSTEM.DEF.CLUSSDR  CLUSSDR    QMGR         MQ25  INACTIVE
- SYSTEM.DEF.RECEIVER RECEIVER    QMGR         MQ25  INACTIVE
- SYSTEM.DEF.REQUESTER REQUESTER   QMGR         MQ25  INACTIVE
- SYSTEM.DEF.SENDER   SENDER     QMGR         MQ25  INACTIVE
- SYSTEM.DEF.SERVER   SERVER     QMGR         MQ25  INACTIVE
- SYSTEM.DEF.SVRCONN  SVRCONN    QMGR         MQ25  INACTIVE
***** End of list *****

Command ==>> _____
F1=Help   F2=Split  F3=Exit   F4=Filter  F5=Refresh F7=Bkwd
F8=Fwd    F9=SwapNext F10=Messages F11=Status F12=Cancel

```

Figure 37. Listing channels

Managing your channels

Table 25 lists the tasks that you can perform to manage your channels, channel initiators, and listeners. It also gives the name of the relevant MQSC command, and points to the topic where each task is discussed.

Table 25. Channel tasks

Task to be performed	MQSC command	See topic
Define a channel	DEFINE CHANNEL	"Defining a channel" on page 247

Table 25. Channel tasks (continued)

Task to be performed	MQSC command	See topic
Alter a channel definition	ALTER CHANNEL	"Altering a channel definition" on page 247
Display a channel definition	DISPLAY CHANNEL	"Displaying a channel definition" on page 248
Delete a channel definition	DELETE CHANNEL	"Deleting a channel definition" on page 248
Start a channel initiator	START CHINIT	"Starting a channel initiator" on page 249
Stop a channel initiator	STOP CHINIT	"Stopping a channel initiator" on page 250
Display channel initiator information	DISPLAY CHINIT	"Displaying information about the channel initiator" on page 249
Start a channel listener	START LISTENER	"Starting a channel listener" on page 251
Stop a channel listener	STOP LISTENER	"Stopping a channel listener" on page 252
Start a channel	START CHANNEL	"Starting a channel" on page 252
Test a channel	PING CHANNEL	"Testing a channel" on page 254
Reset message sequence numbers for a channel	RESET CHANNEL	"Resetting message sequence numbers for a channel" on page 254
Resolve in-doubt messages on a channel	RESOLVE CHANNEL	"Resolving in-doubt messages on a channel" on page 255

Table 25. Channel tasks (continued)

Task to be performed	MQSC command	See topic
Stop a channel	STOP CHANNEL	“Stopping a channel” on page 255
Display channel status	DISPLAY CHSTATUS	“Displaying channel status” on page 257
Display cluster channels	DISPLAY CLUSQMGR	“Displaying cluster channels” on page 258

Defining a channel

To define a channel using the MQSC commands, use DEFINE CHANNEL.

Using the operations and control panels, starting from the initial panel, complete these fields and press Enter:

Field	Value
Action	2 (Define like)
Object type	channel type (for example SENDER) or CHANNEL
Name	
Disposition	The location of the new object.

You are presented with some panels to complete with information about the name and attributes you want for the channel you are defining. They are initialized with the default attribute values. Change any you want before pressing Enter.

Note: If you entered CHANNEL in the **object type** field, you are presented with the Select a Valid Channel Type panel first.

If you want to define a channel with the same attributes as an existing channel, put the name of the channel you want to copy in the **Name** field on the initial panel. The panels will be initialized with the attributes of the existing object.

For information about the channel attributes, see “Channel attributes” on page 71

Note:

1. You are strongly recommended to name all the channels in your network uniquely. As shown in Table 1 on page 26, including the source and target queue manager names in the channel name is a good way to do this.

Altering a channel definition

To alter a channel definition using the MQSC commands, use ALTER CHANNEL.

Using the operations and control panels, starting from the initial panel, complete these fields and press Enter:

Field	Value
Action	3 (Alter)

Field	Value
Object type	channel type (for example SENDER) or CHANNEL
Name	CHANNEL.TO.ALTER
Disposition	The location of the stored object.

You are presented with some panels containing information about the current attributes of the channel. Change any of the unprotected fields that you want by overtyping the new value, and then press Enter to change the channel definition.

For information about the channel attributes, see “Channel attributes” on page 71.

Displaying a channel definition

To display a channel definition using the MQSC commands, use DISPLAY CHANNEL.

Using the operations and control panels, starting from the initial panel, complete these fields and press Enter:

Field	Value
Action	1 (List or Display)
Object type	channel type (for example SENDER) or CHANNEL
Name	CHANNEL.TO.DISPLAY
Disposition	The location of the object.

You are presented with some panels displaying information about the current attributes of the channel.

For information about the channel attributes, see “Channel attributes” on page 71.

Deleting a channel definition

To delete a channel definition using the MQSC commands, use DELETE CHANNEL.

Using the operations and control panels, starting from the initial panel, complete these fields and press Enter:

Field	Value
Action	4 (Manage)
Object type	channel type (for example SENDER) or CHANNEL
Name	CHANNEL.TO.DELETE
Disposition	The location of the object.

You are presented with another panel. Select function type 1 on this panel.

Press Enter to delete the channel definition; you will be asked to confirm that you want to delete the channel definition by pressing Enter again.

Note: The channel initiator has to be running before a channel definition can be deleted (except for client-connection channels).

Displaying information about the channel initiator

To display information about the channel initiator using the MQSC commands, use DISPLAY CHINIT.

Using the operations and control panels, starting from the initial panel, complete these fields and press Enter:

Field	Value
Action	1 (Display)
Object type	SYSTEM
Name	Blank

You are presented with another panel. Select function type 1 on this panel.

Note:

1. Displaying distributed queuing information may take some time if you have lots of channels.
2. The channel initiator has to be running before you can display information about distributed queuing.

Starting a channel initiator

To start a channel initiator using the MQSC commands, use START CHINIT.

Using the operations and control panels, starting from the initial panel, complete these fields and press Enter:

Field	Value
Action	6 (Start)
Object type	SYSTEM
Name	Blank

The Start a System Function panel is displayed. The text following the panel below explains what action you should take.:

```

                                Start a System Function

Select function type, complete fields, then press Enter to start system
function.

Function type . . . . . _      1. Channel initiator
                                2. Channel listener
Action queue manager . . . : MQ25

Channel initiator
  JCL substitution . . . . . _____
                                _____

Channel listener
  Inbound disposition . . . Q  G=Group, Q=Qmgr
  Transport type . . . . . _  L=LU6.2, T=TCP/IP
  LU name (LU6.2) . . . . . _____
  Port number (TCP/IP) . . . 1414
  IP address (TCP/IP) . . . _____

Command ==>> _____
F1=Help      F2=Split      F3=Exit      F9=SwapNext F10=Messages F12=Cancel

```

Figure 38. Starting a system function

Select function type 1 (channel initiator), and press Enter.

Stopping a channel initiator

To stop a channel initiator using the MQSC commands, use STOP CHINIT.

Using the operations and control panels, starting from the initial panel, complete these fields and press Enter:

Field	Value
Action	7 (Stop)
Object type	SYSTEM
Name	Blank

The Stop a System Function panel is displayed. The text following the panel explains how you should use this panel:

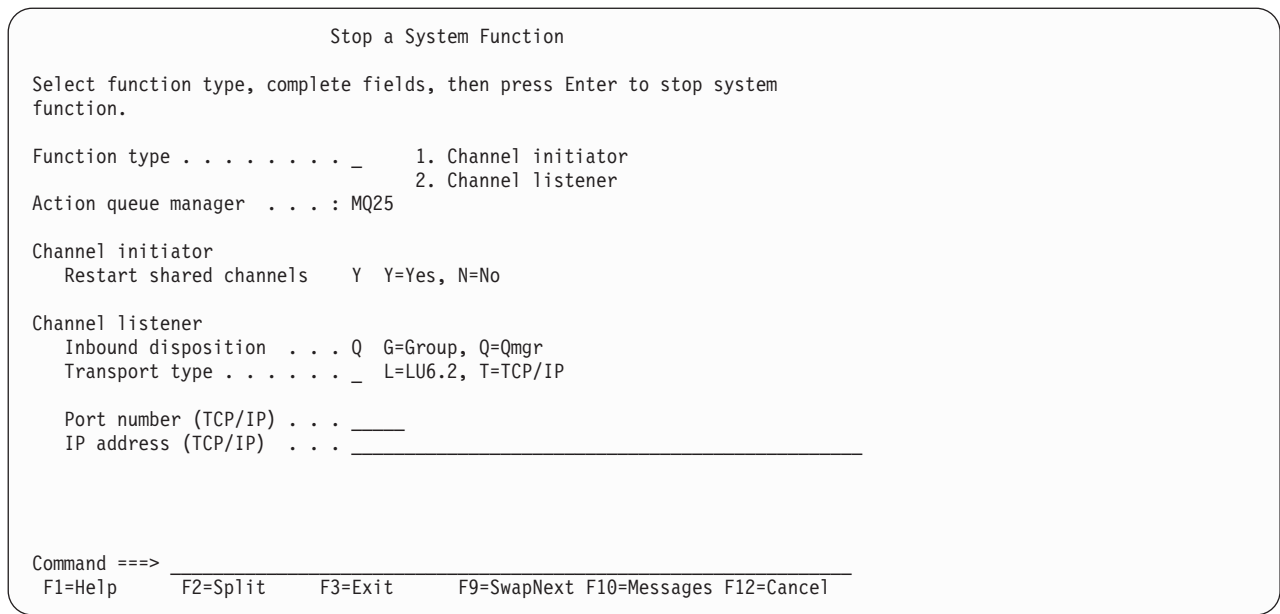


Figure 39. Stopping a function control

Select function type 1 (channel initiator) and press Enter.

The channel initiator will wait for all running channels to stop in quiesce mode before it stops.

Note: If some of the channels are receiver or requester channels that are running but not active, a stop request issued to either the receiver's or sender's channel initiator will cause it to stop immediately.

However, if messages are flowing, the channel initiator waits for the current batch of messages to complete before it stops.

Starting a channel listener

To start a channel listener using the MQSC commands, use START LISTENER.

Using the operations and control panels, starting from the initial panel, complete these fields and press Enter:

Field	Value
Action	6 (Start)
Object type	SYSTEM
Name	Blank

The Start a System Function panel is displayed (see Figure 38 on page 250).

Select function type 2 (channel listener). Select Inbound disposition. Select Transport type. If the Transport type is L, select LU name. If the Transport type is T, select Port number and (optionally) IP address. Press Enter.

Note: For the TCP/IP listener, you can start multiple combinations of Port and IP address.

Stopping a channel listener

To stop a channel listener using the MQSC commands, use STOP LISTENER.

Using the operations and control panels, starting from the initial panel, complete these fields and press Enter:

Field	Value
Action	7 (Stop)
Object type	SYSTEM
Name	Blank

The Stop a System Function panel is displayed (see Figure 39 on page 251).

Select function type 2 (channel listener). Select Inbound disposition. Select Transport type. If the transport type is 'T', select Port number and (optionally) IP address. Press Enter.

Note: For a TCP/IP listener, you can stop specific combinations of Port and IP address, or you can stop all combinations.

Starting a channel

To start a channel using the MQSC commands, use START CHANNEL.

Using the operations and control panels, starting from the initial panel, complete these fields and press Enter:

Field	Value
Action	6 (Start)
Object type	channel type (for example SENDER) or CHANNEL
Name	CHANNEL.TO.USE
Disposition	The disposition of the object.

The Start a Channel panel is displayed. The text following the panel explains how to use the panel.:


```

                                Start a Channel

Select disposition, then press Enter to start channel.

Channel name . . . . . : CHANNEL.TO.USE
Channel type . . . . . : SENDER
Description . . . . . : Description of CHANNEL.TO.USE

Disposition . . . . . P      P=Private on MQ25
                               S=Shared on MQ25
                               A=Shared on any queue manager

Command ==>
F1=Help      F2=Split      F3=Exit      F9=SwapNext F10=Messages F12=Cancel

```

Figure 40. Starting a channel

Select the disposition of the channel instance and on which queue manager it is to be started.

Press Enter to start the channel.

Starting a shared channel:

To start a shared channel, and keep it on a nominated channel initiator, use disposition = S (on the START CHANNEL command, specify CHLDISP(FIXSHARED)). There can be only one instance of the shared channel running at a time. Attempts to start a second instance of the channel will fail.

When you start a channel in this way, the following rules apply to that channel:

- You can stop the channel from any queue manager in the queue-sharing group. You can do this even if the channel initiator on which it was started is not running at the time you issue the stop-channel request. When the channel has stopped, you can restart it by specifying disposition = S (CHLDISP(FIXSHARED)) on the same, or another, channel initiator. You can also start it by specifying disposition = A (CHLDISP(SHARED)).
- If the channel is in the starting or retry state, you can restart it by specifying disposition = S (CHLDISP(FIXSHARED)) on the same or a different channel initiator. You can also start it by specifying disposition = A (CHLDISP(SHARED)).
- The channel is eligible to be trigger started when it goes into the inactive state. Shared channels that are trigger started always have a shared disposition (CHLDISP(SHARED)).
- The channel is eligible to be started with CHLDISP(FIXSHARED), on any channel initiator, when it goes into the inactive state. You can also start it by specifying disposition = A (CHLDISP(SHARED)).
- The channel is not recovered by any other active channel initiator in the queue-sharing group when the channel initiator on which it was started is stopped with SHARED(RESTART), or when the channel initiator terminates abnormally. The channel is recovered only when the channel initiator on which

it was started is next restarted. This stops failed channel-recovery attempts being passed to other channel initiators in the queue-sharing group, which would add to their workload.

Testing a channel

To test a channel using the MQSC commands, use PING CHANNEL.

Using the operations and control panels, starting from the initial panel, complete these fields and press Enter:

Field	Value
Action	5 (Perform)
Object type	SENDER, SERVER, or CHANNEL
Name	CHANNEL.TO.USE
Disposition	The disposition of the channel object.

The Perform a Channel Function panel is displayed. The text following the panel explains how to use the panel.:

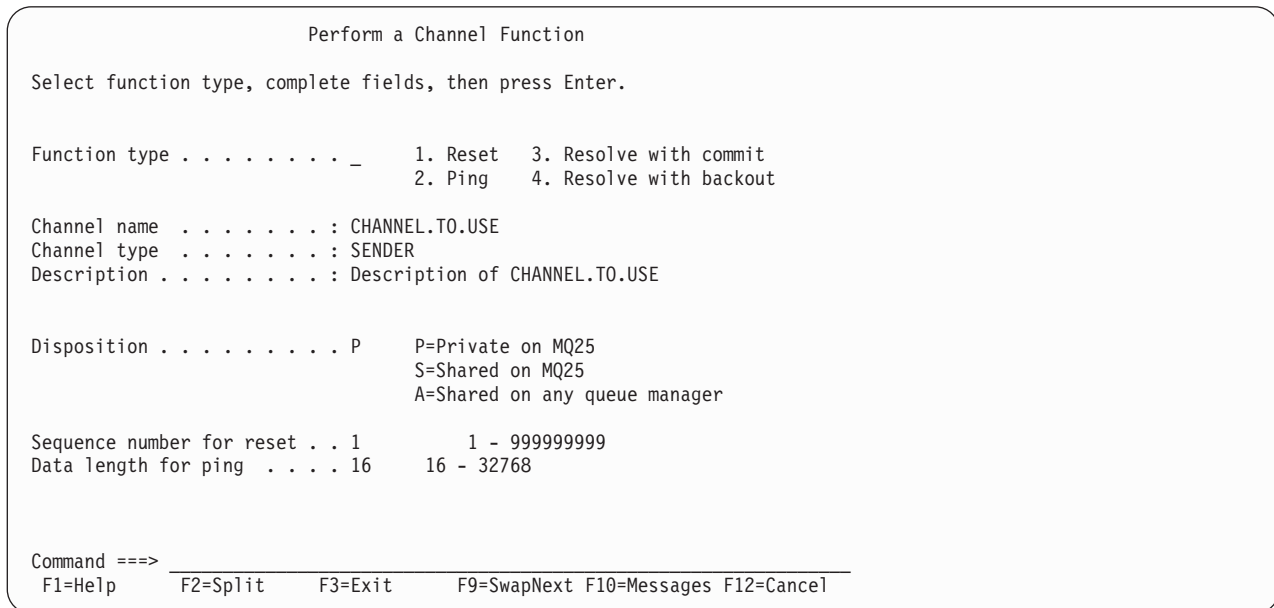


Figure 41. Testing a channel

Select function type 2 (ping).

Select the disposition of the channel for which the test is to be done and on which queue manager it is to be tested.

The data length is initially set to 16. Change it if you want and press Enter.

Resetting message sequence numbers for a channel

To reset channel sequence numbers using the MQSC commands, use RESET CHANNEL.

Using the operations and control panels, starting from the initial panel, complete these fields and press Enter:

Field	Value
Action	5 (Perform)
Object type	channel type (for example SENDER) or CHANNEL
Name	CHANNEL.TO.USE
Disposition	The disposition of the channel object.

The Perform a Channel Function panel is displayed (see Figure 41 on page 254).

Select Function type 1 (reset).

Select the disposition of the channel for which the reset is to be done and on which queue manager it is to be done.

The **sequence number** field is initially set to one. Change this if you want, and press Enter.

Resolving in-doubt messages on a channel

To resolve in-doubt messages on a channel using the MQSC commands, use RESOLVE CHANNEL.

Using the operations and control panels, starting from the initial panel, complete these fields and press Enter:

Field	Value
Action	5 (Perform)
Object type	SENDER, SERVER, or CHANNEL
Name	CHANNEL.TO.USE
Disposition	The disposition of the object.

The Perform a Channel Function panel is displayed (see Figure 41 on page 254).

Select Function type 3 or 4 (resolve with commit or backout). (See “In-doubt channels” on page 65 for more information.)

Select the disposition of the channel for which resolution is to be done and which queue manager it is to be done on. Press Enter.

Stopping a channel

To stop a channel using the MQSC commands, use STOP CHANNEL.

Using the operations and control panels, starting from the initial panel, complete these fields and press Enter:

Field	Value
Action	7 (Stop)
Object type	channel type (for example SENDER) or CHANNEL
Name	CHANNEL.TO.USE
Disposition	The disposition of the object.

The Stop a Channel panel is displayed. The text following the panel explains how to use the panel.:

```

                                Stop a Channel

Complete fields, then press Enter to stop channel.

Channel name . . . . . : CHANNEL.TO.USE
Channel type . . . . . : SENDER
Description . . . . . : Description of CHANNEL.TO.USE

Disposition . . . . . P      P=Private on MQ25
                               A=Shared on any queue manager

Stop mode . . . . . 1      1. Quiesce  2. Force
Stop status . . . . . 1      1. Stopped  2. Inactive

Queue manager . . . . . _____
Connection name . . . . . _____

Command ==> _____
F1=Help      F2=Split      F3=Exit      F9=SwapNext F10=Messages F12=Cancel

```

Figure 42. Stopping a channel

Select the disposition of the channel for which the stop is to be done and on which queue manager it is to be stopped.

Choose the stop mode that you require:

Quiesce

The channel will stop when the current message is completed and the batch will then be ended, even if the batch size value has not been reached and there are messages already waiting on the transmission queue. No new batches will be started. This is the default.

Force The channel stops immediately. If a batch of messages is in progress, an 'in-doubt' situation may result.

Choose the queue manager and connection name for the channel you want to stop.

Choose the status that you require:

Stopped

The channel will not be restarted automatically, and must be restarted manually. This is the default value if no queue manager or connection name is specified. If a name is specified, it is not allowed.

Inactive

The channel will be restarted automatically when required. This is the default value if a queue manager or connection name is specified.

Press Enter to stop the channel.

See "Stopping and quiescing channels" on page 62 for more information. For information about restarting stopped channels, see "Restarting stopped channels" on page 64.

Usage notes:

This section gives some usage notes about stopping a channel:

- If a shared channel is in a retry state and the channel initiator on which it was started is not running, a STOP request for the channel is issued on the queue manager where the command was entered.

Displaying channel status

To display the status of a channel or a set of channels using the MQSC commands, use DISPLAY CHSTATUS.

Note: Displaying channel status information may take some time if you have lots of channels.

Using the operations and control panels on the List Channel panel (see Figure 37 on page 245), a summary of the channel status is shown for each channel as follows:

INACTIVE	No connections are active
<i>status</i>	One connection is active
<i>nnn status</i>	More than one connection is current and all current connections have the same status
<i>nnn</i> CURRENT	More than one connection is current and the current connections do not all have the same status
Blank	WebSphere MQ is unable to determine how many connections are active (for example, because the channel initiator is not running)
	Note: For channel objects with the disposition GROUP, no status is displayed.

where *nnn* is the number of active connections, and *status* is one of the following:

INIT	INITIALIZING
BIND	BINDING
START	STARTING
RUN	RUNNING
STOP	STOPPING or STOPPED
RETRY	RETRYING
REQST	REQUESTING

To display more information about the channel status, press the Status key (F11) on the List Channel or the Display, or Alter channel panels to display the List Channels - Current Status panel (see Figure 43 on page 258).

```

List Channels - Current Status - MQ25                               Row 1 of 16
Type action codes, then press Enter. Press F11 to display saved status.
1=Display current status

Channel name      Connection name      State
Start time      Messages  Last message time  Type  Disposition
<> *
-----
RMA0.CIRCUIT.ACL.F  RMA1                STOP
2005-03-21 10.22.36  557735    2005-03-24 09.51.11 SENDER  PRIVATE MQ25
RMA0.CIRCUIT.ACL.N  RMA1                PRIVATE MQ25
2005-03-21 10.23.09  378675    2005-03-24 09.51.10 SENDER  PRIVATE MQ25
RMA0.CIRCUIT.CL.F   RMA2                PRIVATE MQ25
2005-03-24 01.12.51  45544     2005-03-24 09.51.08 SENDER  PRIVATE MQ25
RMA0.CIRCUIT.CL.N   RMA2                PRIVATE MQ25
2005-03-24 01.13.55  45560     2005-03-24 09.51.11 SENDER  PRIVATE MQ25
RMA1.CIRCUIT.CL.F   RMA1                PRIVATE MQ25
2005-03-21 10.24.12  360757    2005-03-24 09.51.11 RECEIVER PRIVATE MQ25
RMA1.CIRCUIT.CL.N   RMA1                PRIVATE MQ25
2005-03-21 10.23.40  302870    2005-03-24 09.51.09 RECEIVER PRIVATE MQ25
***** End of list *****

Command ==>
F1=Help      F2=Split  F3=Exit    F4=Filter  F5=Refresh  F7=Bkwd
F8=Fwd       F9=SwapNext F10=Messages F11=Saved  F12=Cancel

```

Figure 43. Listing channel connections

The values for status are as follows:

INIT	INITIALIZING
BIND	BINDING
START	STARTING
RUN	RUNNING
STOP	STOPPING or STOPPED
RETRY	RETRYING
REQST	REQUESTING
DOUBT	STOPPED and INDOUBT(YES)

See “Channel states” on page 55 for more information about these.

You can press F11 to see a similar list of channel connections with saved status; press F11 to get back to the **current** list. Note that the saved status does not apply until at least one batch of messages has been transmitted on the channel.

Use action code 1 (or a slash (/)) to select a connection and press Enter. The Display Channel Connection Current Status panels are displayed.

Displaying cluster channels

To display all the cluster channels that have been defined (explicitly or using auto-definition), use the MQSC command, DISPLAY CLUSQMGR.

Using the operations and control panels, starting from the initial panel, complete these fields and press Enter:

Field	Value
Action	1 (List or Display)
Object type	CLUSCHL
Name	*

You are presented with a panel like figure Figure 44, in which the information for each cluster channel occupies three lines, and includes its channel, cluster, and queue manager names. For cluster-sender channels, the overall state is shown.

```

List Cluster-queue-manager Channels - MQ25          Row 1 of 9

Type action codes, then press Enter.  Press F11 to display connection status.
1=Display  5=Perform  6=Start  7=Stop

Channel name      Connection name      State
Type            Cluster name
Cluster queue manager name      Disposition      Suspended
<> *
- TO.MQ90.T      HURSLEY.MACH90.COM(1590)  -      MQ25
- CLUSRCVR      VJH01T                  N
- MQ90          -      MQ25
- TO.MQ95.T      HURSLEY.MACH95.COM(1595)  -      RUN
- CLUSSDRA      VJH01T                  N
- MQ95          -      MQ25
- TO.MQ96.T      HURSLEY.MACH96.COM(1596)  -      RUN
- CLUSSDRB      VJH01T                  N
- MQ96          -      MQ25
***** End of list *****

Command ==>>
F1=Help      F2=Split      F3=Exit      F4=Filter      F5=Refresh      F7=Bkwd
F8=Fwd       F9=SwapNext   F10=Messages F11=Status     F12=Cancel

```

Figure 44. Listing cluster channels

To display full information about one or more channels, type Action code 1 against their names and press Enter. Use Action codes 5, 6, or 7 to perform functions (such as ping, resolve, and reset), and start or stop a cluster channel.

To display more information about the channel status, press the Status key (F11).

Preparing WebSphere MQ for z/OS

This chapter describes the WebSphere MQ for z/OS preparations you need to make before you can start to use distributed queuing. If you are using queue-sharing groups, see “Preparing WebSphere MQ for z/OS for DQM with queue-sharing groups” on page 280.

To enable distributed queuing, you must perform the following three tasks:

- Customize the distributed queuing facility and define the WebSphere MQ objects required; this is described in the WebSphere MQ for z/OS Concepts and Planning Guide and the WebSphere MQ for z/OS System Setup Guide.
- Define access security; this is described in the WebSphere MQ for z/OS System Setup Guide.
- Set up your communications; this is described in “Setting up communication for z/OS” on page 263.

Defining DQM requirements to WebSphere MQ

In order to define your distributed-queuing requirements, you have to:

- Define the channel initiator procedures and data sets
- Define the channel definitions

- Define the queues and other objects
- Define access security

See the WebSphere MQ for z/OS Concepts and Planning Guide for information about these tasks.

Defining WebSphere MQ objects

Use one of the WebSphere MQ command input methods to define WebSphere MQ objects. Refer to “Monitoring and controlling channels on z/OS” on page 243 for information about defining objects.

Transmission queues and triggering channels

Define the following:

- A local queue with the usage of XMITQ for each sending message channel.
- Remote queue definitions.

A remote queue object has three distinct uses, depending upon the way the name and content are specified:

- Remote queue definition
- Queue manager alias definition
- Reply-to queue alias definition

This is shown in Table 2 on page 31.

Use the TRIGDATA field on the transmission queue to trigger the specified channel. For example:

```
DEFINE QLOCAL(MYXMITQ) USAGE(XMITQ) TRIGGER +
      INITQ(SYSTEM.CHANNEL.INITQ) TRIGDATA(MYCHANNEL)
DEFINE CHL(MYCHANNEL) CHLTYPE(SDR) TRPTYPE(TCP) +
      XMITQ(MYXMITQ) CONNAME('9.20.9.30(1555)')
```

The supplied sample CSQ4INYD gives additional examples of the necessary definitions.

Synchronization queue

DQM requires a queue for use with sequence numbers and logical units of work identifiers (LUWID). You must ensure that a queue is available with the name SYSTEM.CHANNEL.SYNCQ (see WebSphere MQ for z/OS Concepts and Planning Guide). This queue must be available otherwise the channel initiator cannot start.

Make sure that you define this queue using INDXTYPE(MSGID). This will improve the speed at which they can be accessed.

Channel command queues

You need to ensure that a channel command queue exists for your system with the name SYSTEM.CHANNEL.INITQ.

If the channel initiator detects a problem with the SYSTEM.CHANNEL.INITQ, it will be unable to continue normally until the problem is corrected. The problem could be one of the following:

- The queue is full
- The queue is not enabled for put

- The page set that the queue is on is full
- The channel initiator does not have the correct security authorization to the queue

If the definition of the queue is changed to GET(DISABLED) while the channel initiator is running, it will not be able to get messages from the queue, and will terminate.

Starting the channel initiator

Triggering is implemented using the channel initiator. On WebSphere MQ for z/OS, this is started with the MQSC command START CHINIT.

Stopping the channel initiator

The channel initiator is stopped automatically when you stop the queue manager. If you need to stop the channel initiator but not the queue manager, you can use the MQSC command STOP CHINIT.

Other things to consider

Here are some other topics that you should consider when preparing WebSphere MQ for distributed queue management.

Operator messages

Because the channel initiator uses a number of asynchronously operating dispatchers, operator messages could appear on the log out of chronological sequence.

Channel operation commands

Channel operation commands generally involve two stages. When the command syntax has been checked and the existence of the channel verified, a request is sent to the channel initiator, and message CSQM134I or CSQM137I is sent to the command issuer to indicate the completion of the first stage. When the channel initiator has processed the command, further messages indicating its success or otherwise are sent to the command issuer along with message CSQ9022I or CSQ9023I respectively. Any error messages generated could also be sent to the z/OS console.

All cluster commands except DISPLAY CLUSQMGR, however, work asynchronously. Commands that change object attributes update the object and send a request to the channel initiator, and commands for working with clusters are checked for syntax and a request is sent to the channel initiator. In both cases, message CSQM130I is sent to the command issuer indicating that a request has been sent; this is followed by message CSQ9022I to indicate that the command has completed successfully, in that a request has been sent. It does not indicate that the cluster request has completed successfully. The requests sent to the channel initiator are processed asynchronously, along with cluster requests received from other members of the cluster. In some cases, these requests have to be sent to the whole cluster to determine if they are successful or not. Any errors are reported to the z/OS on the system where the channel initiator is running. They are not sent to the command issuer.

Undelivered-message queue

A DLQ handler is provided with WebSphere MQ for z/OS. See WebSphere MQ for z/OS System Administration Guide for more information.

Queues in use

MCAs for receiver channels may keep the destination queues open even when messages are not being transmitted; this results in the queues appearing to be 'in use'.

Security changes

If you change security access for a user ID, the change may not take effect immediately. (See one of WebSphere MQ for z/OS Concepts and Planning Guide, WebSphere MQ for z/OS System Setup Guide and WebSphere MQ for z/OS System Administration Guide for more information.)

Communications stopped

TCP:

If TCP is stopped for some reason and then restarted, the WebSphere MQ for z/OS TCP listener waiting on a TCP port is stopped.

Automatic channel reconnect allows the channel initiator to detect that TCP/IP is not available and to automatically restart the TCP/IP listener when TCP/IP returns. This alleviates the need for operations staff to notice the problem with TCP/IP and manually restart the listener. While the listener is out of action, the channel initiator can also be used to retry the listener at the interval specified by LSTRTMR in the channel initiator parameter module. These attempts can continue until TCP/IP returns and the listener successfully restarts automatically. For information about LSTRTMR, see the WebSphere MQ for z/OS System Setup Guide.

LU6.2:

If APPC is stopped, the listener is also stopped. Again, in this case, the listener automatically retries at the LSTRTMR interval so that, if APPC restarts, the listener can restart too.

If the DB2 fails, shared channels that are already running continue to run, but any new channel start requests will fail. When the DB2 is restored new requests are able to complete.

z/OS Automatic Restart Management (ARM)

Automatic restart management (ARM) is a z/OS recovery function that can improve the availability of specific batch jobs or started tasks (for example, subsystems), and therefore result in a faster resumption of productive work.

To use ARM, you must set up your queue managers and channel initiators in a particular way to make them restart automatically. For information about this, see WebSphere MQ for z/OS Concepts and Planning Guide.

Setting up communication for z/OS

DQM is a remote queuing facility for WebSphere MQ. It provides channel control programs for the queue manager that form the interface to communication links. These links are controllable by the system operator. The channel definitions held by distributed queuing management use these connections.

When a distributed-queuing management channel is started, it tries to use the connection specified in the channel definition. For this to succeed, it is necessary for the connection to be defined and available. This section explains how to do this.

You might also find it helpful to refer to “Example configuration - IBM WebSphere MQ for z/OS” on page 266. If you are using queue sharing groups, see “Setting up communication for WebSphere MQ for z/OS using queue-sharing groups” on page 284.

Deciding on a connection

There are two forms of communication protocol that can be used:

- TCP
- LU 6.2 through APPC/MVS

Each channel definition must specify only one protocol as the transmission protocol (Transport Type) attribute. A queue manager can use more than one protocol to communicate.

Defining a TCP connection

The TCP address space name must be specified in the TCP system parameters data set, *tcpip.TCPIP.DATA*. In the data set, a “*TCPIPJOBNAME TCPIP_proc*” statement must be included.

The channel initiator address space must have authority to read the data set. The following techniques can be used to access your *TCPIP.DATA* data set, depending on which TCP/IP product and interface you are using:

- Environment variable, *RESOLVER_CONFIG*
- HFS file, */etc/resolv.conf*
- *//SYSTCPD* DD statement
- *//SYSTCPDD* DD statement
- *jobname/userid.TCPIP.DATA*
- *SYS1.TCPPARMS(TCPDATA)*
- *zapname.TCPIP.DATA*

You must also be careful to specify the high-level qualifier for TCP/IP correctly.

You should have a suitably configured Domain Name System (DNS) server, capable of both Name to IP Address translation and IP Address to Name translation.

For more information, see the following:

- *TCP/IP OpenEdition®: Planning and Release Guide*, SC31-8303

- *z/OS Unix System Services Planning, GA22-7800*

Each TCP channel when started will use TCP resources; you may need to adjust the following parameters in your PROFILE.TCPIP configuration data set:

ACBPOOLSIZE

Add one per started TCP channel, plus one

CCBPOOLSIZE

Add one per started TCP channel, plus one per DQM dispatcher, plus one

DATABUFFERPOOLSIZE

Add two per started TCP channel, plus one

MAXFILEPROC

Controls how many channels each dispatcher in the channel initiator can handle.

This parameter is specified in the BPXPRMxx member of SYSI.PARMLIB. Ensure that you specify a value large enough for your needs.

By default, the channel initiator is only capable of binding to IP addresses associated with the stack named in the TCPNAME queue manager attribute. To allow the channel initiator to communicate using additional TCP/IP stacks on the system, you should change the TCPSTACK queue manager attribute to MULTIPLE.

Sending end

The connection name (CONNAME) field in the channel definition should be set to either the host name (for example MVSHUR1) or the TCP network address of the target, in IPv4 dotted decimal form (for example 9.20.9.30) or IPv6 hexadecimal form (for example fe80:43e4:0204:acff:fe97:2c34:fde0:3485). If the connection name is a host name, a TCP name server is required to convert the host name into a TCP host address. (This is a function of TCP, not WebSphere MQ.)

On the initiating end of a connection (sender, requester, and server channel types) it is possible to provide an optional port number for the connection, for example:

Connection name

9.20.9.30(1555)

In this case the initiating end will attempt to connect to a receiving program listening on port 1555.

The channel initiator can use any TCP/IP stack which is active and available. By default, the channel initiator will bind its outbound channels to the default IP address for the TCP/IP stack named in the TCPNAME queue manager attribute. To connect through a different stack, you should specify either the hostname or IP address of the stack in the LOCLADDR attribute of the channel.

Receiving on TCP

Receiving channel programs are started in response to a startup request from the sending channel. To do this, a listener program has to be started to detect incoming network requests and start the associated channel. You start this listener program with the START LISTENER command, or using the operations and control panels.

By default, the TCP Listener program uses port 1414 and listens on all addresses available to your TCP stack. You may start your TCP listener program to only

listen on a specific address or hostname by specifying IPADDR in the START LISTENER command. (For more information, see “Preparing WebSphere MQ for z/OS for DQM with queue-sharing groups” on page 280, “Listeners”.)

By default, TCP/IP listeners can bind only to addresses associated with the TCP/IP stack named in the TCPNAME queue manager attribute. To start listeners for other addresses, set your TCPSTACK queue manager attribute to ‘MULTIPLE’.

Using the TCP listener backlog option

When receiving on TCP/IP, a maximum number of outstanding connection requests is set. This can be considered a *backlog* of requests waiting on the TCP/IP port for the listener to accept the request.

The default listener backlog value on z/OS is 255. If the backlog reaches this values, the TCP/IP connection is rejected and the channel will not be able to start.

For MCA channels, this results in the channel going into a RETRY state and retrying the connection at a later time.

For client connections, the client receives an MQRC_Q_MGR_NOT_AVAILABLE reason code from MQCONN and should retry the connection at a later time.

Defining an LU6.2 connection

APPC/MVS setup

Each instance of the channel initiator must have the name of the LU that it is to use defined to APPC/MVS, in the APPCPMxx member of SYS1.PARMLIB, as in the following example:

```
LUADD ACBNAME(luname) NOSCHED TPDATA(CSQ.APPCTP)
```

luname is the name of the logical unit to be used. NOSCHED is required; TPDATA is not used. No additions are necessary to the ASCHPMxx member, or to the APPC/MVS TP profile data set.

The side information data set must be extended to define the connections used by DQM. See the supplied sample CSQ4SIDE for details of how to do this using the APPC utility program ATBSDFMU. For details of the TPNAME values to use, see the *Multiplatform APPC Configuration Guide* (“Red Book”) and the following table for information:

Table 26. Settings on the local z/OS system for a remote queue manager platform

Remote platform	TPNAME
z/OS, OS/390, or MVS/ESA	The same as TPNAME in the corresponding side information on the remote queue manager.
i5/OS	The same as the compare value in the routing entry on the i5/OS system.
HP OpenVMS	As specified in the OVMS Run Listener command.
Compaq NonStop Kernel	The same as the TPNAME specified in the receiver-channel definition.
UNIX systems	The same as TPNAME in the corresponding side information on the remote queue manager.

Table 26. Settings on the local z/OS system for a remote queue manager platform (continued)

Remote platform	TPNAME
Windows	As specified in the Windows Run Listener command, or the invokable Transaction Program that was defined using TpSetup on Windows.

If you have more than one queue manager on the same machine, ensure that the TPnames in the channel definitions are unique.

See the *Multiplatform APPC Configuration Guide* also for information about the VTAM definitions that may be required.

In an environment where the queue manager is communicating via APPC with a queue manager on the same or another z/OS system, ensure that either the VTAM definition for the communicating LU specifies SECACPT(ALREADYV), or that there is a RACF® APPCLU profile for the connection between LUs, which specifies CONVSEC(ALREADYV).

The z/OS command VARY ACTIVE must be issued against both base and listener LUs before attempting to start either inbound or outbound communications.

Connecting to APPC/MVS (LU 6.2):

The connection name (CONNNAME) field in the channel definition should be set to the symbolic destination name, as specified in the side information data set for APPC/MVS.

The LU name to use (defined to APPC/MVS as described above) must also be specified in the channel initiator parameters. It must be set to the same LU that will be used for receiving by the listener.

The channel initiator uses the "SECURITY(SAME)" APPC/MVS option, so it is the user ID of the channel initiator address space that is used for outbound transmissions, and will be presented to the receiver.

Receiving on LU 6.2:

Receiving MCAs are started in response to a startup request from the sending channel. To do this, a listener program has to be started to detect incoming network requests and start the associated channel. The listener program is an APPC/MVS server. You start it with the START LISTENER command, or using the operations and control panels. You must specify the LU name to use by means of a symbolic destination name defined in the side information data set. The local LU so identified must be the same as that used for outbound transmissions, as set in the channel initiator parameters.

Example configuration - IBM WebSphere MQ for z/OS

This chapter gives an example of how to set up communication links from WebSphere MQ for z/OS to WebSphere MQ products on the following platforms:

- Windows
- AIX
- Compaq Tru64 UNIX

- HP-UX
- Solaris
- Linux
- i5/OS
- VSE/ESA

You can also connect any of the following:

- z/OS to z/OS
- z/OS to MVS/ESA
- MVS/ESA to MVS/ESA

First it describes the parameters needed for an LU 6.2 connection; then it describes:

- “Establishing an LU 6.2 connection” on page 270
- “Establishing a TCP connection” on page 272

Once the connection is established, you need to define some channels to complete the configuration. This is described in “WebSphere MQ for z/OS configuration” on page 272.

See “Example configuration chapters in this book” on page 101 for background information about this chapter and how to use it.

Configuration parameters for an LU 6.2 connection

Table 27 presents a worksheet listing all the parameters needed to set up communication from z/OS to one of the other WebSphere MQ platforms. The worksheet shows examples of the parameters, which have been tested in a working environment, and leaves space for you to fill in your own values. An explanation of the parameter names follows the worksheet. Use the worksheet in this chapter in conjunction with the worksheet in the chapter for the platform to which you are connecting.

The steps required to set up an LU 6.2 connection are described in “Establishing an LU 6.2 connection” on page 270 with numbered cross references to the parameters on the worksheet.

Configuration worksheet

Use this worksheet to record the values you use for your configuration. Where numbers appear in the Reference column they indicate that the value must match that in the appropriate worksheet elsewhere in this book. The examples that follow in this chapter refer back to the values in the ID column. The entries in the Parameter Name column are explained in “Explanation of terms” on page 269.

Table 27. Configuration worksheet for z/OS using LU 6.2

ID	Parameter Name	Reference	Example Used	User Value
<i>Definition for local node</i>				
1	Command prefix		+cpf	
2	Network ID		NETID	
3	Node name		MVSPU	
4	Local LU name		MVSLU	

Table 27. Configuration worksheet for z/OS using LU 6.2 (continued)

ID	Parameter Name	Reference	Example Used	User Value
5	Symbolic destination		M1	
6	Modename		#INTER	
7	Local Transaction Program name		MQSERIES	
8	LAN destination address		400074511092	
Connection to a Windows system				
The values in this section of the table must match those used in Table 13 on page 130, as indicated.				
13	Symbolic destination		M3	
14	Modename	21	#INTER	
15	Remote Transaction Program name	7	MQSERIES	
16	Partner LU name	5	WINNTLU	
21	Remote node ID	4	05D 30F65	
Connection to an AIX system				
The values in this section of the table must match those used in Table 17 on page 156, as indicated.				
13	Symbolic Destination		M4	
14	Modename	18	#INTER	
15	Remote Transaction Program name	6	MQSERIES	
16	Partner LU name	4	AIXLU	
Connection to an HP-UX system				
The values in this section of the table must match those used in Table 19 on page 172, as indicated.				
13	Symbolic Destination		M5	
14	Modename	6	#INTER	
15	Remote Transaction Program name	7	MQSERIES	
16	Partner LU name	5	HPUXLU	
Connection to a Solaris system				
The values in this section of the table must match those used in Table 21 on page 195, as indicated.				
13	Symbolic destination		M7	
14	Modename	21	#INTER	
15	Remote Transaction Program name	8	MQSERIES	
16	Partner LU name	7	SOLARLU	
Connection to a Linux (x86 platform) system				
The values in this section of the table must match those used in Configuration worksheet for Communications Server for Linux, as indicated.				
13	Symbolic destination		M8	
14	Modename	6	#INTER	
15	Remote Transaction Program name	7	MQSERIES	
16	Partner LU name	5	LINUXLU	
Connection to an i5/OS system				
The values in this section of the table must match those used in Table 35 on page 351, as indicated.				
13	Symbolic Destination		M9	

Table 27. Configuration worksheet for z/OS using LU 6.2 (continued)

ID	Parameter Name	Reference	Example Used	User Value
14	Modename	21	#INTER	
15	Remote Transaction Program name	8	MQSERIES	
16	Partner LU name	3	AS400LU	
<i>Connection to a VSE/ESA system</i>				
The values in this section of the table must match those used in your VSE/ESA system.				
13	Symbolic destination		MA	
14	Modename		#INTER	
15	Remote Transaction Program name	4	MQ01	
16	Partner LU name	3	VSELU	

Explanation of terms

1 Command prefix

This is the unique command prefix of your WebSphere MQ for z/OS queue-manager subsystem. The z/OS systems programmer defines this at installation time, in SYS1.PARMLIB(IEFSSNss), and will be able to tell you the value.

2 Network ID

The VTAM startup procedure in your installation is partly customized by the ATCSTRxx member of the data set referenced by the DDNAME VTAMLST. The Network ID is the value specified for the NETID parameter in this member. For Network ID you must specify the name of the NETID that owns the WebSphere MQ communications subsystem (WebSphere MQ channel initiator). Your network administrator will tell you the value.

3 Node name

VTAM, being a low-entry network node, does not have a Control Point name for Advanced Peer-to-Peer Networking (APPN) use. It does however have a system services control point name (SSCPNAME). For node name, you must specify the name of the SSCP that owns the WebSphere MQ communications subsystem (WebSphere MQ channel initiator). This is defined in the same ATCSTRxx member as the Network ID. Your network administrator will tell you the value.

4 Local LU name

A logical unit (LU) is software that serves as an interface or translator between a transaction program and the network. It manages the exchange of data between transaction programs. The local LU name is the unique VTAM APPLID of this WebSphere MQ subsystem. Your network administrator will tell you this value.

5 13 Symbolic destination

This is the name you give to the CPI-C side information profile. You need a side information entry for each LU 6.2 listener.

6 14 Modename

This is the name given to the set of parameters that control the LU 6.2 conversation. An entry with this name and similar attributes must be defined at each end of the session. In VTAM, this corresponds to a mode table entry. Your network administrator will assign this to you.

7 15 Transaction Program name

WebSphere MQ applications trying to converse with this queue manager will specify a symbolic name for the program to be run at the receiving end. This will have been specified in the TPNAME attribute on the channel definition at the sender. For simplicity, wherever possible use a transaction program name of MQSERIES, or in the case of a connection to VSE/ESA, where the length is limited to 4 bytes, use MQTP.

See Table 26 on page 265 for more information.

8 LAN destination address

This is the LAN destination address that your partner nodes will use to communicate with this host. When you are using a 3745 network controller, it will be the value specified in the LOCADD parameter for the line definition to which your partner is physically connected. If your partner nodes use other devices such as 317X or 6611 devices, the address will have been set during the customization of those devices. Your network administrator will tell you this value.

16 Partner LU name

This is the LU name of the WebSphere MQ queue manager on the system with which you are setting up communication. This value is specified in the side information entry for the remote partner.

21 Remote node ID

For a connection to Windows, this is the ID of the local node on the Windows system with which you are setting up communication.

Establishing an LU 6.2 connection

To establish an LU 6.2 connection, there are two steps:

1. Define yourself to the network.
2. Define a connection to the partner.

Defining yourself to the network

1. SYS1.PARMLIB(APPCPMxx) contains the startup parameters for APPC. You must add a line to this file to tell APPC where to locate the sideinfo. This line should be of the form:

```
SIDEINFO  
    DATASET(APPC.APPCSI)
```

2. Add another line to SYS1.PARMLIB(APPCPMxx) to define the local LU name you intend to use for the WebSphere MQ LU 6.2 listener. The line you add should take the form:

```
LUADD ACBNAME(mvs lu)  
    NOSCHED  
    TPDATA(csq.appctp)
```

Specify values for ACBNAME(4) and TPDATA .

The NOSCHED parameter tells APPC that our new LU will not be using the LU 6.2 scheduler (ASCH), but has one of its own. TPDATA refers to the Transaction Program data set in which LU 6.2 stores information about transaction programs. Again, WebSphere MQ will not use this, but it is required by the syntax of the LUADD command.

3. Start the APPC subsystem with the command:

```
START APPC,SUB=MSTR,APPC=xx
```

where *xx* is the suffix of the PARMLIB member in which you added the LU in step 1.

Note: If APPC is already running, it can be refreshed with the command:

```
SET APPC=xx
```

The effect of this is cumulative, that is, APPC will not lose its knowledge of objects already defined to it in this or another PARMLIB member.

4. Add the new LU to a suitable VTAM major node definition. These are typically in SYS1.VTAMLST. The APPL definition will look similar to the sample shown in Figure 45.

```
MVSLU APPL ABCNAME=MVSLU,      4
           APPXC=YES,
           AUTOSES=0,
           DDRAINL=NALLOW,
           DLOGMOD=#INTER,     6
           DMINWML=10,
           DMINWNR=10,
           DRESPL=NALLOW,
           DSESLIM=60,
           LMDENT=19,
           MODETAB=MTICICS,
           PARSESS=YES,
           VERIFY=NONE,
           SECACPT=ALREADYV,
           SRBEXIT=YES
```

Figure 45. Channel Initiator APPL definition

5. Activate the major node. This can be done with the command:

```
V NET,ACT,ID=majornode
```

6. Add an entry defining your LU to the CPI-C side information data set. Use the APPC utility program ATBSDFMU to do this. Sample JCL is in *thlqual.SCSQPROC(CSQ4SIDE)* (where *thlqual* is the target library high-level qualifier for WebSphere MQ data sets in your installation.)

The entry you add will look like this:

```
SIADD
      DESTNAME(M1)           5
      MODENAME(#INTER)      6
      TPNAME(MQSERIES)       7
      PARTNER_LU(MVSLU)     4
```

7. Alter the queue manager object to use the correct distributed queueing parameters using the following command. You must specify the local LU (4) assigned to your queue manager in the LUNAME attribute of the queue manager .

```
ALTER QMGR LUNAME(MVSLU)
```

Defining a connection to a partner

Note: This example is for a connection to a Windows system but the task is the same for other platforms.

Add an entry to the CPI-C side information data set to define the connection. Sample JCL to do this is in *thlqual.SCSQPROC(CSQ4SIDE)*.

The entry you add will look like this:

```

SIADD
  DESTNAME(M3)           13
  MODENAME(#INTER)      14
  TPNAME(MQSERIES)      15
  PARTNER_LU(WINNTLU)   16

```

Establishing a TCP connection

Alter the queue manager object to use the correct distributed queueing parameters using the following command. You must add the name of the TCP address space to the TCPNAME queue manager attribute.

```
ALTER QMGR TCPNAME(TCPIP)
```

What next?

The TCP connection is now established. You are ready to complete the configuration. Go to “WebSphere MQ for z/OS configuration.”

WebSphere MQ for z/OS configuration

1. Start the channel initiator using the command:

```
+cpf START CHINIT 1
```

2. Start an LU 6.2 listener using the command:

```
+cpf START LSTR LUNAME(M1) TRPTYPE(LU62)
```

The LUNAME of M1 refers to the symbolic name you gave your LU (5). You must specify TRPTYPE(LU62), otherwise the listener will assume you want TCP.

3. Start a TCP listener using the command:

```
+cpf START LSTR
```

If you wish to use a port other than 1414 (the default WebSphere MQ port), use the command:

```
+cpf START LSTR PORT(1555)
```

WebSphere MQ channels will not initialize successfully if the channel negotiation detects that the message sequence number is different at each end. You may need to reset this manually.

Channel configuration

The following sections detail the configuration to be performed on the z/OS queue manager to implement the channel described in Figure 32 on page 101.

Examples are given for connecting WebSphere MQ for z/OS and WebSphere MQ for Windows. If you wish to connect to WebSphere MQ on another platform use the appropriate set of values from the table in place of those for Windows.

Note: The words in **bold** are user-specified and reflect the names of WebSphere MQ objects used throughout these examples. If you change the names used here, ensure that you also change the other references made to these objects throughout this book. All others are keywords and should be entered as shown.

Table 28. Configuration worksheet for WebSphere MQ for z/OS

ID	Parameter Name	Reference	Example Used	User Value
<i>Definition for local node</i>				

Table 28. Configuration worksheet for WebSphere MQ for z/OS (continued)

ID	Parameter Name	Reference	Example Used	User Value
A	Queue Manager Name		MVS	
B	Local queue name		MVS.LOCALQ	
Connection to WebSphere MQ for Windows				
The values in this section of the table must match those used in Table 14 on page 146, as indicated.				
C	Remote queue manager name	A	WINNT	
D	Remote queue name		WINNT.REMOTEQ	
E	Queue name at remote system	B	WINNT.LOCALQ	
F	Transmission queue name		WINNT	
G	Sender (LU 6.2) channel name		MVS.WINNT.SNA	
H	Sender (TCP) channel name		MVS.WINNT.TCP	
I	Receiver (LU 6.2) channel name	G	WINNT.MVS.SNA	
J	Receiver (TCP/IP) channel name	H	WINNT.MVS.TCP	
Connection to WebSphere MQ for AIX				
The values in this section of the table must match those used in Table 18 on page 168, as indicated.				
C	Remote queue manager name	A	AIX	
D	Remote queue name		AIX.REMOTEQ	
E	Queue name at remote system	B	AIX.LOCALQ	
F	Transmission queue name		AIX	
G	Sender (LU 6.2) channel name		MVS.AIX.SNA	
H	Sender (TCP/IP) channel name		MVS.AIX.TCP	
I	Receiver (LU 6.2) channel name	G	AIX.MVS.SNA	
J	Receiver (TCP/IP) channel name	H	AIX.MVS.TCP	
Connection to MQSeries for Compaq Tru64 UNIX				
The values in this section of the table must match those used in your Compaq Tru64 UNIX system.				
C	Remote queue manager name	A	DECUX	
D	Remote queue name		DECUX.REMOTEQ	
E	Queue name at remote system	B	DECUX.LOCALQ	
F	Transmission queue name		DECUX	
H	Sender (TCP) channel name		DECUX.MVS.TCP	
J	Receiver (TCP) channel name	H	MVS.DECUX.TCP	
Connection to WebSphere MQ for HP-UX				
The values in this section of the table must match those used in Table 20 on page 191, as indicated.				
C	Remote queue manager name	A	HPUX	
D	Remote queue name		HPUX.REMOTEQ	
E	Queue name at remote system	B	HPUX.LOCALQ	
F	Transmission queue name		HPUX	
G	Sender (LU 6.2) channel name		MVS.HPUX.SNA	
H	Sender (TCP) channel name		MVS.HPUX.TCP	
I	Receiver (LU 6.2) channel name	G	HPUX.MVS.SNA	

Table 28. Configuration worksheet for WebSphere MQ for z/OS (continued)

ID	Parameter Name	Reference	Example Used	User Value
J	Receiver (TCP) channel name	H	HPUX.MVS.TCP	
Connection to WebSphere MQ for Solaris				
The values in this section of the table must match those used in Table 22 on page 212, as indicated.				
C	Remote queue manager name	A	SOLARIS	
D	Remote queue name		SOLARIS.REMOTEQ	
E	Queue name at remote system	B	SOLARIS.LOCALQ	
F	Transmission queue name		SOLARIS	
G	Sender (LU 6.2) channel name		MVS.SOLARIS.SNA	
H	Sender (TCP) channel name		MVS.SOLARIS.TCP	
I	Receiver (LU 6.2) channel name	G	SOLARIS.MVS.SNA	
J	Receiver (TCP/IP) channel name	H	SOLARIS.MVS.TCP	
Connection to WebSphere MQ for Linux				
The values in this section of the table must match those used in Table 24 on page 235, as indicated.				
C	Remote queue manager name	A	LINUX	
D	Remote queue name		LINUX.REMOTEQ	
E	Queue name at remote system	B	LINUX.LOCALQ	
F	Transmission queue name		LINUX	
G	Sender (LU 6.2) channel name		MVS.LINUX.SNA	
H	Sender (TCP) channel name		MVS.LINUX.TCP	
I	Receiver (LU 6.2) channel name	G	LINUX.MVS.SNA	
J	Receiver (TCP/IP) channel name	H	LINUX.MVS.TCP	
Connection to WebSphere MQ for i5/OS				
The values in this section of the table must match those used in Table 36 on page 364, as indicated.				
C	Remote queue manager name	A	AS400	
D	Remote queue name		AS400.REMOTEQ	
E	Queue name at remote system	B	AS400.LOCALQ	
F	Transmission queue name		AS400	
G	Sender (LU 6.2) channel name		MVS.AS400.SNA	
H	Sender (TCP/IP) channel name		MVS.AS400.TCP	
I	Receiver (LU 6.2) channel name	G	AS400.MVS.SNA	
J	Receiver (TCP/IP) channel name	H	AS400.MVS.TCP	
Connection to MQSeries for VSE/ESA				
The values in this section of the table must match those used in your VSE/ESA system.				
C	Remote queue manager name	A	VSE	
D	Remote queue name		VSE.REMOTEQ	
E	Queue name at remote system	B	VSE.LOCALQ	
F	Transmission queue name		VSE	
G	Sender channel name		MVS.VSE.SNA	
I	Receiver channel name	G	VSE.MVS.SNA	

WebSphere MQ for z/OS sender-channel definitions using LU 6.2:

```
Local Queue
  Object type : QLOCAL
  Name : WINNT F
  Usage : X (XmitQ)

Remote Queue
  Object type : QREMOTE
  Name : WINNT.REMOTEQ D
  Name on remote system : WINNT.LOCALQ E
  Remote system name : WINNT C
  Transmission queue : WINNT F

Sender Channel
  Channel name : MVS.WINNT.SNA G
  Transport type : L (LU6.2)
  Transmission queue name : WINNT F
  Connection name : M3 13
```

WebSphere MQ for z/OS receiver-channel definitions using LU 6.2:

```
Local Queue
  Object type : QLOCAL
  Name : MVS.LOCALQ B
  Usage : N (Normal)

Receiver Channel
  Channel name : WINNT.MVS.SNA I
```

WebSphere MQ for z/OS sender-channel definitions using TCP:

```
Local Queue
  Object type : QLOCAL
  Name : WINNT F
  Usage : X (XmitQ)

Remote Queue
  Object type : QREMOTE
  Name : WINNT.REMOTEQ D
  Name on remote system : WINNT.LOCALQ E
  Remote system name : WINNT C
  Transmission queue : WINNT F

Sender Channel
  Channel name : MVS.WINNT.TCP H
  Transport type : T (TCP)
  Transmission queue name : WINNT F
  Connection name : winnt.tcpip.hostname
```

WebSphere MQ for z/OS receiver-channel definitions using TCP:

```
Local Queue
  Object type : QLOCAL
  Name : MVS.LOCALQ B
  Usage : N (Normal)

Receiver Channel
  Channel name : WINNT.MVS.TCP J
```

Message channel planning example for z/OS

This chapter provides a detailed example of how to connect z/OS, OS/390, or MVS/ESA queue managers together so that messages can be sent between them.

The example illustrates the preparations needed to allow an application using queue manager QM1 to put messages on a queue at queue manager QM2. An application running on QM2 can retrieve these messages, and send responses to a reply queue on QM1.

The example illustrates the use of both TCP/IP and LU 6.2 connections. The example assumes that channels are to be triggered to start when the first message arrives on the transmission queue they are servicing.

What the example shows

This example shows the WebSphere MQ commands (MQSC) that you can use in WebSphere MQ for z/OS for DQM.

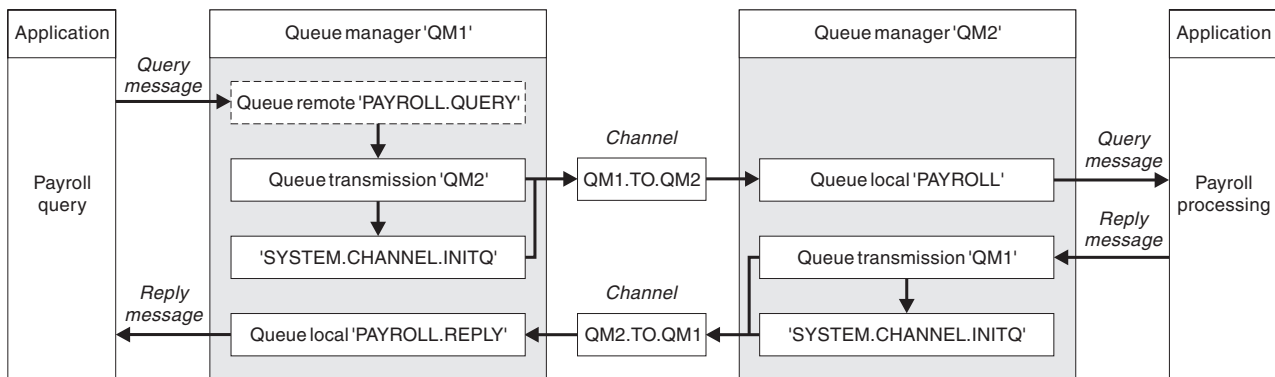


Figure 46. The first example for WebSphere MQ for z/OS

It involves a payroll query application connected to queue manager QM1 that sends payroll query messages to a payroll processing application running on queue manager QM2. The payroll query application needs the replies to its queries sent back to QM1. The payroll query messages are sent from QM1 to QM2 on a sender-receiver channel called QM1.TO.QM2, and the reply messages are sent back from QM2 to QM1 on another sender-receiver channel called QM2.TO.QM1. Both of these channels are triggered to start as soon as they have a message to send to the other queue manager.

The payroll query application puts a query message to the remote queue "PAYROLL.QUERY" defined on QM1. This remote queue definition resolves to the local queue "PAYROLL" on QM2. In addition, the payroll query application specifies that the reply to the query is sent to the local queue "PAYROLL.REPLY" on QM1. The payroll processing application gets messages from the local queue "PAYROLL" on QM2, and sends the replies to wherever they are required; in this case, local queue "PAYROLL.REPLY" on QM1.

Both queue managers are assumed to be running on z/OS. In the example definitions for TCP/IP, QM1 has a host address of 9.20.9.31 and is listening on port 1411, and QM2 has a host address of 9.20.9.32 and is listening on port 1412. In the definitions for LU 6.2, QM1 is listening on a symbolic luname called LUNAME1 and QM2 is listening on a symbolic luname called LUNAME2. The example assumes that these are already defined on your z/OS system and available for use. To define them, see "Example configuration - IBM WebSphere MQ for z/OS" on page 266.

The object definitions that need to be created on QM1 are:

- Remote queue definition, PAYROLL.QUERY
- Transmission queue definition, QM2 (default=remote queue manager name)
- Sender channel definition, QM1.TO.QM2
- Receiver channel definition, QM2.TO.QM1
- Reply-to queue definition, PAYROLL.REPLY

The object definitions that need to be created on QM2 are:

- Local queue definition, PAYROLL
- Transmission queue definition, QM1 (default=remote queue manager name)
- Sender channel definition, QM2.TO.QM1
- Receiver channel definition, QM1.TO.QM2

The example assumes that all the SYSTEM.COMMAND.* and SYSTEM.CHANNEL.* queues required to run DQM have been defined as shown in the supplied sample definitions, CSQ4INSG and CSQ4INSX.

The connection details are supplied in the CONNAME attribute of the sender channel definitions.

You can see a diagram of the arrangement in Figure 46 on page 276.

Queue manager QM1 example

The following object definitions allow applications connected to queue manager QM1 to send request messages to a queue called PAYROLL on QM2, and to receive replies on a queue called PAYROLL.REPLY on QM1.

All the object definitions have been provided with the DESCR and REPLACE attributes. The other attributes supplied are the minimum required to make the example work. The attributes that are not supplied take the default values for queue manager QM1.

Run the following commands on queue manager QM1.

Remote queue definition:

```
DEFINE QREMOTE(PAYROLL.QUERY) DESCR('Remote queue for QM2') REPLACE +
PUT(ENABLED) XMITQ(QM2) RNAME(PAYROLL) RQMNAME(QM2)
```

Note: The remote queue definition is not a physical queue, but a means of directing messages to the transmission queue, QM2, so that they can be sent to queue manager QM2.

Transmission queue definition:

```
DEFINE QLOCAL(QM2) DESCR('Transmission queue to QM2') REPLACE +
USAGE(XMITQ) PUT(ENABLED) GET(ENABLED) TRIGGER TRIGTYPE(FIRST) +
TRIGDATA(QM1.TO.QM2) INITQ(SYSTEM.CHANNEL.INITQ)
```

When the first message is put on this transmission queue, a trigger message is sent to the initiation queue, SYSTEM.CHANNEL.INITQ. The channel initiator gets the message from the initiation queue and starts the channel identified in the trigger data. The channel initiator can only get trigger messages from the SYSTEM.CHANNEL.INITQ queue, so you should not use any other queue as the initiation queue.

Sender channel definition:

For a TCP/IP connection:

```
DEFINE CHANNEL(QM1.TO.QM2) CHLTYPE(SDR) TRPTYPE(TCP) +  
REPLACE DESCR('Sender channel to QM2') XMITQ(QM2) +  
CONNAME('9.20.9.32(1412)')
```

For an LU 6.2 connection:

```
DEFINE CHANNEL(QM1.TO.QM2) CHLTYPE(SDR) TRPTYPE(LU62) +  
REPLACE DESCR('Sender channel to QM2') XMITQ(QM2) +  
CONNAME('LUNAME2')
```

Receiver channel definition:

For a TCP/IP connection:

```
DEFINE CHANNEL(QM2.TO.QM1) CHLTYPE(RCVR) TRPTYPE(TCP) +  
REPLACE DESCR('Receiver channel from QM2')
```

For an LU 6.2 connection:

```
DEFINE CHANNEL(QM2.TO.QM1) CHLTYPE(RCVR) TRPTYPE(LU62) +  
REPLACE DESCR('Receiver channel from QM2')
```

Reply-to queue definition:

```
DEFINE QLOCAL(PAYROLL.REPLY) REPLACE PUT(ENABLED) GET(ENABLED) +  
DESCR('Reply queue for replies to query messages sent to QM2')
```

The reply-to queue is defined as PUT(ENABLED). This ensures that reply messages can be put to the queue. If the replies cannot be put to the reply-to queue, they are sent to the dead-letter queue on QM1 or, if this queue is not available, remain on transmission queue QM1 on queue manager QM2. The queue has been defined as GET(ENABLED) to allow the reply messages to be retrieved.

Queue manager QM2 example

The following object definitions allow applications connected to queue manager QM2 to retrieve request messages from a local queue called PAYROLL, and to put replies to these request messages to a queue called PAYROLL.REPLY on queue manager QM1.

You do not need to provide a remote queue definition to enable the replies to be returned to QM1. The message descriptor of the message retrieved from local queue PAYROLL contains both the reply-to queue and the reply-to queue manager names. Therefore, as long as QM2 can resolve the reply-to queue manager name to that of a transmission queue on queue manager QM2, the reply message can be sent. In this example, the reply-to queue manager name is QM1 and so queue manager QM2 simply requires a transmission queue of the same name.

All the object definitions have been provided with the DESCR and REPLACE attributes and are the minimum required to make the example work. The attributes that are not supplied take the default values for queue manager QM2.

Run the following commands on queue manager QM2.

Local queue definition:

```
DEFINE QLOCAL(PAYROLL) REPLACE PUT(ENABLED) GET(ENABLED) +  
DESCR('Local queue for QM1 payroll details')
```

This queue is defined as PUT(ENABLED) and GET(ENABLED) for the same reason as the reply-to queue definition on queue manager QM1.

Transmission queue definition:

```
DEFINE QLOCAL(QM1) DESCR('Transmission queue to QM1') REPLACE +
USAGE(XMITQ) PUT(ENABLED) GET(ENABLED) TRIGGER TRIGTYPE(FIRST) +
TRIGDATA(QM2.TO.QM1) INITQ(SYSTEM.CHANNEL.INITQ)
```

When the first message is put on this transmission queue, a trigger message is sent to the initiation queue, SYSTEM.CHANNEL.INITQ. The channel initiator gets the message from the initiation queue and starts the channel identified in the trigger data. The channel initiator can only get trigger messages from SYSTEM.CHANNEL.INITQ so you should not use any other queue as the initiation queue.

Sender channel definition:

For a TCP/IP connection:

```
DEFINE CHANNEL(QM2.TO.QM1) CHLTYPE(SDR) TRPTYPE(TCP) +
REPLACE DESCR('Sender channel to QM1') XMITQ(QM1) +
CONNAME('9.20.9.31(1411)')
```

For an LU 6.2 connection:

```
DEFINE CHANNEL(QM2.TO.QM1) CHLTYPE(SDR) TRPTYPE(LU62) +
REPLACE DESCR('Sender channel to QM1') XMITQ(QM1) +
CONNAME('LUNAME1')
```

Receiver channel definition:

For a TCP/IP connection:

```
DEFINE CHANNEL(QM1.TO.QM2) CHLTYPE(RCVR) TRPTYPE(TCP) +
REPLACE DESCR('Receiver channel from QM1')
```

For an LU 6.2 connection:

```
DEFINE CHANNEL(QM1.TO.QM2) CHLTYPE(RCVR) TRPTYPE(LU62) +
REPLACE DESCR('Receiver channel from QM1')
```

Running the example

When you have created the required objects, you must:

- Start the channel initiator for both queue managers
- Start the listener for both queue managers

The applications can then send messages to each other. Because the channels are triggered to start by the arrival of the first message on each transmission queue, you do not need to issue the START CHANNEL MQSC command.

For details about starting a channel initiator see “Starting a channel initiator” on page 249, and for details about starting a listener see “Starting a channel listener” on page 251.

Expanding this example

This example can be expanded by:

- Adding more queue, and channel definitions to allow other applications to send messages between the two queue managers.
- Adding user exit programs on the channels to allow for link encryption, security checking, or additional message processing.

- Using queue manager aliases and reply-to queue aliases to understand more about how these can be used in the organization of your queue manager network.

Preparing WebSphere MQ for z/OS for DQM with queue-sharing groups

This chapter describes the concept of distributed queuing with queue-sharing groups on WebSphere MQ for z/OS.

It also describes the components of distributed queuing in this environment and the WebSphere MQ preparations required before DQM can be used with queue-sharing groups, and the advantages of using DQM with queue-sharing groups.

For information on how to monitor and control channels when using queue-sharing groups, see “Monitoring and controlling channels on z/OS” on page 243.

Concepts

This section describes the concepts related to distributed queuing with queue-sharing groups. For additional information on the concepts of shared queues and queue-sharing groups, see WebSphere MQ for z/OS Concepts and Planning Guide, “*Shared queues*” .

Class of service

A shared queue is a type of local queue that offers a different class of service. Messages on a shared queue are stored in a coupling facility (CF), which allows them to be accessed by all queue managers in the queue-sharing group. A message on a shared queue must be a message of length no more than 100MB.

Generic interface

A queue-sharing group has a generic interface that allows the network to view the group as a single entity. This is achieved by having a single generic address that can be used to connect to any queue manager within the group.

Each queue manager in the queue-sharing group listens for inbound session requests on an address that is logically related to the generic address. For more information see “Listeners” below.

Components

What follows is a description of the components required to enable distributed queuing with queue-sharing groups.

Listeners

The group LU 6.2 and TCP/IP listeners listen on an address that is logically connected to the generic address.

For the LU 6.2 listener, the specified LUGROUP is mapped to the VTAM generic resource associated with the queue-sharing group. For an example of setting up this technology, see Table 26 on page 265.

For the TCP/IP listener, the specified port has three mutually exclusive means of being connected to the generic address:

- In the case of a front-end router such as the IBM Network dispatcher (see *Network Dispatcher User's Guide, GC31-8496-03*), inbound connect requests are forwarded from the router to the members of the queue-sharing group.
- In the case of TCP/IP WLM/DNS, each listener registers as being part of the WLM group. This is a registration type model, similar to the VTAM generic resource for LU 6.2. For an example of setting up this technology, see "Using WLM/DNS" on page 290. WLM/DNS only maps hostname and does not map port numbers. This means that all the group listeners in a queue-sharing group must use the same port number. Use the WebSphere MQ command (MQSC) as shown in the following examples:
 - On queue manager QM1:

```
START LSTR PORT(2424) INDISP(GROUP) +
IPADDR(QM1.MACH.IBM.COM)
```
 - On queue manager QM2:

```
START LSTR PORT(2424) INDISP(GROUP) +
IPADDR(QM2.MACH.IBM.COM)
```
- In the case of TCP/IP's Sysplex Distributor, each listener that is running and is listening on a particular address that is set up as a Distributed DVIPA will be allocated a proportion of the incoming requests. For an example of setting up this technology, see "Using Sysplex Distributor" on page 291

Transmission queues and triggering

A shared transmission queue is used to store messages before they are moved from the queue-sharing group to the destination. It is a shared queue and it is accessible to all queue managers in the queue-sharing group.

Triggering:

A triggered shared queue can generate more than one trigger message for a satisfied trigger condition. There is one trigger message generated for each local initiation queue defined on a queue manager in the queue-sharing group associated with the triggered shared queue.

In the case of distributed queuing, each channel initiator receives a trigger message for a satisfied shared transmission queue trigger condition. However, only one channel initiator will actually process the triggered start, and the others will fail safely. The triggered channel is then started with a load balanced start (see "Load-balanced channel start" on page 283) that will be triggered to start channel QSG.TO.QM2. To create a shared transmission queue, use the WebSphere MQ commands (MQSC) as shown in the following example:

```
DEFINE QLOCAL(QM2) DESCR('Transmission queue to QM2') +
USAGE(XMITQ) QSGDISP(SHARED) +
CFSTRUCT(APPLICATION1) INITQ(SYSTEM.CHANNEL.INITQ) +
TRIGGER TRIGDATA(QSG.TO.QM2)
```

Message channel agents

A channel can only be started on a channel initiator if it has access to a channel definition for a channel with that name. A channel definition can be defined to be private to a queue manager or stored on the shared repository and available anywhere (a group definition). This means that a group defined channel is available on any channel initiator in the queue-sharing group.

Note: The private copy of the group definition can be changed or deleted.

To create group channel definitions, use the WebSphere MQ commands (MQSC) as shown in the following examples:

```
DEFINE CHL(QSG.TO.QM2) CHLTYPE(SDR) +  
TRPTYPE(TCP) CONNAME(QM2.MACH.IBM.COM) +  
XMITQ(QM2) QSGDISP(GROUP)  
  
DEFINE CHL(QM2.TO.QSG) CHLTYPE(RCVR) TRPTYPE(TCP) +  
QSGDISP(GROUP)
```

There are two perspectives from which to look at the message channel agents used for distributed queuing with queue-sharing groups:

- Inbound
- Outbound

Inbound:

An inbound channel is a shared channel if it is connected to the queue manager through the group listener. It is connected either through the generic interface to the queue-sharing group, then directed to a queue manager within the group, or targeted at a specific queue manager's group port or the luname used by the group listener.

Outbound:

An outbound channel is a shared channel if it moves messages from a shared transmission queue. In the above example commands, sender channel QSG.TO.QM2 is a shared channel because its transmission queue, QM2 is defined with QSGDISP(SHARED).

Synchronization queue

Shared channels have their own shared synchronization queue called SYSTEM.QSG.CHANNEL.SYNCQ, which is accessible to any member of the queue-sharing group. (Private channels continue to use the private synchronization queue. See "Synchronization queue" on page 260) This means that the channel can be restarted on a different queue manager and channel initiator instance within the queue-sharing group in the event of failure of the communications subsystem, channel initiator or queue manager. (See "Shared channel recovery" on page 283 for details.)

DQM with queue-sharing groups requires that a shared queue is available with the name SYSTEM.QSG.CHANNEL.SYNCQ. This queue must be available so that a group listener can successfully start.

If a group listener fails because the queue was not available, the queue can be defined and the listener can be restarted without recycling the channel initiator, and the non-shared channels are not affected.

Make sure that you define this queue using INDXTYPE(MSGID). This will improve the speed at which they can be accessed.

Benefits

The following section describes the benefits of shared queuing, which are:

- Load-balanced channel start
- Shared channel recovery
- Client channels

Load-balanced channel start

A shared transmission queue can be serviced by an outbound channel running on any channel initiator in the queue-sharing group. Load-balanced channel start determines where a start channel command is targeted. An appropriate channel initiator is chosen that has access to the necessary communications subsystem. For example, a channel defined with TRPTYPE(LU6.2) will not be started on a channel initiator that only has access to a TCP/IP subsystem.

The choice of channel initiator is dependant on the channel load and the headroom of the channel initiator. The channel load is the number of active channels as a percentage of the maximum number of active channels allowed as defined in the channel initiator parameters. The headroom is the difference between the number of active channels and the maximum number allowed.

Inbound shared channels can be load-balanced across the queue-sharing group by use of a generic address, as described in “Listeners” on page 280.

Shared channel recovery

The following table shows the types of shared-channel failure and how each type is handled.

Type of failure:	What happens:
Channel initiator communications subsystem failure	The channels dependent on the communications subsystem enter channel retry, and are restarted on an appropriate queue-sharing group channel initiator by a load-balanced start command.
Channel initiator failure	The channel initiator fails, but the associated queue manager remains active. The queue manager monitors the failure and initiates recovery processing.
Queue manager failure	The queue manager fails (failing the associated channel initiator). Other queue managers in the queue-sharing group monitor the event and initiate peer recovery.
Shared status failure	Channel state information is stored in DB2, so a loss of connectivity to DB2 becomes a failure when a channel state change occurs. Running channels can carry on running without access to these resources. On a failed access to DB2, the channel enters retry.

Shared channel recovery processing on behalf of a failed system requires connectivity to DB2 to be available on the system managing the recovery to retrieve the shared channel status.

Client channels

Client connection channels can benefit from the high availability of messages in queue-sharing groups that are connected to the generic interface instead of being connected to a specific queue manger. (For more information about this, see the WebSphere MQ Clients manual.)

Clusters and queue-sharing groups

You can make your shared queue available to a cluster in a single definition. To do this you specify the name of the cluster when you define the shared queue.

Users in the network see the shared queue as being hosted by each queue manager within the queue-sharing group (the shared queue is not advertised as being hosted by the queue-sharing group). Clients can start sessions with all members of the queue-sharing group to put messages to the same shared queue.

For more information, see *WebSphere MQ Queue Manager Clusters*.

Channels and serialization

If a queue manager in a queue-sharing group fails while a message channel agent is dealing with uncommitted messages on one or more shared queues, the channel and the associated channel initiator will end, and shared queue peer recovery will take place for the queue manager.

Because shared queue peer recovery is an asynchronous activity, peer channel recovery might try to simultaneously restart the channel in another part of the queue sharing group before shared queue peer recovery is complete. If this happens, committed messages might be processed ahead of the messages still being recovered. To ensure that messages are not processed out of sequence in this way, message channel agents that process messages on shared queues serialize their access to these queues by issuing the MQCONNX API call.

An attempt to start a channel for which shared queue peer recovery is still in progress might result in a failure. An error message indicating that recovery is in progress is issued, and the channel is put into retry state. Once queue manager peer recovery is complete, the channel can restart at the time of the next retry.

An attempt to RESOLVE, PING, or DELETE a channel can fail for the same reason.

Intra-group queuing

Intra-group queuing (IGQ) can effect potentially fast and less-expensive small message transfer between queue managers within a queue-sharing group (QSG), without the need to define channels between the queue managers. That is, intra-group queuing can be used to deliver, more efficiently, small messages to queues residing on remote queue managers within a queue-sharing group. See “Intra-group queuing” on page 298 for more information.

Setting up communication for WebSphere MQ for z/OS using queue-sharing groups

When a distributed-queuing management channel is started, it tries to use the connection specified in the channel definition. For this to succeed, it is necessary for the connection to be defined and available. This section explains how to do this.

You might also find it useful to refer to “Example configuration - IBM WebSphere MQ for z/OS using queue-sharing groups” on page 286.

Deciding on a connection

There are two forms of communication protocol that can be used:

- TCP
- LU 6.2 through APPC/MVS

Defining a TCP connection

For information on setting up your TCP, see “Defining a TCP connection” on page 263.

Sending end

The connection name (CONNNAME) field in the channel definition to connect to your queue sharing group should be set to the generic interface of your queue-sharing group (see “Generic interface” on page 280). If you are using DNS/WLM, the generic interface is the name in the DNSGROUP queue manager attribute. If it is not set, it is the queue-sharing group name. For details of DNSGROUP, see WebSphere MQ Script (MQSC) Command Reference.

Receiving on TCP using a queue-sharing group

Receiving shared channel programs are started in response to a startup request from the sending channel. To do this, a listener has to be started to detect incoming network requests and start the associated channel. You start this listener program with the START LISTENER command, using the inbound disposition of the group, or using the operations and control panels.

All group listeners in the queue-sharing group must be listening on the same port. If you have more than one channel initiator running on a single MVS image you can define virtual IP addresses and start your TCP listener program to only listen on a specific address or hostname by specifying IPADDR in the START LISTENER command. (For more information, see “Preparing WebSphere MQ for z/OS for DQM with queue-sharing groups” on page 280.)

Defining an LU6.2 connection

For information on setting up APPC/MVS, see “Setting up communication for z/OS” on page 263.

Connecting to APPC/MVS (LU 6.2)

The connection name (CONNNAME) field in the channel definition to connect to your queue-sharing group should be set to the symbolic destination name, as specified in the side information data set for APPC/MVS. The partner LU specified in this symbolic destination should be the generic resource name. See “Defining yourself to the network using generic resources” on page 289 for more details.

Receiving on LU 6.2 using a generic interface

Receiving shared MCAs are started in response to a startup request from the sending channel. To do this, a group listener program has to be started to detect incoming network requests and start the associated channel. The listener program is an APPC/MVS server. You start it with the START LISTENER command, using an inbound disposition group, or using the operations and control panels. You

must specify the LU name to use by means of a symbolic destination name defined in the side information data set. See “Defining yourself to the network using generic resources” on page 289 for more details.

Example configuration - IBM WebSphere MQ for z/OS using queue-sharing groups

This chapter gives an example of how to set up communication links from a queue-sharing group on WebSphere MQ for z/OS to WebSphere MQ products on the following platforms:

- Windows
- AIX

(You can also connect from z/OS to z/OS.)

First it describes the parameters needed for an LU 6.2 connection; then it describes:

- “Establishing an LU 6.2 connection into a queue-sharing group” on page 288
- “Establishing a TCP connection into a queue-sharing group” on page 290

Setting up communication links from a queue-sharing group to a distributed platform is the same as described in “Example configuration - IBM WebSphere MQ for z/OS” on page 266. There are examples to other platforms in that chapter.

When the connection is established, you need to define some channels to complete the configuration. This is described in “WebSphere MQ for z/OS shared channel configuration” on page 291.

See “Example configuration chapters in this book” on page 101 for background information about this chapter and how to use it.

Configuration parameters for an LU 6.2 connection

Table 29 on page 287 presents a worksheet listing all the parameters needed to set up communication from z/OS to one of the other WebSphere MQ platforms. The worksheet shows examples of the parameters, which have been tested in a working environment, and leaves space for you to fill in your own values. An explanation of the parameter names follows the worksheet. Use the worksheet in this chapter in conjunction with the worksheet in the chapter for the platform to which you are connecting.

The steps required to set up an LU 6.2 connection are described in “Establishing an LU 6.2 connection into a queue-sharing group” on page 288, with numbered cross references to the parameters on the worksheet.

Configuration worksheet

Use this worksheet to record the values you use for your configuration. Where numbers appear in the Reference column they indicate that the value must match that in the appropriate worksheet elsewhere in this book. The examples that follow in this chapter refer back to the values in the ID column. The entries in the Parameter Name column are explained in “Explanation of terms” on page 287.

Table 29. Configuration worksheet for z/OS using LU 6.2

ID	Parameter Name	Reference	Example Used	User Value
<i>Definition for local node using generic resources</i>				
1	Command prefix		+cpf	
2	Network ID		NETID	
3	Node name		MVSPU	
6	Modename		#INTER	
7	Local Transaction Program name		MQSERIES	
8	LAN destination address		400074511092	
9	Local LU name		MVSLU1	
10	Generic resource name		MVSGR	
11	Symbolic destination		G1	
12	Symbolic destination for generic resource name		G2	
<i>Connection to a Windows system</i>				
The values in this section of the table must match those used in Table 13 on page 130, as indicated.				
13	Symbolic destination		M3	
14	Modename	21	#INTER	
15	Remote Transaction Program name	7	MQSERIES	
16	Partner LU name	5	WINNTLU	
21	Remote node ID	4	05D 30F65	
<i>Connection to an AIX system</i>				
The values in this section of the table must match those used in Table 17 on page 156, as indicated.				
13	Symbolic Destination		M4	
14	Modename	18	#INTER	
15	Remote Transaction Program name	6	MQSERIES	
16	Partner LU name	4	AIXLU	

Explanation of terms

1 Command prefix

This is the unique command prefix of your WebSphere MQ for z/OS queue-manager subsystem. The z/OS systems programmer defines this at installation time, in SYS1.PARMLIB(IEFSSNss), and will be able to tell you the value.

2 Network ID

The VTAM startup procedure in your installation is partly customized by the ATCSTRxx member of the data set referenced by the DDNAME VTAMLST. The Network ID is the value specified for the NETID parameter in this member. For Network ID you must specify the name of the NETID that owns the WebSphere MQ communications subsystem. Your network administrator will tell you the value.

3 Node name

VTAM, being a low-entry network node, does not have a Control Point name for Advanced Peer-to-Peer Networking (APPN) use. It does however have a system services control point name (SSCPNAME). For node name,

you must specify the name of the SSCP that owns the WebSphere MQ communications subsystem. This is defined in the same ATCSTRxx member as the Network ID. Your network administrator will tell you the value.

9 Local LU name

A logical unit (LU) is software that serves as an interface or translator between a transaction program and the network. It manages the exchange of data between transaction programs. The local LU name is the unique VTAM APPLID of this WebSphere MQ subsystem. Your network administrator will tell you this value.

11 12 13 Symbolic destination

This is the name you give to the CPI-C side information profile. You need a side information entry for each LU 6.2 listener.

6 14 Modename

This is the name given to the set of parameters that control the LU 6.2 conversation. An entry with this name and similar attributes must be defined at each end of the session. In VTAM, this corresponds to a mode table entry. Your network administrator will assign this to you.

7 15 Transaction Program name

WebSphere MQ applications trying to converse with this queue manager will specify a symbolic name for the program to be run at the receiving end. This will have been specified in the TPNAME attribute on the channel definition at the sender. For simplicity, wherever possible use a transaction program name of MQSERIES, or in the case of a connection to VSE/ESA, where the length is limited to 4 bytes, use MQTP.

See Table 26 on page 265 for more information.

8 LAN destination address

This is the LAN destination address that your partner nodes will use to communicate with this host. When you are using a 3745 network controller, it will be the value specified in the LOCADD parameter for the line definition to which your partner is physically connected. If your partner nodes use other devices such as 317X or 6611 devices, the address will have been set during the customization of those devices. Your network administrator will tell you this value.

10 Generic resource name

A generic resource name is a unique name assigned to a group of LU names used by the channel initiators in a queue-sharing group.

16 Partner LU name

This is the LU name of the WebSphere MQ queue manager on the system with which you are setting up communication. This value is specified in the side information entry for the remote partner.

21 Remote node ID

For a connection to Windows, this is the ID of the local node on the Windows system with which you are setting up communication.

Establishing an LU 6.2 connection into a queue-sharing group

To establish an LU 6.2 connection, there are two steps:

1. Define yourself to the network using generic resources.
2. Define a connection to the partner.

Defining yourself to the network using generic resources

This example describes how to use VTAM Generic Resources to have one connection name to connect to the queue-sharing group.

1. SYS1.PARMLIB(APPCCPMxx) contains the start-up parameters for APPC. You must add a line to this file to tell APPC where to locate the sideinfo. This line should be of the form:

```
SIDEINFO
  DATASET(APPCCPMxx)
```

2. Add another line to SYS1.PARMLIB(APPCCPMxx) to define the local LU name you intend to use for the WebSphere MQ LU 6.2 group listener. The line you add should take the form

```
LUADD ACBNAME(mvsLu1)
      NOSCHED
      TPDATA(csq.appctp)
      GRNAME(mvsg)
```

Specify values for ACBNAME (9), TPDATA and GRNAME(10).

The NOSCHED parameter tells APPC that our new LU will not be using the LU 6.2 scheduler (ASCH), but has one of its own. TPDATA refers to the Transaction Program data set in which LU 6.2 stores information about transaction programs. Again, WebSphere MQ will not use this, but it is required by the syntax of the LUADD command.

3. Start the APPC subsystem with the command:

```
START APPC,SUB=MSTR,APPC=xx
```

where xx is the suffix of the PARMLIB member in which you added the LU in step 1.

Note: If APPC is already running, it can be refreshed with the command:

```
SET APPC=xx
```

The effect of this is cumulative, that is, APPC will not lose its knowledge of objects already defined to it in this or another PARMLIB member.

4. Add the new LU to a suitable VTAM major node definition. These are typically in SYS1.VTAMLST. The APPL definition will look similar to the sample shown.

```
        MVSLU APPL ACBNAME=MVSLU1,      9
              APPXC=YES,
              AUTOSSES=0,
              DDRAINL=NALLOW,
              DLOGMOD=#INTER,          6
              DMINWML=10,
              DMINWNR=10,
              DRESPL=NALLOW,
              DSESLIM=60,
              LMDENT=19,
              MODETAB=MTCICS,
              PARSESS=YES,
              VERIFY=NONE,
              SECACPT=ALREADYV,
              SRBEXIT=YES
```

5. Activate the major node. This can be done with the command:

```
V,NET,ACT,majornode
```

6. Add entries defining your LU and generic resource name to the CPI-C side information data set. Use the APPC utility program ATBSDFMU to do this. Sample JCL is in *thlqual*.SCSQPROC(CSQ4SIDE) (where *thlqual* is the target library high-level qualifier for WebSphere MQ data sets in your installation.)

The entries you add will look like this:

```
SIADD
  DESTNAME(G1)          11
  MODENAME(#INTER)
  TPNAME(MQSERIES)
  PARTNER_LU(MVSLU1)   9
SIADD
  DESTNAME(G2)          12
  MODENAME(#INTER)
  TPNAME(MQSERIES)
  PARTNER_LU(MVSGR)    10
```

- Alter the queue manager object to use the correct distributed queueing parameters using the following command. You must specify the local LU (9) assigned to your queue manager in the LUGROUP attribute of the queue manager.

```
ALTER QMGR LUGROUP(MVSLU1)
```

Defining a connection to a partner

Note: This example is for a connection to a Windows system but the task is the same for other platforms.

Add an entry to the CPI-C side information data set to define the connection. Sample JCL to do this is in *thlqual.SCSQPROC(CSQ4SIDE)*.

The entry you add will look like this:

```
SIADD
  DESTNAME(M3)          13
  MODENAME(#INTER)     14
  TPNAME(MQSERIES)     15
  PARTNER_LU(WINNTLU)  16
```

What next?

The connection is now established. You are ready to complete the configuration. Go to “WebSphere MQ for z/OS shared channel configuration” on page 291.

Establishing a TCP connection into a queue-sharing group

Alter the queue manager object to use the correct distributed queueing parameters using the following command. You must add the name of the TCP address space to the TCPNAME queue manager attribute.

Using WLM/DNS

Alter the queue manager object to use the correct distributed queueing parameters.

You must set DNSWLM(YES); optionally you can add the name of the group name to be used as a hostname to the DNSGROUP attribute. If you leave the name blank, the queue-sharing group name is used.

```
ALTER QMGR TCPNAME(TCPIP) DNSWLM(YES) DNSGROUP(MYGROUP)
```

WLM/DNS does not offer any support for mapping one incoming port number to a different outgoing port number. This means that each channel initiator must use a different hostname, by one of the following methods:

- Run each channel initiator on a different MVS image
- Run each channel initiator with a different TCP stack on the same MVS image.

- Have multiple Virtual IP Addresses (VIPAs) on one TCP stack.
- Use the TCP/IP SHAREPORT option to allow the same port to be used for multiple listeners.

See *z/OS Communications Server: IP User's Guide and Commands*, SC31-8780 for more information on VIPA.

See *z/OS Communications Server IP Configuration Reference*, SC31-8776 for more information on SHAREPORT.

See *TCP/IP in a sysplex*, SG24-5235, an IBM Redbooks™ publication, for more information on WLM/DNS.

Using Sysplex Distributor

This example shows how to set up Sysplex Distributor to use one connection name to connect to the queue-sharing group.

1. Define a Distributed DVIPA address as follows:
 - a. Add a DYNAMICXCF statement to the IPCONFIG. This is used for inter-image connectivity via dynamically created XCF TCP/IP links.
 - b. Use the VIPADYNAMIC block on each image in the Sysplex.
 - 1) On the owning image, code a VIPADefine statement to create the DVIPA. Then code a VIPADISTRIBUTE statement to distribute it to all other or selected images.
 - 2) On the backup image(s), code a VIPABACKUP statement for the DVIPA address.
2. If more than one channel initiator will be started on any LPAR in the sysplex then add the SHAREPORT option for the port to be shared in the PORT reservation list in the PROFILE data set.

See *z/OS CS: IP Configuration Guide* and *z/OS CS: IP Configuration Reference* for more information.

Sysplex Distributor will balance the inbound connections between each LPAR. If there is more than one channel initiator on an LPAR, then the use of SHAREPORT will pass that inbound connection to the listener port with the smallest number of connections.

What next?

The TCP connection is now established. You are ready to complete the configuration. Go to “WebSphere MQ for z/OS shared channel configuration.”

WebSphere MQ for z/OS shared channel configuration

1. Start the channel initiator using the command:


```
+cpf START CHINIT 1
```
2. Start an LU6.2 group listener using the command:


```
+cpf START LSTR LUNAME(G1) TRPTYPE(LU62) INDISP(GROUP)
```

The LUNAME of G1 refers to the symbolic name you gave your LU (11).
3. If you are using Virtual IP Addressing, either with WLM/DNS or Sysplex Distributor and wish to listen on a specific address, use the command:


```
+cpf START LSTR PORT(1555) INDISP(GROUP) IPADDR(mvsvipa)
```

There can be only one instance of the shared channel running at a time. If you try to start a second instance of the channel it will fail (the error message varies depending on other factors). The shared synchronization queue keeps track of the channel status.

WebSphere MQ channels will not initialize successfully if the channel negotiation detects that the message sequence number is different at each end. You may need to reset this manually.

Shared channel configuration

The following sections detail the configuration to be performed on the z/OS queue manager to implement the channel described in Figure 32 on page 101.

Examples are given for connecting WebSphere MQ for z/OS and Windows. If you wish to connect to WebSphere MQ on another platform use the appropriate set of values from the table in place of those for Windows.

Note: The words in **bold** are user-specified and reflect the names of WebSphere MQ objects used throughout these examples. If you change the names used here, ensure that you also change the other references made to these objects throughout this book. All others are keywords and should be entered as shown.

Table 30. Configuration worksheet for WebSphere MQ for z/OS using queue-sharing groups

ID	Parameter Name	Reference	Example Used	User Value
<i>Definition for local node</i>				
A	Queue Manager Name		QSG	
B	Local queue name		QSG.SHAREDQ	
<i>Connection to WebSphere MQ for Windows</i>				
The values in this section of the table must match those used in Table 14 on page 146, as indicated.				
C	Remote queue manager name	A	WINNT	
D	Remote queue name		WINNT.REMOTEQ	
E	Queue name at remote system	B	WINNT.LOCALQ	
F	Transmission queue name		WINNT	
G	Sender (LU 6.2) channel name		QSG.WINNT.SNA	
H	Sender (TCP) channel name		QSG.WINNT.TCP	
I	Receiver (LU 6.2) channel name	G	WINNT.QSG.SNA	
J	Receiver (TCP/IP) channel name	H	WINNT.QSG.TCP	
<i>Connection to WebSphere MQ for AIX</i>				
The values in this section of the table must match those used in Table 18 on page 168, as indicated.				
C	Remote queue manager name		AIX	
D	Remote queue name		AIX.REMOTEQ	
E	Queue name at remote system	B	AIX.LOCALQ	
F	Transmission queue name		AIX	
G	Sender (LU 6.2) channel name		QSG.AIX.SNA	
H	Sender (TCP/IP) channel name		QSG.AIX.TCP	
I	Receiver (LU 6.2) channel name	G	AIX.QSG.SNA	
J	Receiver (TCP/IP) channel name	H	AIX.QSG.TCP	

WebSphere MQ for z/OS shared sender-channel definitions using LU 6.2:

```
Local Queue
  Object type : QLOCAL
  Name : WINNT F
  Usage : X (XmitQ)
  Disposition : SHARED

Remote Queue
  Object type : QREMOTE
  Name : WINNT.REMOTEQ D
  Name on remote system : WINNT.LOCALQ E
  Remote system name : WINNT C
  Transmission queue : WINNT F
  Disposition : GROUP

Sender Channel
  Channel name : MVS.WINNT.SNA G
  Transport type : L (LU6.2)
  Transmission queue name : WINNT F
  Connection name : M3 13
  Disposition : GROUP
```

WebSphere MQ for z/OS shared receiver-channel definitions using LU 6.2:

```
Local Queue
  Object type : QLOCAL
  Name : QSG.SHAREDQ B
  Usage : N (Normal)
  Disposition : SHARED

Receiver Channel
  Channel name : WINNT.QSG.SNA I
  Disposition : GROUP
```

WebSphere MQ for z/OS shared sender-channel definitions using TCP:

```
Local Queue
  Object type : QLOCAL
  Name : WINNT F
  Usage : X (XmitQ)
  Disposition : SHARED

Remote Queue
  Object type : QREMOTE
  Name : WINNT.REMOTEQ D
  Name on remote system : WINNT.LOCALQ E
  Remote system name : WINNT C
  Transmission queue : WINNT F
  Disposition : GROUP

Sender Channel
  Channel name : QSG.WINNT.TCP H
  Transport type : T (TCP)
  Transmission queue name : WINNT F
  Connection name : winnt.tcpip.hostname
  Disposition : GROUP
```

WebSphere MQ for z/OS shared receiver-channel definitions using TCP:

```
Local Queue
  Object type : QLOCAL
  Name : QSG.SHAREDQ B
  Usage : N (Normal)
  Disposition : SHARED
```

```
Receiver Channel
Channel name : WINNT.QSG.TCP    J
Disposition : GROUP
```

Message channel planning example for z/OS using queue-sharing groups

This example illustrates the preparations needed to allow an application using queue manager QM3 to put a message on a queue in a queue-sharing group that has queue members QM4 and QM5.

It is recommended that you are familiar with the example in “Message channel planning example for z/OS” on page 275 before trying this example.

What this example shows

This example shows the WebSphere MQ commands (MQSC) that you can use in WebSphere MQ for z/OS for distributed queuing with queue-sharing groups. This example expands the payroll query scenario of the example in “Message channel planning example for z/OS” on page 275 to show how to add higher availability of query processing by adding more serving applications to serve a shared queue.

The payroll query application is now connected to queue manager QM3 and puts a query to the remote queue 'PAYROLL QUERY' defined on QM3. This remote queue definition resolves to the shared queue 'PAYROLL' hosted by the queue managers in the queue-sharing group QSG1. The payroll processing application now has two instances running, one connected to QM4 and one connected to QM5.

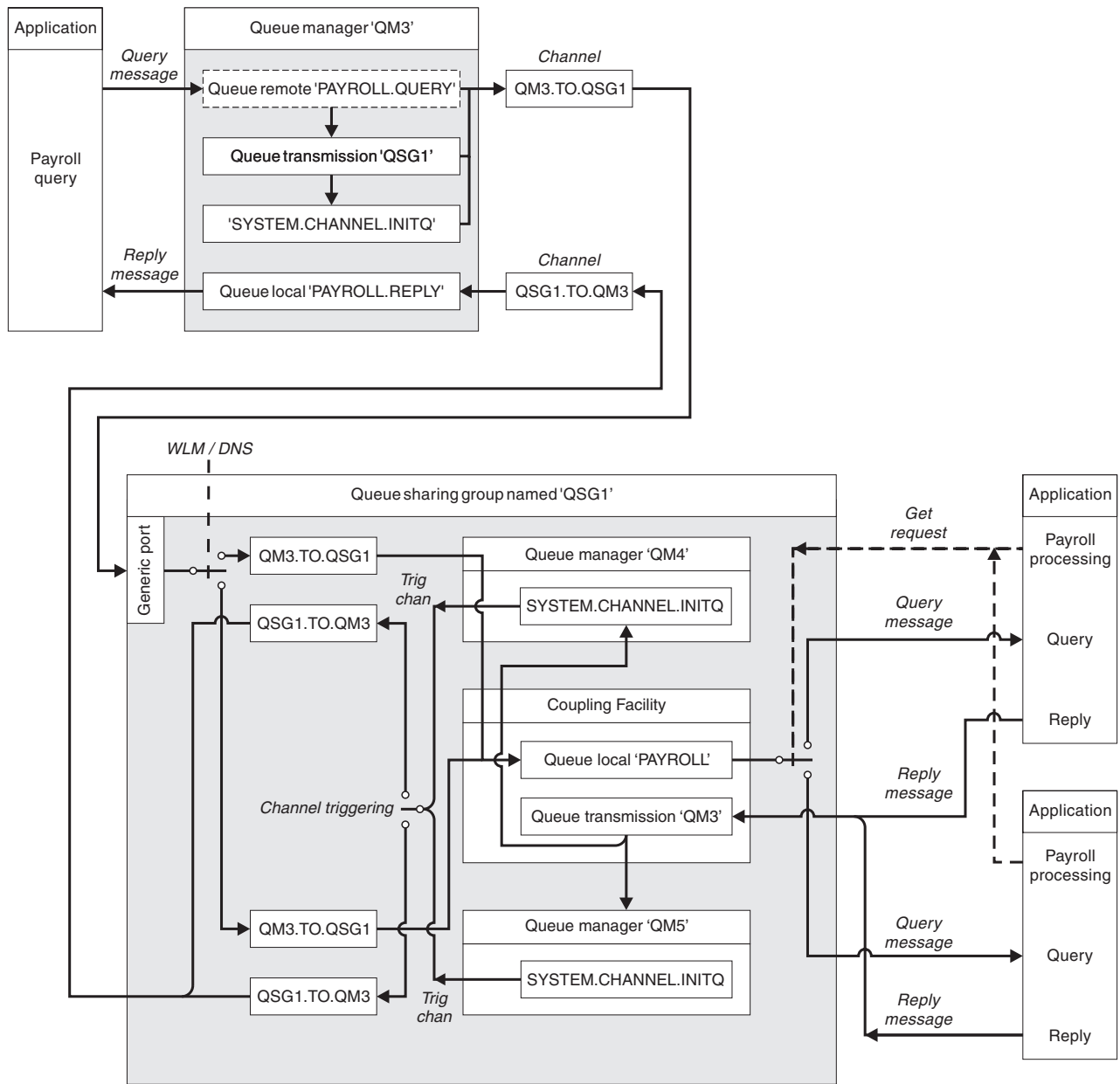


Figure 47. Message channel planning example for WebSphere MQ for z/OS using queue-sharing groups

All three queue managers are assumed to be running on z/OS. In the example definitions for TCP/IP, QM4 has a host name of MVSIP01 and QM5 has a host name of MVSIP02. Both queue managers are listening on port 1414 and have registered to use WLM/DNS. The generic address that WLM/DNS provides for this group is QSG1.MVSIP. QM3 has a host address of 9.20.9.31 and is listening on port 1411.

In the example definitions for LU6.2, QM3 is listening on a symbolic luname called LUNAME1. The name of the generic resource defined for VTAM for the lunames listened on by QM4 and QM5 is LUQSG1. The example assumes that these are already defined on your z/OS system and are available for use. To define them see “Defining yourself to the network using generic resources” on page 289.

In this example QSG1 is the name of a queue-sharing group, and queue managers QM4 and QM5 are the names of members of the group.

Queue-sharing group definitions

Producing the following object definitions for one member of the queue-sharing group makes them available to all the other members.

Queue managers QM4 and QM5 are members of the queue sharing group. The definitions produced for QM4 are also available for QM5.

It is assumed that the coupling facility list structure is called 'APPLICATION1'. If it is not called 'APPLICATION1', you must use your own coupling facility list structure name for the example.

The shared object definitions are stored in DB2 and their associated messages are stored within the Coupling Facility.

```
DEFINE QLOCAL(PAYROLL) QSGDISP(SHARED) REPLACE PUT(ENABLED) GET(ENABLED) +
CFSTRUCT(APPLICATION1) +
DESCR('Shared queue for payroll details')
```

```
DEFINE QLOCAL(QM3) QSGDISP(SHARED) REPLACE USAGE(XMITQ) PUT(ENABLED) +
CFSTRUCT(APPLICATION1) +
DESCR('Transmission queue to QM3') TRIGGER TRIGTYPE(FIRST) +
TRIGDATA(QSG1.TO.QM3) GET(ENABLED) INITQ(SYSTEM.CHANNEL.INITQ)
```

Shared objects:

Group objects:

The group object definitions are stored in DB2, and each queue manager in the queue-sharing group creates a local copy of the defined object.

Sender channel definition:

For a TCP/IP connection:

```
DEFINE CHANNEL(QSG1.TO.QM3) CHLTYPE(SDR) QSGDISP(GROUP) TRPTYPE(TCP) +
REPLACE DESCR('Sender channel to QM3') XMITQ(QM3) +
CONNNAME('9.20.9.31(1411)')
```

For an LU6.2 connection:

```
DEFINE CHANNEL(QSG1.TO.QM3) CHLTYPE(SDR) QSGDISP(GROUP) TRPTYPE(LU62) +
REPLACE DESCR('Sender channel to QM3') XMITQ(QM3) +
CONNNAME('LUNAME1')
```

Receiver channel definition:

For a TCP/IP connection:

```
DEFINE CHANNEL(QM3.TO.QSG1) CHLTYPE(RCVR) TRPTYPE(TCP) +
REPLACE DESCR('Receiver channel from QM3') QSGDISP(GROUP)
```

For an LU 6.2 connection:

```
DEFINE CHANNEL(QM3.TO.QSG1) CHLTYPE(RCVR) TRPTYPE(LU62) +
REPLACE DESCR('Receiver channel from QM3') QSGDISP(GROUP)
```

Queue manager QM3 example

QM3 is not a member of the queue-sharing group.

The following object definitions allow it to put messages to a queue in the queue-sharing group.

Sender channel definition:

The conname for this channel is the generic address of the queue-sharing group.

For a TCP/IP connection:

```
DEFINE CHANNEL(QM3.TO.QSG1) CHLTYPE(SDR) TRPTYPE(TCP) +  
REPLACE DESCR('Sender channel to QSG1') XMITQ(QSG1) +  
CONNAME('QSG1.MVSIP(1414)')
```

For an LU 6.2 connection:

```
DEFINE CHANNEL(QM3.TO.QSG1) CHLTYPE(SDR) TRPTYPE(LU62) +  
REPLACE DESCR('Sender channel to QSG1') XMITQ(QSG1) +  
CONNAME('LUQSG1') TPNAME('MQSERIES') MODENAME('#INTER')
```

Remaining definitions

These definitions are required for the same purposes as those in the first example.

```
DEFINE QREMOTE(PAYROLL.QUERY) DESCR('Remote queue for QSG1') REPLACE +  
PUT(ENABLED) XMITQ(QSG1) RNAME(APPL) RQMNAME(QSG1)
```

```
DEFINE QLOCAL(QSG1) DESCR('Transmission queue to QSG1') REPLACE +  
USAGE(XMITQ) PUT(ENABLED) GET(ENABLED) TRIGGER TRIGTYPE(FIRST) +  
TRIGDATA(QM3.TO.QSG1) INITQ(SYSTEM.CHANNEL.INITQ)
```

```
DEFINE CHANNEL(QSG1.TO.QM3) CHLTYPE(RCVR) TRPTYPE(TCP) +  
REPLACE DESCR('Receiver channel from QSG1')
```

```
DEFINE CHANNEL(QSG1.TO.QM3) CHLTYPE(RCVR) TRPTYPE(LU62) +  
REPLACE DESCR('Receiver channel from QSG1')
```

```
DEFINE QLOCAL(PAYROLL.REPLY) REPLACE PUT(ENABLED) GET(ENABLED) +  
DESCR('Reply queue for replies to query messages sent to QSG1')
```

Running the example

When you have created the required objects, you must do the following.

- Start the channel initiator for all three queue managers.
- Start the listeners for both queue managers in the queue-sharing group.

For a TCP/IP connection each member of the group must have a group listener started that is listening on port 1414.

```
STA LSTR PORT(1414) IPADDR(MVSIP01) INDISP(GROUP)
```

The above entry starts the listener on QM4, for example.

For an LU6.2 connection each member of the group must have a group listener started that is listening on a symbolic luname that corresponds to the generic resource LUQSG1.

- Start the listener on QM3

```
STA LSTR PORT(1411)
```

Intra-group queuing

This chapter describes intra-group queuing on WebSphere MQ for z/OS. This function is only available to queue managers defined to a queue-sharing group.

For information about queue-sharing groups, see “Preparing WebSphere MQ for z/OS for DQM with queue-sharing groups” on page 280.

Concepts

This section describes the concepts of intra-group queuing.

Intra-group queuing (IGQ) can effect potentially fast and less-expensive small message transfer between queue managers within a queue-sharing group (QSG), without the need to define channels between the queue managers. That is, intra-group queuing can be used to deliver, more efficiently, small messages to queues residing on remote queue managers within a queue-sharing group.

The following diagram shows a typical example of intra-group queuing.

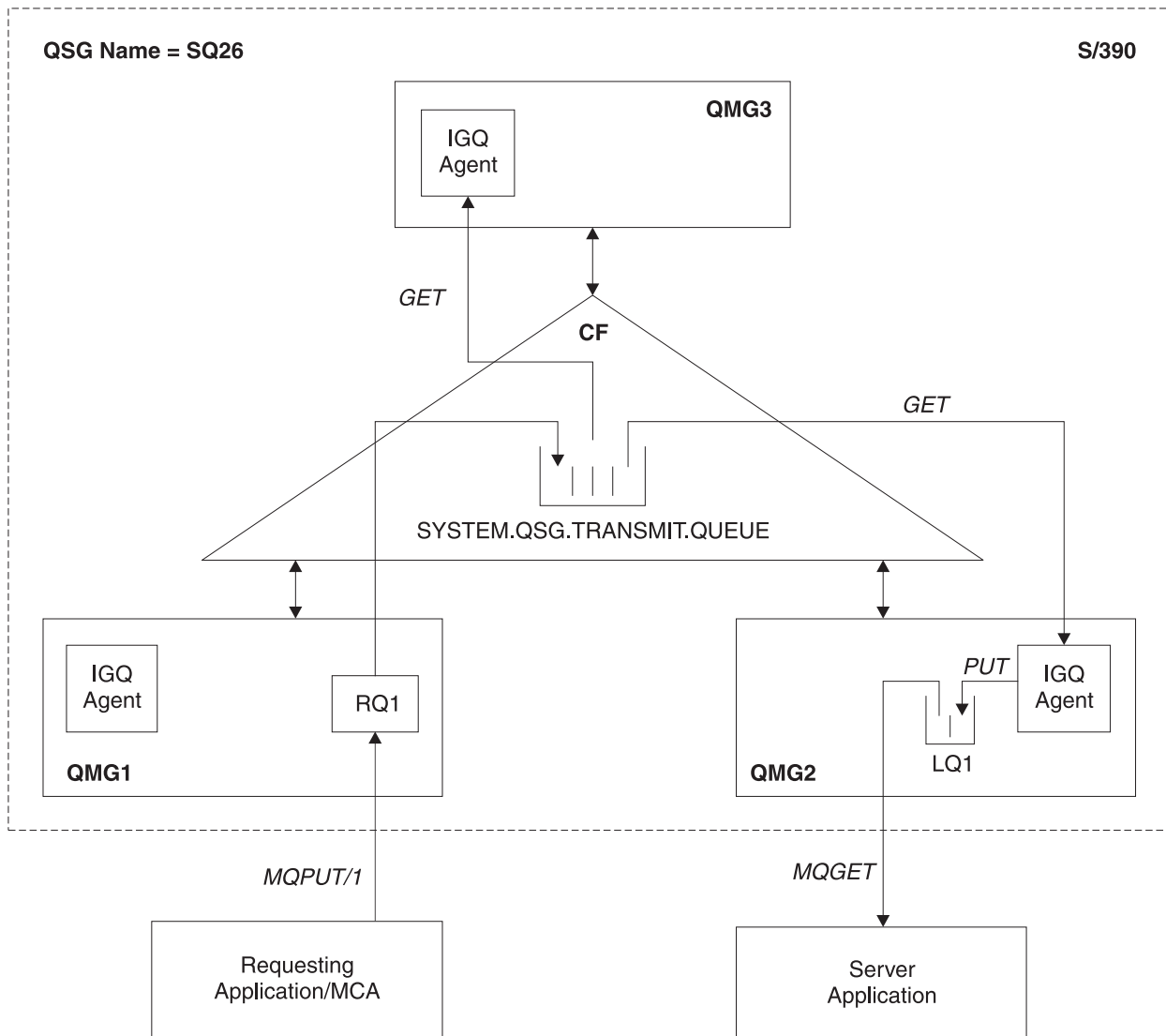


Figure 48. An example of intra-group queuing

The diagram shows:

- IGQ agents running on three queue managers (QMG1, QMG2 and QMG3) that are defined to a queue-sharing group called SQ26.
- Shared transmission queue SYSTEM.QSG.TRANSMIT.QUEUE that is defined in the Coupling Facility (CF).
- A remote queue definition that is defined in queue manager QMG1.
- A local queue that is defined in queue manager QMG2.
- A requesting application (this could be a Message Channel Agent (MCA)) that is connected to queue manager QMG1.
- A server application that is connected to queue manager QMG2.
- A request message being placed on to the SYSTEM.QSG.TRANSMIT.QUEUE.

Intra-group queuing and the intra-group queuing agent

An IGQ agent is started during queue manager initialization. When applications open and put messages to remote queues, the local queue manager determines whether intra-group queuing should be used for message transfer. If intra-group

queuing is to be used, the local queue manager places the message on to the SYSTEM.QSG.TRANSMIT.QUEUE. The IGQ agent on the target remote queue manager retrieves the message and places it on to the destination queue.

Terminology

This section describes the terminology of intra-group queuing.

Intra-group queuing

Intra-group queuing can effect potentially fast and less expensive message transfer between queue managers in a queue-sharing group, without the need to define channels.

Shared transmission queue for use by intra-group queuing

Each queue-sharing group has a shared transmission queue called SYSTEM.QSG.TRANSMIT.QUEUE for use by intra-group queuing. If intra-group queuing is enabled, SYSTEM.QSG.TRANSMIT.QUEUE appears in the name resolution path when opening remote queues. When applications (including Message Channel Agents (MCAs)) put messages to a remote queue, the local queue manager determines the eligibility of messages for fast transfer and places them on SYSTEM.QSG.TRANSMIT.QUEUE.

Intra-group queuing agent

The IGQ agent is the task, started at queue manager initialization, that waits for suitable messages to arrive on the SYSTEM.QSG.TRANSMIT.QUEUE. The IGQ agent retrieves suitable messages from this queue and delivers them to the destination queue(s).

The IGQ agent for each queue manager is always started because intra-group queuing is used by the queue manager itself for its own internal processing.

Benefits

This section describes the benefits of intra-group queuing.

Reduced system definitions

Intra-group queuing removes the need to define channels between queue managers in a queue-sharing group.

Reduced system administration

Because there are no channels defined between queue managers in a queue-sharing group, there is no requirement for channel administration.

Improved performance

Because there is only one IGQ agent needed for the delivery of a message to a target queue (instead of two intermediate sender and receiver agents), the delivery of messages using intra-group queuing can be less expensive than the delivery of messages using channels. In intra-group queuing there is only a receiving component, because the need for the sending component has been removed. This saving is because the message is available to the IGQ agent at the destination

queue manager for delivery to the destination queue once the put operation at the local queue manager has completed and, in the case of messages put in syncpoint scope, committed.

Supports migration

Applications external to a queue-sharing group can deliver messages to a queue residing on any queue manager in the queue-sharing group, while being connected only to a particular queue manager in the queue-sharing group. This is because messages arriving on a receiver channel, destined for a queue on a remote queue manager, can be transparently sent to the destination queue using intra-group queuing. This facility allows applications to be deployed among the queue-sharing group without the need to change any systems that are external to the queue-sharing group.

A typical configuration is illustrated by the following diagram, in which:

- A requesting application connected to queue manager QMG1 needs to send a message to a local queue on queue manager QMG3.
- Queue manager QMG1 is connected only to queue manager QMG2.
- Queue managers QMG2 and QMG3, which were previously connected using channels, are now members of queue-sharing group SQ26.

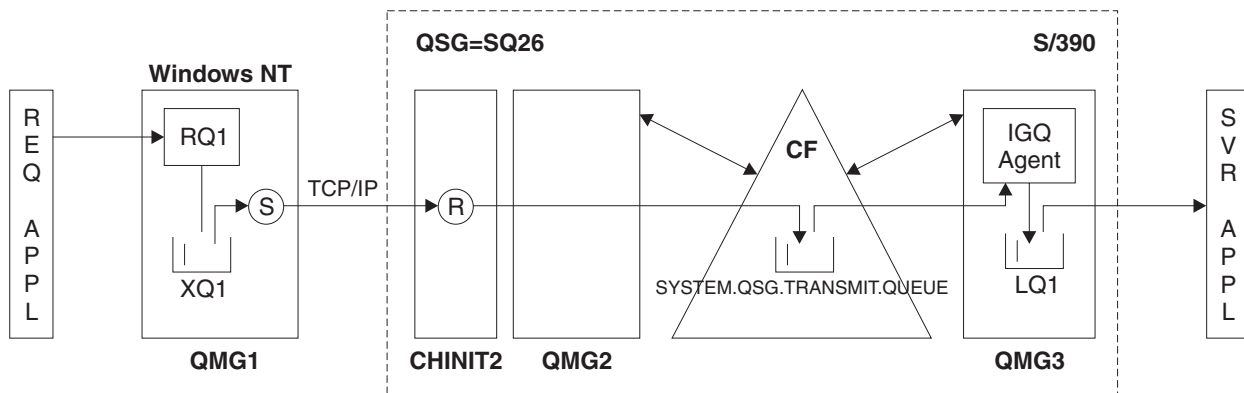


Figure 49. An example of migration support

The flow of operations is as follows:

1. The requesting application puts a message, destined for local queue LQ1 at remote queue manager QMG3, on to remote queue definition RQ1.
2. Queue manager QMG1, running on a Windows NT workstation, places the message on to the transmission queue XQ1.
3. Sender MCA (S) on QM1 transmits the message, via TCP/IP, to the receiver MCA (R) on channel initiator CHINIT2.
4. Receiver MCA (R) on channel initiator CHINIT2 places the message on to the shared transmission queue SYSTEM.QSG.TRANSMIT.QUEUE.
5. IGQ agent on queue manager QMG3 retrieves the message from the SYSTEM.QSG.TRANSMIT.QUEUE and places it on to the target local queue LQ1.
6. The server application retrieves the message from the target local queue and processes it.

Transparent delivery of messages when multi-hopping between queue managers in a queue-sharing group

The above diagram also illustrates the transparent delivery of messages when multi-hopping between queue managers in a queue-sharing group. Messages arriving on a given queue manager within the queue-sharing group, but destined for a queue on another queue manager in the queue-sharing group, can be easily transmitted to the destination queue on the destination queue manager, using intra-group queuing.

Limitations

This section describes the limitations of intra-group queuing.

Messages eligible for transfer using intra-group queuing

Because intra-group queuing makes use of a shared transmission queue that is defined in the Coupling Facility (CF), intra-group queuing is limited to delivering messages of the maximum supported message length for shared queues minus the length of a transmission queue header (MQXQH).

Number of intra-group queuing agents per queue manager

Only one IGQ agent is started per queue manager in a queue-sharing group.

Starting and stopping the intra-group queuing agent

The IGQ agent is started during queue manager initialization and terminated during queue manager shut-down. It is designed to be a long running, self recovering (in the event of abnormal termination), task. If there is an error with the definition of the SYSTEM.QSG.TRANSMIT.QUEUE (for example, if this queue is Get inhibited) the IGQ agent keeps retrying. If the IGQ agent should encounter an error that results in normal termination of the agent while the queue manager is still active, it can be restarted by issuing an ALTER QMGR IGQ(ENABLED) command. This avoids the need to recycle the queue manager.

Getting started

This section describes getting started with intra-group queuing.

Enabling intra-group queuing

To enable intra-group queuing on your queue managers, you need to do the following :

- Define a shared transmission queue called SYSTEM.QSG.TRANSMIT.QUEUE. The definition of this queue can be found in thlqual.SCSQPROCS(CSQ4INSS), the CSQINP2 sample for SYSTEM objects for queue-sharing groups. This queue must be defined with the correct attributes, as stated in thlqual.SCSQPROCS(CSQ4INSS), for intra-group queuing to work properly.
- Because the IGQ agent is always started at queue manager initialization, intra-group queuing is always available for inbound message processing, that is, the IGQ agent processes any messages that are placed on the SYSTEM.QSG.TRANSMIT.QUEUE. However, to enable intra-group queuing for outbound processing, the queue manager attribute IGQ must be set to ENABLED.

Disabling intra-group queuing

To disable intra-group queuing for outbound message transfer, set the queue manager attribute IGQ to DISABLED. Note that if intra-group queuing is disabled for a given queue manager, the IGQ agent on that queue manager can still process inbound messages that have been placed on the SYSTEM.QSG.TRANSMIT.QUEUE by a queue manager that does have intra-group queuing enabled for outbound transfer.

Using intra-group queuing

Once intra-group queuing is enabled, it is available for use and a queue manager uses it whenever possible. That is, when an application puts a message to a remote queue definition, to a fully qualified remote queue, or to a cluster queue, the queue manager determines if the message is eligible to be delivered using intra-group queuing and if it is, places the message on to SYSTEM.QSG.TRANSMIT.QUEUE. There is no need to make changes to user applications, or to application queues, because for eligible messages the queue manager makes use of the SYSTEM.QSG.TRANSMIT.QUEUE, in preference to any other transmission queue.

Configurations

In addition to the typical intra-group queuing configuration described in Figure 48 on page 299, other configurations are possible.

Distributed queuing with intra-group queuing (multiple delivery paths)

For applications that process short messages it might be feasible to configure only intra-group queuing for delivering messages between queue managers in a queue-sharing group. However, for applications that process large (greater than the maximum supported message length for shared queues minus the length of the MQXQH) messages, it may be necessary to configure distributed queuing with intra-group queuing. The following diagram illustrates this configuration.

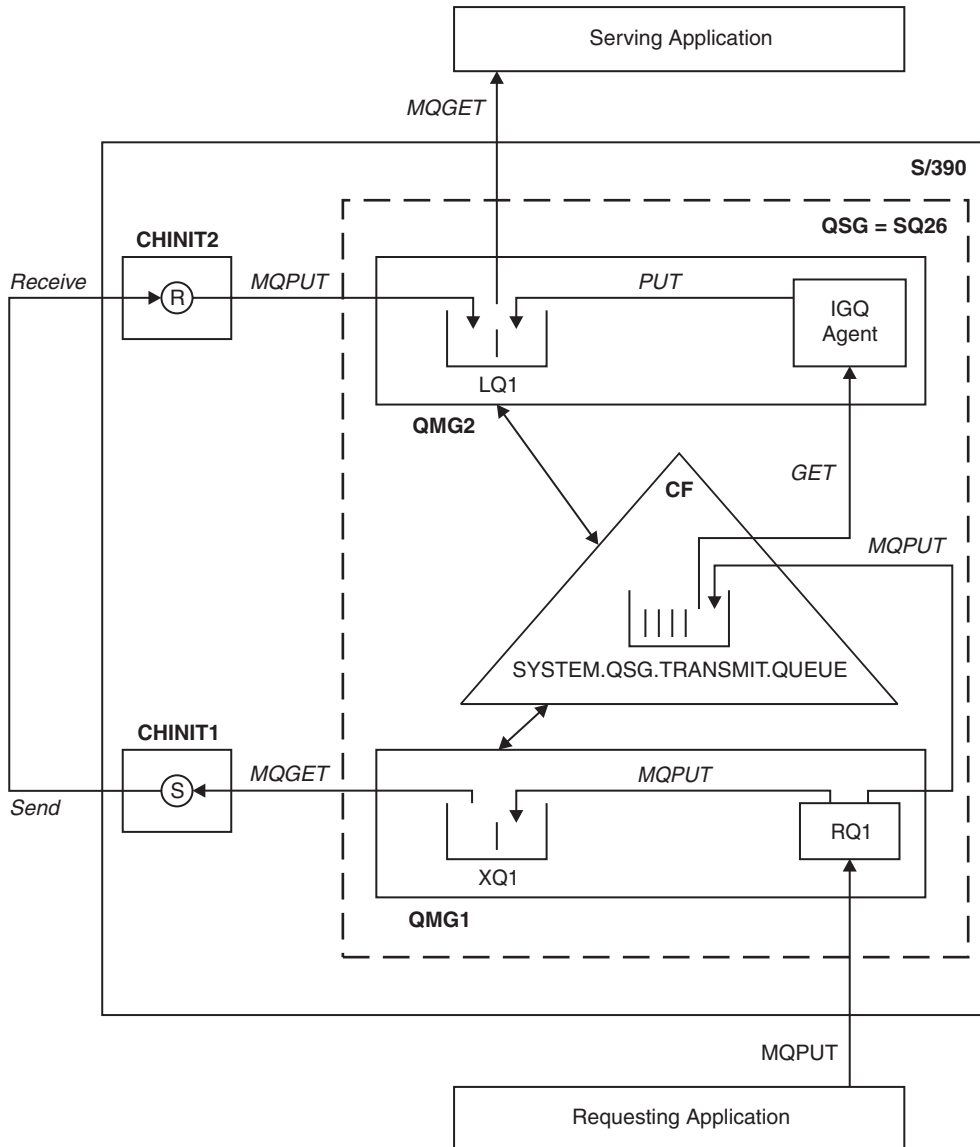


Figure 50. An example configuration

Open/Put processing:

1. It is important to note that when the requesting application opens remote queue RQ1, name resolution occurs for both the non-shared transmission queue XQ1 and the shared transmission queue SYSTEM.QSG.TRANSMIT.QUEUE.
2. When the requesting application puts a message on to the remote queue, the queue manager decides, based on whether intra-group queuing is enabled for outbound transfer and on the message characteristics, whether to put the message to transmission queue XQ1, or to transmission queue SYSTEM.QSG.TRANSMIT.QUEUE. The queue manager places all large messages on to transmission queue XQ1, and all small messages on to transmission queue SYSTEM.QSG.TRANSMIT.QUEUE.
3. If transmission queue XQ1 is full, or is not available, put requests for large messages fail synchronously with a suitable return and reason code. However, put requests for small messages continue to succeed and are placed on transmission queue SYSTEM.QSG.TRANSMIT.QUEUE.

4. If transmission queue `SYSTEM.QSG.TRANSMIT.QUEUE` is full, or cannot be put to, put requests for small messages fail synchronously with a suitable return and reason code. However, put requests for large messages continue to succeed and are placed on transmission queue `XQ1`. In this case, no attempt is made to put the small messages on to a transmission queue.

Flow for large messages:

1. The requesting application puts large messages to remote queue `RQ1`.
2. Queue manager `QMG1` puts the messages on to transmission queue `XQ1`.
3. Sender MCA (S) on queue manager `QMG1` retrieves the messages from transmission queue `XQ1` and sends them to queue manager `QMG2`.
4. Receiver MCA (R) on queue manager `QMG2` receives the messages and places them on to destination queue `LQ1`.
5. The serving application retrieves and subsequently processes the messages from queue `LQ1`.

Flow for small messages:

1. The requesting application puts small messages on to remote queue `RQ1`.
2. Queue manager `QMG1` puts the messages on to transmission queue `SYSTEM.QSG.TRANSMIT.QUEUE`.
3. IGQ on queue manager `QMG2` retrieves the messages and places them on to the destination queue `LQ1`.
4. The serving application retrieves the messages from queue `LQ1`.

Points to note about such a configuration:

1. The requesting application does not need to be aware of the underlying mechanism used for the delivery of messages.
2. A potentially faster message delivery mechanism can be achieved for small messages.
3. Multiple paths are available for message delivery (that is, the normal channel route and the intra-group queuing route).
4. The intra-group queuing route, being potentially faster, is selected in preference to the normal channel route. Depending on the message characteristics, message delivery may be divided across the two paths. Hence, messages may be delivered out of sequence (though this is also possible if messages are delivered using only the normal channel route).
5. Once a route has been selected, and messages have been placed on to the respective transmission queues, only the selected route will be used for message delivery. Any unprocessed messages on the `SYSTEM.QSG.TRANSMIT.QUEUE` are not diverted to transmission queue `XQ1`.

Clustering with intra-group queuing (multiple delivery paths)

It is possible to configure queue managers so that they are in a cluster as well as in a queue-sharing group. When messages are sent to a cluster queue and the local and remote destination queue managers are in the same queue-sharing group, intra-group queuing is used for the delivery of small messages (using the `SYSTEM.QSG.TRANSMIT.QUEUE`), while the `SYSTEM.CLUSTER.TRANSMIT.QUEUE` is used for the delivery of large messages. Also, the `SYSTEM.CLUSTER.TRANSMIT.QUEUE` is used for the delivery of messages to any queue manager that is in the cluster, but outside the queue-sharing group. The following diagram illustrates this configuration (the

channel initiators are not shown).

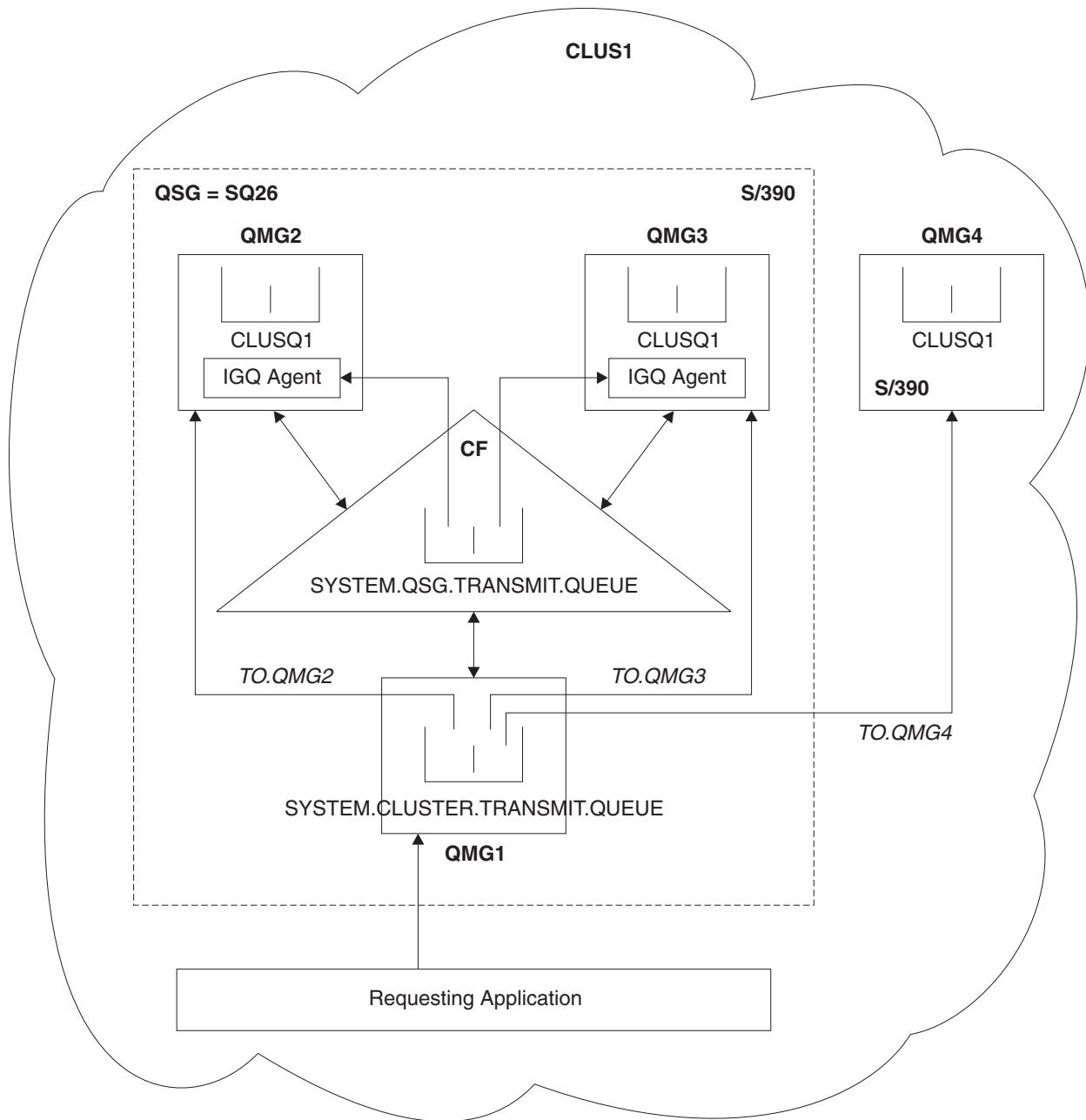


Figure 51. An example of clustering with intra-group queuing

The diagram shows:

- Four z/OS queue managers QMG1, QMG2, QMG3 and QMG4 configured in a cluster CLUS1.
- Queue managers QMG1, QMG2 and QMG3 configured in a queue-sharing group SQ26.
- IGQ agents running on queue managers QMG2 and QMG3.
- The local SYSTEM.CLUSTER.TRANSMIT.QUEUE defined in QMG1.
- The shared SYSTEM.QSG.TRANSMIT.QUEUE defined in the CF.

- Cluster channels TO.QMG2 (connecting QMG1 to QMG2), TO.QMG3 (connecting QMG1 to QMG3), and TO.QMG4 (connecting QMG1 to QMG4).
- Cluster queue CLUSQ1 being hosted on queue managers QMG2, QMG3 and QMG4.

Assume that the requesting application opens the cluster queue with the MQOO_BIND_NOT_FIXED option, so that the target queue manager for the cluster queue is selected at put time.

If the selected target queue manager is QMG2:

- All large messages put by the requesting application are
 - Put to the SYSTEM.CLUSTER.TRANSMIT.QUEUE on QMG1
 - Transferred to cluster queue CLUSQ1 on QMG2 using cluster channel TO.QMG2
- All small messages put by the requesting application are
 - Put to the shared transmission queue SYSTEM.QSG.TRANSMIT.QUEUE
 - Retrieved by the IGQ agent on QMG2
 - Put to the cluster queue CLUSQ1 on QMG2

If the selected target queue manager is QMG4:

- Because QMG4 is not a member of queue-sharing group SQ26, all messages put by the requesting application are
 - Put to the SYSTEM.CLUSTER.TRANSMIT.QUEUE on QMG1
 - Transferred to cluster queue CLUSQ1 on QMG4 using cluster channel TO.QMG4

Points to note about such a configuration:

- The requesting application does not need to be aware of the underlying mechanism used for the delivery of messages.
- A potentially faster delivery mechanism is achieved for the transfer of small non-persistent messages between queue managers in a queue-sharing group (even if the same queue managers are in a cluster).
- Multiple paths are available for message delivery (that is, both the cluster route and the intra-group queuing route).
- The intra-group queuing route, being potentially faster, is selected in preference to the cluster route. Depending on the message characteristics, message delivery may be divided across the two paths. Hence, messages may be delivered out of sequence. It is important to note that this will be true irrespective of the MQOO_BIND_* option specified by the application. Intra-group queuing distributes messages in the same way as clustering does, depending on whether the MQOO_BIND_NOT_FIXED, MQOO_BIND_ON_OPEN, or MQOO_BIND_AS_Q_DEF is specified on open.
- Once a route has been selected, and messages have been placed on to the respective transmission queues, only the selected route is used for message delivery. Any unprocessed messages on the SYSTEM.QSG.TRANSMIT.QUEUE are not diverted to the SYSTEM.CLUSTER.TRANSMIT.QUEUE.

Clustering, intra-group queuing and distributed queuing

It is possible to configure a queue manager that is a member of a cluster as well as a queue-sharing group and is connected to a distributed queue manager using a sender/receiver channel pair. This configuration is a combination of distributed

queuing with intra-group queuing, described in “Distributed queuing with intra-group queuing (multiple delivery paths)” on page 303, and clustering with intra-group queuing, described in “Clustering with intra-group queuing (multiple delivery paths)” on page 305.

Messages

This section describes the messages put to the SYSTEM.QSG.TRANSMIT.QUEUE.

Message structure

Like all other messages that are put to transmission queues, messages that are put to the SYSTEM.QSG.TRANSMIT.QUEUE are prefixed with the transmission queue header (MQXQH).

Message persistence

In WebSphere MQ Version 5 Release 3 and above, shared queues support both persistent and non-persistent messages.

If the queue manager terminates while the IGQ agent is processing non-persistent messages, or if the IGQ agent terminates abnormally while in the middle of processing messages, non-persistent messages being processed may be lost. Applications must make arrangements for the recovery of non-persistent messages if their recovery is required.

If a put request for a non-persistent message, issued by the IGQ agent, fails unexpectedly, the message being processed is lost.

Delivery of messages

The IGQ agent retrieves and delivers all nonpersistent messages outside of syncpoint scope, and all persistent messages within syncpoint scope. In this case, the IGQ agent acts as the syncpoint coordinator. The IGQ agent therefore processes nonpersistent messages in a way similar to the way fast, nonpersistent messages are processed on a message channel. See “Fast, nonpersistent messages” on page 20.

Batching of messages

The IGQ agent uses a fixed batchsize of 50 messages. Any persistent messages retrieved within a batch will be committed at intervals of 50 messages. The agent commits a batch consisting of persistent messages when there are no more messages available for retrieval on the SYSTEM.QSG.TRANSMIT.QUEUE.

Message size

The maximum size of message that can be put to the SYSTEM.QSG.TRANSMIT.QUEUE is the maximum supported message length for shared queues minus the length of a transmission queue header (MQXQH).

Default message persistence and default message priority

If the SYSTEM.QSG.TRANSMIT.QUEUE is in the queue name resolution path established at open time, then for messages that are put with default persistence and default priority (or with default persistence or default priority), the usual rules are applied in the selection of the queue whose default priority and persistence

values are used. (Refer to WebSphere MQ Application Programming Reference for information on the rules of queue selection).

Undelivered/unprocessed messages

If an IGQ agent cannot deliver a message to the destination queue, the IGQ agent:

- Honours the MQRO_DISCARD_MSG report option (if the Report options field of the MQMD for the undelivered message indicates that it should) and discards the undelivered message.
- Attempts to place the undelivered message on to the dead letter queue for the destination queue manager, if the message has not already been discarded. The IGQ agent prefixes the message with a dead letter queue header (MQDLH).

If a dead letter queue is not defined, or if an undelivered message cannot be put to the dead letter queue, and if the undelivered message is:

- persistent, the IGQ agent backs out the current batch of persistent messages that it is processing, and enters a state of retry. For more information, see “Retry capability of the intra-group queuing agent” on page 311.
- non-persistent, the IGQ agent discards the message and continues to process the next message.

If a queue manager in a queue-sharing group is terminated before its associated IGQ agent has had time to process all its messages, the unprocessed messages remain on the SYSTEM.QSG.TRANSMIT.QUEUE until the queue manager is next started. The IGQ agent then retrieves and delivers the messages to the destination queues.

If the coupling facility fails before all the messages on the SYSTEM.QSG.TRANSMIT.QUEUE have been processed, any unprocessed non-persistent messages will be lost.

IBM recommends that applications do not put messages directly to transmission queues. If an application does put messages directly to the SYSTEM.QSG.TRANSMIT.QUEUE, the IGQ agent may not be able to process these messages and they will simply remain on the SYSTEM.QSG.TRANSMIT.QUEUE. Users will then have to use their own methods to deal with these unprocessed messages.

Report messages

This section describes report messages.

Confirmation of arrival (COA)/confirmation of delivery (COD) report messages:

COA and COD messages are generated by the queue manager, when intra-group queuing is used.

Expiry report messages:

Expiry report messages are generated by the queue manager, when intra-group queuing is used.

Exception report messages:

Depending on the MQRO_EXCEPTION_* report option specified in the *Report options* field of the message descriptor for the undelivered message, the IGQ agent

generates the required exception report and places it on the specified reply-to queue. (Note that intra-group queuing can be used to deliver the exception report to the destination reply-to queue).

The persistence of the report message is the same as the persistence of the undelivered message. If the IGQ agent fails to resolve the name of the destination reply-to queue, or if it fails to put the reply message to a transmission queue (for subsequent transfer to the destination reply-to queue) it attempts to put the exception report to the dead letter queue of the queue manager on which the report message is generated. If this is not possible, then if the undelivered message is:

- persistent, the IGQ agent discards the exception report, backs out the current batch of messages and enters a state of retry. For more information, see “Retry capability of the intra-group queuing agent” on page 311.
- non-persistent, the IGQ agent discards the exception report and continues processing the next message on the `SYSTEM.QSG.TRANSMIT.QUEUE`.

Security

This section describes the security arrangements for intra-group queuing.

Queue manager attributes `IGQAUT` (IGQ authority) and `IGQUSER` (IGQ agent user ID) can be set to control the level of security checking that is performed when the IGQ agent opens destination queues.

Intra-group queuing authority (IGQAUT)

The `IGQAUT` attribute can be set to indicate the type of security checks to be performed, and hence to determine the userids to be used by the IGQ agent when it establishes the authority to put messages on to the destination queue.

The `IGQAUT` attribute is analogous to the `PUTAUT` attribute that is available on channel definitions.

Intra-group queuing user identifier (IGQUSER)

The `IGQUSER` attribute can be used to nominate a user ID to be used by the IGQ agent when it establishes the authority to put messages on to a destination queue.

The `IGQUSER` attribute is analogous to the `MCAUSER` attribute that is available on channel definitions.

Refer to WebSphere MQ Script (MQSC) Command Reference and WebSphere MQ Application Programming Reference for more information about the `IGQAUT` and `IGQUSER` queue manager attributes. Refer to the chapter on security in WebSphere MQ for z/OS System Setup Guide for a table of the userids checked for intra-group queuing.

Specific properties

This section describes the specific properties of intra-group queuing.

Queue name resolution

Refer to WebSphere MQ Script (MQSC) Command Reference, WebSphere MQ Application Programming Reference and WebSphere MQ for z/OS System Setup Guide for details of queue name resolution when intra-group queuing is used.

Invalidation of object handles (MQRC_OBJECT_CHANGED)

If the attributes of an object are found to have changed after the object is opened, the queue manager invalidates the object handle with MQRC_OBJECT_CHANGED on its next use.

Intra-group queuing introduces the following new rules for object handle invalidation :

- If the SYSTEM.QSG.TRANSMIT.QUEUE was included in the name resolution path during open processing because intra-group queuing was ENABLED at open time, but intra-group queuing is found to be DISABLED at put time, then the queue manager invalidates the object handle and fails the put request with MQRC_OBJECT_CHANGED.
- If the SYSTEM.QSG.TRANSMIT.QUEUE was not included in the name resolution path during open processing because intra-group queuing was DISABLED at open time, but intra-group queuing is found to be ENABLED at put time, then the queue manager invalidates the object handle and fails the put request with MQRC_OBJECT_CHANGED.
- If the SYSTEM.QSG.TRANSMIT.QUEUE was included in the name resolution path during open processing because intra-group queuing was enabled at open time, but the SYSTEM.QSG.TRANSMIT.QUEUE definition is found to have changed by put time, then the queue manager invalidates the object handle and fails the put request with MQRC_OBJECT_CHANGED.

Self recovery of the intra-group queuing agent

In the event that the IGQ agent terminates abnormally, message CSQM067E is issued and the IGQ agent performs self recovery.

Retry capability of the intra-group queuing agent

In the event that the IGQ agent encounters a problem accessing the SYSTEM.QSG.TRANSMIT.QUEUE (because it is not defined, for example, or is defined with incorrect attributes, or is inhibited for Gets, or for some other reason), the IGQ agent goes into the state of retry. The IGQ agent observes short and long retry counts and intervals. The values for these counts and intervals, which cannot be changed, are as follows:

Constant	Value
Short retry count	10
Short retry interval	60 secs = 1 min
Long retry count	999,999,999
Long retry interval	1200 secs = 20 min

The intra-group queuing agent and Serialization

If there is a failure of a queue manager in a queue-sharing group while the IGQ agent is dealing with uncommitted messages on a shared queue or queues, the

IGQ agent ends, and shared queue peer recovery takes place for the failing queue manager. Because shared queue peer recovery is an asynchronous activity, this leaves the possibility for the failing queue manager, and also the IGQ agent for that queue manager, to restart before shared queue peer recovery is complete. This in turn leaves the possibility for any committed messages to be processed ahead of and out of sequence with the messages still being recovered. To ensure that messages are not processed out of sequence, the IGQ agent serializes access to shared queues by issuing the MQCONN API call.

An attempt, by the IGQ agent to serialize access to shared queues while peer recovery is still in progress might fail. An error message is issued and the IGQ agent is put into retry state. When queue manager peer recovery is complete, for example at the time of the next retry, the IGQ agent can start.

Example configuration — WebSphere MQ for z/OS using intra-group queuing

This section describes how a typical payroll query application, that currently uses distributed queuing to transfer small messages between queue managers, could be migrated to exploit queue sharing groups and shared queues.

Three configurations are described to illustrate the use of distributed queuing, intra-group queuing with shared queues, and shared queues. The associated diagrams show only the flow of data in one direction, that is, from queue manager QMG1 to queue manager QMG3.

Configuration 2 describes how queue-sharing groups and intra-group queuing can be used, with no effect on the back end payroll server application, to transfer messages between queue managers QMG1 and QMG3. This configuration removes the need for channel definitions between queue managers QMG2 and QMG3 because intra-group queuing is used to transfer messages between these two queue managers.

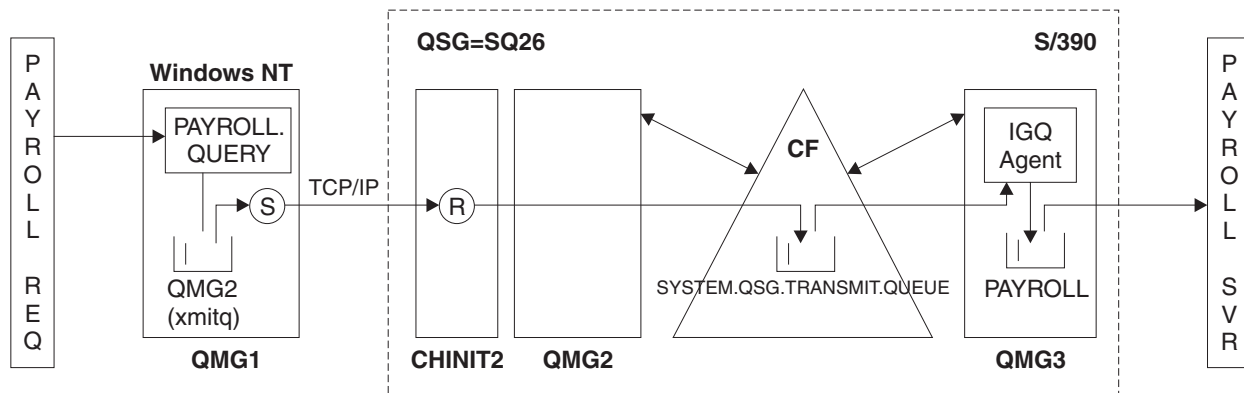


Figure 52. Configuration 2

The flow of operations is as follows:

1. A query is entered using the payroll request application connected to queue manager QMG1.
2. The payroll request application puts the query on to remote queue PAYROLL.QUERY. As queue PAYROLL.QUERY resolves to transmission queue QMG2, the query is put on to transmission queue QMG2.

3. Sender channel (S) on queue manager QMG1 delivers the query to the partner receiver channel (R) on queue manager QMG2.
4. Receiver channel (R) on queue manager QMG2 puts the query on to queue PAYROLL on queue manager QMG3. As queue PAYROLL on QMG3 resolves to shared transmission queue SYSTEM.QSG.TRANSMIT.QUEUE, the query is put on to shared transmission queue SYSTEM.QSG.TRANSMIT.QUEUE.
5. IGQ agent on queue manager QMG3 retrieves the query from shared transmission queue SYSTEM.QSG.TRANSMIT.QUEUE, and puts it on to local queue PAYROLL on queue manager QMG3.
6. The payroll server application connected to queue manager QMG3 retrieves the query from local queue PAYROLL, processes it, and generates a suitable reply.

Note: The payroll query example transfers small messages only. If you need to transfer both persistent and non-persistent messages, a combination of Configuration 1 and Configuration 2 can be established, so that large messages can be transferred using the distributed queuing route, while small messages can be transferred using the potentially faster intra-group queuing route.

Configuration 1

Configuration 1 describes how distributed queuing is currently used to transfer messages between queue managers QMG1 and QMG3.

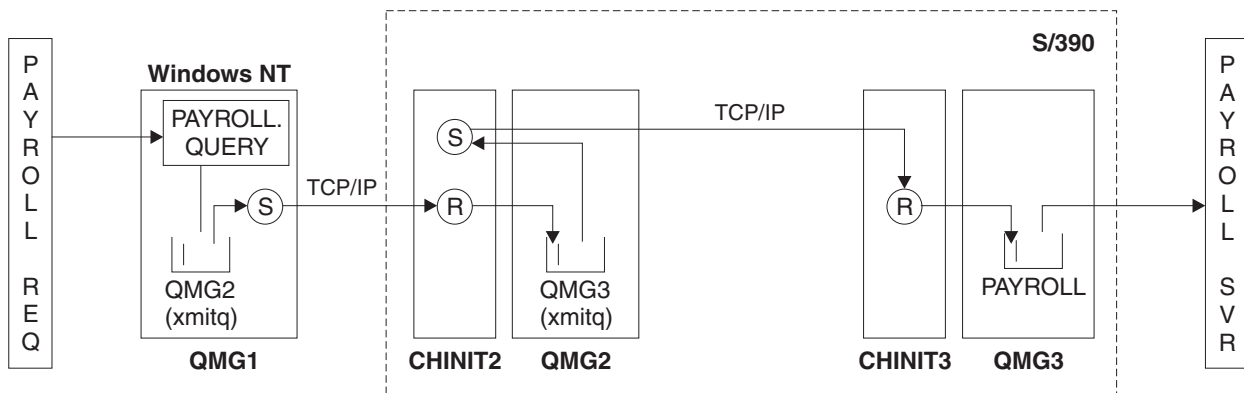


Figure 53. Configuration 1: z/OS using intra-group queuing

The flow of operations is as follows:

1. A query is entered using the payroll request application connected to queue manager QMG1.
2. The payroll request application puts the query on to remote queue PAYROLL.QUERY. As queue PAYROLL.QUERY resolves to transmission queue QMG2, the query is put on to transmission queue QMG2.
3. Sender channel (S) on queue manager QMG1 delivers the query to the partner receiver channel (R) on queue manager QMG2.
4. Receiver channel (R) on queue manager QMG2 puts the query on to queue PAYROLL on queue manager QMG3. As queue PAYROLL on QMG3 resolves to transmission queue QMG3, the query is put on to transmission queue QMG3.
5. Sender channel (S) on queue manager QMG2 delivers the query to the partner receiver channel (R) on queue manager QMG3.
6. Receiver channel (R) on queue manager QMG3 puts the query on to local queue PAYROLL.

- The payroll server application connected to queue manager QMG3 retrieves the query from local queue PAYROLL, processes it, and generates a suitable reply.

Configuration 2

Configuration 3

Configuration 3 describes how queue-sharing groups and shared queues can be used, with no effect on the backend payroll server application, to transfer messages between queue managers QMG1 and QMG3.

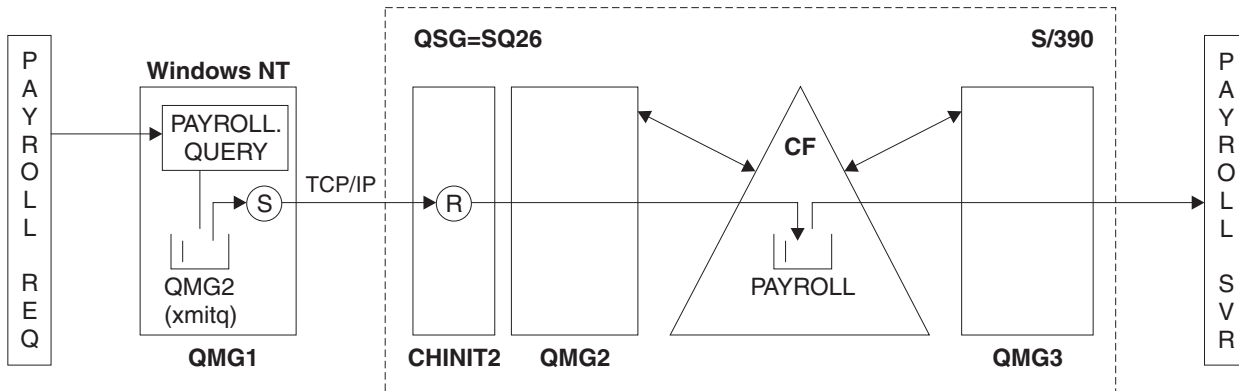


Figure 54. Configuration 3

The flow of operations is:

- A query is entered using the payroll request application connected to queue manager QMG1.
- The payroll request application puts the query on to remote queue PAYROLL.QUERY. As queue PAYROLL.QUERY resolves to transmission queue QMG2, the query is put on to transmission queue QMG2.
- Sender channel (S) on queue manager QMG1 delivers the query to the partner receiver channel (R) on queue manager QMG2.
- Receiver channel (R) on queue manager QMG2 puts the query on to shared queue PAYROLL.
- The payroll server application connected to queue manager QMG3 retrieves the query from shared queue PAYROLL, processes it, and generates a suitable reply.

This configuration is certainly the simplest to configure. However, it should be noted that distributed queuing or intra-group queuing would need to be configured to transfer replies (generated by the payroll server application connected to queue manager QMG3) from queue manager QMG3 to queue manager QMG2, and then on to queue manager QMG1. (See “What this example shows” on page 294 for the configuration used to transfer replies back to the payroll request application.)

No definitions are required on QMG3.

Configuration 1 definitions

The definitions required for Configuration 1 are as follows (please note that the definitions do not take into account triggering, and that only channel definitions for communication using TCP/IP are provided).

On QMG1:

Remote queue definition

```
DEFINE QREMOTE(PAYROLL.QUERY) DESCR('Remote queue for QMG3') REPLACE +  
PUT(ENABLED) RNAME(PAYROLL) RQMNAME(QMG3) XMITQ(QMG2)
```

Transmission queue definition

```
DEFINE QLOCAL(QMG2) DESCR('Transmission queue to QMG2') REPLACE +  
PUT(ENABLED) USAGE(XMITQ) GET(ENABLED)
```

Sender channel definition (for TCP/IP)

```
DEFINE CHANNEL(QMG1.TO.QMG2) CHLTYPE(SDR) TRPTYPE(TCP) REPLACE +  
DESCR('Sender channel to QMG2') XMITQ(QMG2) CONNAME('MVSQMG2(1415)')
```

This is the place where you should replace MVSQMG2(1415) with your queue manager connection name and port.

Receiver channel definition (for TCP/IP)

```
DEFINE CHANNEL(QMG2.TO.QMG1) CHLTYPE(RCVR) TRPTYPE(TCP) +  
REPLACE DESCR('Receiver channel from QMG2')
```

Reply-to queue definition

```
DEFINE QLOCAL(PAYROLL.REPLY) REPLACE PUT(ENABLED) GET(ENABLED) +  
DESCR('Reply queue for replies to payroll queries sent to QMG3')
```

On QMG2:

Transmission queue definition

```
DEFINE QLOCAL(QMG1) DESCR('Transmission queue to QMG1') REPLACE +  
PUT(ENABLED) USAGE(XMITQ) GET(ENABLED)
```

```
DEFINE QLOCAL(QMG3) DESCR('Transmission queue to QMG3') REPLACE +  
PUT(ENABLED) USAGE(XMITQ) GET(ENABLED)
```

Sender channel definitions (for TCP/IP)

```
DEFINE CHANNEL(QMG2.TO.QMG1) CHLTYPE(SDR) TRPTYPE(TCP) REPLACE +  
DESCR('Sender channel to QMG1') XMITQ(QMG1) CONNAME('WINTQMG1(1414)')
```

This is the place where you should replace WINTQMG1(1414) with your queue manager connection name and port.

```
DEFINE CHANNEL(QMG2.TO.QMG3) CHLTYPE(SDR) TRPTYPE(TCP) REPLACE +  
DESCR('Sender channel to QMG3') XMITQ(QMG3) CONNAME('MVSQMG3(1416)')
```

This is the place where you should replace MVSQMG3(1416) with your queue manager connection name and port.

Receiver channel definition (for TCP/IP)

```
DEFINE CHANNEL(QMG1.TO.QMG2) CHLTYPE(RCVR) TRPTYPE(TCP) +  
REPLACE DESCR('Receiver channel from QMG1')
```

```
DEFINE CHANNEL(QMG3.TO.QMG2) CHLTYPE(RCVR) TRPTYPE(TCP) +  
REPLACE DESCR('Receiver channel from QMG3')
```

On QMG3:

Local queue definition

```
DEFINE QLOCAL(PAYROLL) DESCR('Payroll query request queue') REPLACE +  
PUT(ENABLED) USAGE(NORMAL) GET(ENABLED) SHARE
```

```
DEFINE QLOCAL(QMG2) DESCR('Transmission queue to QMG2') REPLACE +  
PUT(ENABLED) USAGE(XMITQ) GET(ENABLED)
```

Sender channel definitions (for TCP/IP)

```
DEFINE CHANNEL(QMG3.TO.QMG2) CHLTYPE(SDR) TRPTYPE(TCP) REPLACE +  
DESCR('Sender channel to QMG2) XMITQ(QMG2) CONNAME('MVSQMG2(1415)')
```

This is the place where you should replace MVSQMG2(1415) with your queue manager connection name and port.

Receiver channel definition (for TCP/IP)

```
DEFINE CHANNEL(QMG2.TO.QMG3) CHLTYPE(RCVR) TRPTYPE(TCP) +  
REPLACE DESCR('Receiver channel from QMG2)
```

Configuration 2 definitions

The definitions required for Configuration 2 are as follows (please note that the definitions do not take into account triggering, and that only channel definitions for communication using TCP/IP are provided). It is assumed that queue managers QMG2 and QMG3 are already configured to be members of the same queue-sharing group.

On QMG1:

Remote queue definition

```
DEFINE QREMOTE(PAYROLL.QUERY) DESCR('Remote queue for QMG3') REPLACE +  
PUT(ENABLED) RNAME(PAYROLL) RQMNAME(QMG3) XMITQ(QMG2)
```

Transmission queue definition

```
DEFINE QLOCAL(QMG2) DESCR('Transmission queue to QMG2') REPLACE +  
PUT(ENABLED) USAGE(XMITQ) GET(ENABLED)
```

Sender channel definition (for TCP/IP)

```
DEFINE CHANNEL(QMG1.TO.QMG2) CHLTYPE(SDR) TRPTYPE(TCP) REPLACE +  
DESCR('Sender channel to QMG2') XMITQ(QMG2) CONNAME('MVSQMG2(1415)')
```

This is the place where you should replace MVSQMG2(1415) with your queue manager connection name and port.

Receiver channel definition (for TCP/IP)

```
DEFINE CHANNEL(QMG2.TO.QMG1) CHLTYPE(RCVR) TRPTYPE(TCP) +  
REPLACE DESCR('Receiver channel from QMG2')
```

Reply-to queue definition

```
DEFINE QLOCAL(PAYROLL.REPLY) REPLACE PUT(ENABLED) GET(ENABLED) +  
DESCR('Reply queue for replies to payroll queries sent to QMG3')
```


On QMG2:

Transmission queue definition

```
DEFINE QLOCAL(QMG1) DESCR('Transmission queue to QMG1') REPLACE +
PUT(ENABLED) USAGE(XMITQ) GET(ENABLED)

DEFINE QLOCAL(SYSTEM.QSG.TRANSMIT.QUEUE) QSGDISP(SHARED) +
DESCR('IGQ Transmission queue') REPLACE PUT(ENABLED) USAGE(XMITQ) +
GET(ENABLED) INDXTYPE(CORRELID) CFSTRUCT('APPLICATION1') +
DEFSOPT(SHARED) DEFPSIST(NO)
```

This is the place where you should replace APPLICATION1 with your defined CF structure name. Also note that this queue, being a shared queue, need only be defined on one of the queue managers in the queue sharing group.

Sender channel definitions (for TCP/IP)

```
DEFINE CHANNEL(QMG2.TO.QMG1) CHLTYPE(SDR) TRPTYPE(TCP) REPLACE +
DESCR('Sender channel to QMG1') XMITQ(QMG1) CONNAME('WINTQMG1(1414)')
```

This is the place where you should replace WINTQMG1(1414) with your queue manager connection name and port.

Receiver channel definition (for TCP/IP)

```
DEFINE CHANNEL(QMG1.TO.QMG2) CHLTYPE(RCVR) TRPTYPE(TCP) +
REPLACE DESCR('Receiver channel from QMG1')
```

Queue Manager definition

```
ALTER QMGR IGQ(ENABLED)
```

On QMG3:

Local queue definition

```
DEFINE QLOCAL(PAYROLL) DESCR('Payroll query request queue') REPLACE +
PUT(ENABLED) USAGE(NORMAL) GET(ENABLED) SHARE
```

Queue Manager definition

```
ALTER QMGR IGQ(ENABLED)
```

Configuration 3 definitions

The definitions required for Configuration 3 are as follows (please note that the definitions do not take into account triggering, and that only channel definitions for communication using TCP/IP are provided). It is assumed that queue managers QMG2 and QMG3 are already configured to be members of the same queue-sharing group.

On QMG1:

Remote queue definition

```
DEFINE QREMOTE(PAYROLL.QUERY) DESCR('Remote queue for QMG3') REPLACE +
PUT(ENABLED) RNAME(PAYROLL) RQMNAME(QMG3) XMITQ(QMG2)
```

Transmission queue definition

```
DEFINE QLOCAL(QMG2) DESCR('Transmission queue to QMG2') REPLACE +
PUT(ENABLED) USAGE(XMITQ) GET(ENABLED)
```

Sender channel definition (for TCP/IP)

```
DEFINE CHANNEL(QMG1.TO.QMG2) CHLTYPE(SDR) TRPTYPE(TCP) +
REPLACE DESCR('Sender channel to QMG2') XMITQ(QMG2) CONNAME('MVSQMG2(1415)')
```

This is the place where you should replace MVSQMG2(1415) with your queue manager connection name and port.

Receiver channel definition (for TCP/IP)

```
DEFINE CHANNEL(QMG2.TO.QMG1) CHLTYPE(RCVR) TRPTYPE(TCP) +
REPLACE DESCR('Receiver channel from QMG2')
```

Reply-to queue definition

```
DEFINE QLOCAL(PAYROLL.REPLY) REPLACE PUT(ENABLED) GET(ENABLED) +
DESCR('Reply queue for replies to payroll queries sent to QMG3')
```

On QMG2:

Transmission queue definition

```
DEFINE QLOCAL(QMG1) DESCR('Transmission queue to QMG1') REPLACE +
PUT(ENABLED) USAGE(XMITQ) GET(ENABLED)
```

Sender channel definitions (for TCP/IP)

```
DEFINE CHANNEL(QMG2.TO.QMG1) CHLTYPE(SDR) TRPTYPE(TCP) +
REPLACE DESCR('Sender channel to QMG1') XMITQ(QMG1) CONNAME('WINTQMG1(1414)')
```

This is the place where you should replace WINTQMG1(1414) with your queue manager connection name and port.

Receiver channel definition (for TCP/IP)

```
DEFINE CHANNEL(QMG1.TO.QMG2) CHLTYPE(RCVR) TRPTYPE(TCP) +
REPLACE DESCR('Receiver channel from QMG1')
```

Local queue definition

```
DEFINE QLOCAL(PAYROLL) QSGDISP(SHARED) DESCR('Payroll query request queue') +
REPLACE PUT(ENABLED) USAGE(NORMAL) GET(ENABLED) SHARE +
DEFSOPT(SHARED) DEFPSIST(NO) CFSTRUCT(APPLICATION1)
```

This is the place where you should replace APPLICATION1 with your defined CF structure name. Also note that this queue, being a shared queue, need only be defined on one of the queue managers in the queue sharing group.

On QMG3:

Running the example

For Configuration 1:

1. Start queue managers QMG1, QMG2 and QMG3.
2. Start channel initiators for QMG2 and QMG3.
3. Start the listeners on QMG1, QMG2 and QMG3 to listen on ports 1414, 1415 and 1416, respectively.
4. Start sender channel(s) on QMG1, QMG2 and QMG3.
5. Start the payroll query requesting application connected to QMG1.
6. Start the payroll server application connected to QMG3.
7. Submit a payroll query request to QMG3 and wait for the payroll reply.

For Configuration 2:

1. Start queue managers QMG1, QMG2, and QMG3.
2. Start the channel initiator for QMG2.
3. Start the listeners on QMG1, QMG2 to listen on ports 1414 and 1415, respectively.
4. Start the sender channel on QMG1 and QMG2.
5. Start the payroll query requesting application connected to QMG1.
6. Start the payroll server application connected to QMG3.
7. Submit a payroll query request to QMG3 and wait for the payroll reply.

For Configuration 3:

1. Start queue managers QMG1, QMG2 and QMG3.
2. Start the channel initiator for QMG2.
3. Start the listeners on QMG1, QMG2 to listen on ports 1414 and 1415, respectively.
4. Start sender channels on QMG1 and QMG2.
5. Start the payroll query requesting application connected to QMG1.
6. Start the payroll server application connected to QMG3.
7. Submit a payroll query request to QMG3 and wait for the payroll reply.

Expanding the example

The example can be:

- Expanded to make use of channel triggering as well as application (PAYROLL and PAYROLL.REPLY queue) triggering.
- Configured for communication using LU6.2.
- Expanded to configure more queue managers to the queue sharing group. Then the server application can be cloned to run on other queue manager instances to provide multiple servers for the PAYROLL query queue.
- Expanded to increase the number of instances of the payroll query requesting application to demonstrate the processing of requests from multiple clients.
- Expanded to make use of security (IGQAUT and IGQUSER).

Chapter 5. DQM in WebSphere MQ for i5/OS

Monitoring and controlling channels on i5/OS

Use the DQM commands and panels to create, monitor, and control the channels to remote queue managers. Each queue manager has a DQM program for controlling interconnections to compatible remote queue managers. See “Operator commands” for a list of the commands you need when setting up and controlling message channels.

DQM channel control

DQM channel control is achieved using WebSphere MQ for i5/OS panels, commands, programs, WebSphere MQ Explorer, a sequence number file, and files for the channel definitions. The following is a brief description of the components of the channel control function:

- Channel definition files are held as queue manager objects. For the location of the default definitions, see the System Administration Guide
 - /QIBM/UserData/mqm/qmgrs/./auth/channels
 - /QIBM/UserData/mqm/qmgrs/./auth/channels/&mangled
 - /QIBM/UserData/mqm/qmgrs/./auth/clntconn
 - /QIBM/UserData/mqm/qmgrs/./auth/clntconn/&mangled
 - /QIBM/UserData/mqm/qmgrs/./channels
 - /QIBM/UserData/mqm/qmgrs/./clntconn
- The channel commands are a subset of the WebSphere MQ for i5/OS set of commands.

Use the command GO CMDMQM to display the full set of WebSphere MQ for i5/OS commands.

- You use channel definition panels, or commands to:
 - Create, copy, display, change, and delete channel definitions
 - Start and stop channels, ping, reset channel sequence numbers, and resolve in-doubt messages when links cannot be re-established
 - Display status information about channels
- Channels can also be managed using MQSC
- Channels can also be managed using WebSphere MQ Explorer
- Sequence numbers and *logical unit of work (LUW)* identifiers are stored in the synchronization file, and are used for channel synchronization purposes.

Operator commands

The following table shows the full list of WebSphere MQ for i5/OS commands that you may need when setting up and controlling channels. In general, issuing a command results in the appropriate panel being displayed.

The commands can be grouped as follows:

- Channel commands
 - CHGMQMCHL, Change MQM Channel

- CPYMQMCHL, Copy MQM Channel
- CRTMQMCHL, Create MQM Channel
- DLTMQMCHL, Delete MQM Channel
- DSPMQMCHL, Display MQM Channel
- ENDMQMCHL, End MQM Channel
- ENDMQMMLSR, End MQM Listener
- PNGMQMCHL, Ping MQM Channel
- RSTMQMCHL, Reset MQM Channel
- RSVMQMCHL, Resolve MQM Channel
- STRMQMCHL, Start MQM Channel
- STRMQMCHLI, Start MQM Channel Initiator
- STRMQMMLSR, Start MQM Listener
- WRKMQMCHL, Work with MQM Channel
- WRKMQMCHST, Work with MQM Channel Status
- Cluster commands
 - RFRMQMCL, Refresh Cluster
 - RSMMQMCLQM, Resume Cluster Queue Manager
 - RSTMQMCL, Reset Cluster
 - SPDMQMCLQM, Suspend Cluster Queue Manager
 - WRKMQMCL, Work with Clusters
- Command Server commands
 - DSPMQMCSV, Display MQM Command Server
 - ENDMQMCSV, End MQM Command Server
 - STRMQMCSV, Start MQM Command Server
- Data Type Conversion Command
 - CVTMQMMDTA, Convert MQM Data Type Command
- Dead-Letter Queue Handler Command
 - STRMQMDLQ, Start WebSphere MQ Dead-Letter Queue Handler
- Media Recovery Commands
 - RCDMQMIMG, Record MQM Object Image
 - RCRMQMOBJ, Recreate MQM Object
- WebSphere MQ commands
 - STRMQMMQSC, Start MQSC Commands
- Name command
 - DSPMQMOBJN, Display MQM Object Names
- Namelist commands
 - CHGMQMNL, Change MQM Namelist
 - CPYMQMNL, Copy MQM Namelist
 - CRTMQMNL, Create MQM Namelist
 - DLTMQMNL, Delete MQM Namelist
 - DSPMQMNL, Display MQM Namelist
 - WRKMQMNL, Work with MQM Namelists
- Process commands
 - CHGMQMPC, Change MQM Process
 - CPYMQMPC, Copy MQM Process

- CRTMQMPRC, Create MQM Process
- DLTMQMPRC, Delete MQM Process
- DSPMQMPRC, Display MQM Process
- WRKMQMPRC, Work with MQM Processes
- Queue commands
 - CHGMQM, Change MQM Queue
 - CLRMQM, Clear MQM Queue
 - CPYMQM, Copy MQM Queue
 - CRTMQM, Create MQM Queue
 - DLTMQM, Delete MQM Queue
 - DSPMQM, Display MQM Queue
 - WRKMQMSG, Work with MQM Messages
 - WRKMQM, Work with MQM Queues
- Queue Manager commands
 - CCTMQM, Connect Message Queue Manager
 - CHGMQM, Change Message Queue Manager
 - CRTMQM, Create Message Queue Manager
 - DLTMQM, Delete Message Queue Manager
 - DSCMQM, Disconnect Message Queue Manager
 - DSPMQM, Display Message Queue Manager
 - ENDMQM, End Message Queue Manager
 - STRMQM, Start Message Queue Manager
 - WRKMQM, Work with Message Queue managers
- Security commands
 - DSPMQMAUT, Display MQM Object Authority
 - GRTMQMAUT, Grant MQM Object Authority
 - RVKMQMAUT, Revoke MQM Object Authority
- Trace commands
 - TRCMQM, Trace MQM Job
- Transaction commands
 - RSVMQMTRN, Resolve MQSeries Transaction
 - WRKMQMTRN, Display MQSeries Transaction
- Trigger Monitor commands
 - STRMQMTRM, Start Trigger Monitor

Getting started

Use these commands and panels to:

1. Define message channels and associated objects
2. Monitor and control message channels

By using the F4=Prompt key, you can specify the relevant queue manager. If you do not use the prompt, the default queue manager is assumed. With F4=Prompt, an additional panel is displayed where you may enter the relevant queue manager name and sometimes other data.

The objects you need to define with the panels are:

- Transmission queues
- Remote queue definitions
- Queue manager alias definitions
- Reply-to queue alias definitions
- Reply-to local queues
- Processes for triggering (MCAs)
- Message channel definitions

See “Making your applications communicate” on page 15 for more discussion on the concepts involved in the use of these objects.

Channels must be completely defined, and their associated objects must exist and be available for use, before a channel can be started. This chapter shows you how to do this.

In addition, the particular communication link for each channel must be defined and available before a channel can be run. For a description of how LU 6.2 and TCP/IP links are defined, see the particular communication guide for your installation.

Creating objects

Use the CRTMQMQ command to create the queue and alias objects, such as: transmission queues, remote queue definitions, queue manager alias definitions, reply-to queue alias definitions, and reply-to local queues.

For a list of default objects, see the WebSphere MQ for i5/OS System Administration Guide book.

Creating a channel

To create a new channel:

1. Use F6 from the Work with MQM Channels panel (the second panel that displays channel details).

Alternatively, use the CRTMQMCHL command from the command line.

Either way, this displays the Create Channel panel. Type:

- The name of the channel in the field provided
- The channel type for this end of the link

2. Press Enter.

Note: You are strongly recommended to name all the channels in your network uniquely. As shown in Table 1 on page 26, including the source and target queue manager names in the channel name is a good way to do this.

Your entries are validated and errors are reported immediately. Correct any errors and continue.

You are presented with the appropriate channel settings panel for the type of channel you have chosen. Fill in the fields with the information you have gathered previously. Press Enter to create the channel.

You are provided with help in deciding on the content of the various fields in the descriptions of the channel definition panels in the help panels, and in “Channel attributes” on page 71.

```

Create MQM Channel (CRTMQMCHL)

Type choices, press Enter.

Channel name . . . . . > CHANNAME _____
Channel type . . . . . > *SDR_____ *RCVR, *SDR, *SVR, *RQSTR...
Message Queue Manager name      *DFT_____

Replace . . . . . *NO_____ *NO, *YES
Transport type . . . . . *TCP_____ *LU62, *TCP, *SYSDFTCHL
Text 'description' . . . . . > 'Example Channel Definition' _____

Connection name . . . . . *SYSDFTCHL_____
_____
_____
_____
_____
_____
_____
_____
More...
F3=Exit F4=Prompt F5=Refresh F12=Cancel F13=How to use this display
F24=More keys

```

Figure 55. Create channel (1)

```

Create MQM Channel (CRTMQMCHL)

Type choices, press Enter.

Transmission queue . . . . . 'TRANSMISSION_QUEUE_NAME' _____

Message channel agent . . . . . *NONE_____ Name, *SYSDFTCHL, *NONE
Library . . . . . _____ Name
Message channel agent user ID . *SYSDFTCHL_____ Character value...
Coded Character Set Identifier *SYSDFTCHL_____ 0-9999, *SYSDFTCHL
Batch size . . . . . 50_____ 1-9999, *SYSDFTCHL
Disconnect interval . . . . . 6000_____ 1-999999, *SYSDFTCHL
Short retry interval . . . . . 60_____ 0-999999999, *SYSDFTCHL
Short retry count . . . . . 10_____ 0-999999999, *SYSDFTCHL
Long retry interval . . . . . 1200_____ 0-999999999, *SYSDFTCHL
Long retry count . . . . . 999999999_____ 0-999999999, *SYSDFTCHL
Security exit . . . . . *NONE_____ Name, *SYSDFTCHL, *NONE
Library . . . . . _____ Name
Security exit user data . . . . *SYSDFTCHL_____

More...
F3=Exit F4=Prompt F5=Refresh F12=Cancel F13=How to use this display
F24=More keys

```

Figure 56. Create channel (2)

```

Create MQM Channel (CRTMQMCHL)

Type choices, press Enter.

Send exit . . . . . *NONE_____ Name, *SYSDFTCHL, *NONE
Library . . . . . _____ Name
+ for more values _____
Send exit user data . . . . . _____
+ for more values _____
Receive exit . . . . . *NONE_____ Name, *SYSDFTCHL, *NONE
Library . . . . . _____ Name
+ for more values _____
Receive exit user data . . . . . _____
+ for more values _____
Message exit . . . . . *NONE_____ Name, *SYSDFTCHL, *NONE
Library . . . . . _____ Name
+ for more values _____

More...
F3=Exit F4=Prompt F5=Refresh F12=Cancel F13=How to use this display
F24=More keys

```

Figure 57. Create channel (3)

```

Create MQM Channel (CRTMQMCHL)

Type choices, press Enter.

Message exit user data . . . . . _____
+ for more values _____
Convert message . . . . . *SYSDFTCHL_ *YES, *NO, *SYSDFTCHL
Sequence number wrap . . . . . 999999999_ 100-999999999, *SYSDFTCHL
Maximum message length . . . . . 4194304_ 0-4194304, *SYSDFTCHL
Heartbeat interval . . . . . 300_ 0-999999999, *SYSDFTCHL
Non Persistent Message Speed . . *FAST_ *FAST, *NORMAL, *SYSDFTCHL
Password . . . . . *SYSDFTCHL_ Character value, *BLANK...
Task User Profile . . . . . *SYSDFTCHL_ Character value, *BLANK...
Transaction Program Name . . . . *SYSDFTCHL

Bottom
F3=Exit F4=Prompt F5=Refresh F12=Cancel F13=How to use this display
F24=More keys

```

Figure 58. Create channel (4)

Starting a channel

Listeners are valid for TCP only. For SNA listeners, you must configure your communications subsystem.

For applications to be able to exchange messages you must start a listener program for inbound connections using the STRMQMLSR command.

For outbound connections you must start the channel in one of the following ways:

1. Use the CL command STRMQMCHL, specifying the channel name, to start the channel as a process or a thread, depending on the MCATYPE parameter. (If channels are started as threads, they are threads of a channel initiator.)
STRMQMCHL CHLNAME(QM1.TO.QM2) MQNAME(MYQMGR)

2. Use a channel initiator to trigger the channel. One channel initiator is started automatically when the queue manager is started. This can be eliminated by changing the chinit stanza in the qm.ini file for that queue manager.
3. Use the WRKMQMCHL command to begin the Work with Channels panel and choose option 14 to start a channel.

Selecting a channel

To select a channel, use the WRKMQMCHL command to begin at the Work with Channels panel:

1. Move the cursor to the option field at the left of the required channel name.
2. Type an option number.
3. Press Enter to activate your choice.

If you select more than one channel, the options are activated in sequence.

```

Work with MQM Channels
Queue Manager Name . . : CNX
Type options, press Enter.
  2=Change  3=Copy  4=Delete  5=Display  8=Work with Status  13=Ping
 14=Start  15=End  16=Reset  17=Resolve

Opt   Name                Type      Transport  Status
-----
      CHLNIC                *RCVR    *TCP       INACTIVE
      CORSAIR.TO.MUSTANG    *SDR     *LU62      INACTIVE
      FV.CHANNEL.MC.DJE1    *RCVR    *TCP       INACTIVE
      FV.CHANNEL.MC.DJE2    *SDR     *TCP       INACTIVE
      FV.CHANNEL.MC.DJE3    *RQSTR   *TCP       INACTIVE
      FV.CHANNEL.MC.DJE4    *SVR     *TCP       INACTIVE
      FV.CHANNEL.PETER     *RCVR    *TCP       INACTIVE
      FV.CHANNEL.PETER.LU  *RCVR    *LU62      INACTIVE
      FV.CHANNEL.PETER.LU1 *RCVR    *LU62      INACTIVE
More...

Parameters or command
====>
F3=Exit  F4=Prompt  F5=Refresh  F6=Create  F9=Retrieve  F12=Cancel
F21=Print

```

Figure 59. Work with channels

Browsing a channel

To browse the settings of a channel, use the WRKMQMCHL command to begin at the Display Channel panel:

1. Move the cursor to the left of the required channel name.
2. Type option 5 (Display).
3. Press Enter to activate your choice.

If you select more than one channel, they are presented in sequence.

Alternatively, you can use the DSPMQMCHL command from the command line.

This results in the respective Display Channel panel being displayed with details of the current settings for the channel. The fields are described in “Channel attributes” on page 71.

```

                                Display MQM Channel
Channel name . . . . . : ST.JST.2T01
Queue Manager Name . . . . . : QMREL
Channel type . . . . . : *SDR
Transport type . . . . . : *TCP
Text 'description' . . . . . : John's sender to WINSDOA1

Connection name . . . . . : MUSTANG

Transmission queue . . . . . : WINSDOA1

Message channel agent . . . . . :
  Library . . . . . :
Message channel agent user ID : *NONE
Batch interval . . . . . : 0
Batch size . . . . . : 50
Disconnect interval . . . . . : 6000

F3=Exit  F12=Cancel  F21=Print

```

Figure 60. Display a TCP/IP channel (1)

```

                                Display MQM Channel
Short retry interval . . . . . : 60
Short retry count . . . . . : 10
Long retry interval . . . . . : 6000
Long retry count . . . . . : 10
Security exit . . . . . :
  Library . . . . . :
Security exit user data . . . . . :
Send exit . . . . . :
  Library . . . . . :
Send exit user data . . . . . :
Receive exit . . . . . :
  Library . . . . . :
Receive exit user data . . . . . :
Message exit . . . . . :
  Library . . . . . :
Message exit user data . . . . . :

                                More...

F3=Exit  F12=Cancel  F21=Print

```

Figure 61. Display a TCP/IP channel (2)

```
Display MQM Channel
Sequence number wrap . . . . . : 999999999
Maximum message length . . . . . : 10000
Convert message . . . . . : *NO
Heartbeat interval . . . . . : 300
Nonpersistent message speed . . : *FAST

Bottom

F3=Exit  F12=Cancel  F21=Print
```

Figure 62. Display a TCP/IP channel (3)

Renaming a channel

To rename a message channel, begin at the Work with Channels panel:

1. End the channel.
2. Use option 3 (Copy) to create a duplicate with the new name.
3. Use option 5 (Display) to check that it has been created correctly.
4. Use option 4 (Delete) to delete the original channel.

If you decide to rename a message channel, ensure that both channel ends are renamed at the same time.

Work with channel status

Use the WRKMQMCHST command to bring up the first of three screens showing the status of your channels. You can view the three status screens in sequence when you select Change-view (F11).

Alternatively, selecting option 8 (Work with Status) from the Work with MQM Channels panel also brings up the first status panel.

Work with channel status applies to all message channels. It does not apply to MQI channels other than server-connection channels on WebSphere MQ for i5/OS V5.1 and later.

Note: The Work with Channel Status screens only show channels that are active after messages have been sent through the channel and the sequence number has been incremented.

```

MQSeries Work with Channel Status

Type options, press Enter.
 5=Display 13=Ping 14=Start 15=End 16=Reset 17=Resolve

Opt Name Connection Indoubt Last Seq
CARTS_CORSAIR_CHAN GBIBMIYA.WINSDOA1 NO 1
CHLNIC 9.20.2.213 NO 3
FV.CHANNEL.PETER2 9.20.2.213 NO 6225
JST.1.2 9.20.2.201 NO 28
MP_MUST_TO_CORS 9.20.2.213 NO 100
MUSTANG.TO.CORSAIR GBIBMIYA.WINSDOA1 NO 10
MP_CORS_TO_MUST 9.20.2.213 NO 101
JST.2.3 9.5.7.126 NO 32
PF_WINSDOA1_LU62 GBIBMIYA.IYA80020 NO 54
PF_WINSDOA1_LU62 GBIBMIYA.WINSDOA1 NO 500
ST.JCW.EXIT.2T01.CHL 9.20.2.213 NO 216

Bottom

Parameters or command
===>
F3=Exit F4=Prompt F5=Refresh F6=Create F9=Retrieve F11=Change view
F12=Cancel F21=Print

```

Figure 63. Channel status (1)

Change the view with F11.

```

MQSeries Work with Channel Status

Type options, press Enter.
 5=Display 13=Ping 14=Start 15=End 16=Reset 17=Resolve

Opt Transmission Queue LUWID
7516E58A40C000EC
7515A36C0D800157
7515E790AC8001CA
7516FF2284800009
75147C6629C0009D
7516DDE5778000A8
FV_MKP_TRANS_QUEUE 75147B61A44000FA
JST.3 75170185D0000133
PF.WINSDOA1 7516DA3955C00097
PF.WINSDOA1 7516DE2396C000BC
ST.JCW.EXIT.2T01.XMIT.QUEUE 7516C51291400016

Bottom

Parameters or command
===>
F3=Exit F4=Prompt F5=Refresh F6=Create F9=Retrieve F11=Change view
F12=Cancel F21=Print

```

Figure 64. Channel status (2)

```

MQSeries Work with Channel Status

Type options, press Enter.
 5=Display 13=Ping 14=Start 15=End 16=Reset 17=Resolve

Opt      Indoubt      Indoubt      Indoubt
         Msgs       Seq         LUWID
0         0         0 0000000000000000
0         0         0 0000000000000000
0         0         0 0000000000000000
0         0         0 0000000000000000
0         0         0 0000000000000000
0         0         0 0000000000000000
0         0         0 0000000000000000
0         101      75147B61A44000FA
0         32       75170185D0000133
0         54       7516DA3955C00097
0         500      7516DE2396C000BC
0         216      7516C51291400016

Parameters or command
====>
F3=Exit  F4=Prompt  F5=Refresh  F6=Create  F9=Retrieve  F11=Change view
F12=Cancel F21=Print

Bottom

```

Figure 65. Channel status (3)

The options available in the Work with Channel Status panel are:

Menu option	Description
5=Display	Displays the channel settings.
13=Ping	Initiates a Ping action, where appropriate.
14=Start	Starts the channel.
15=End	Stops the channel.
16=Reset	Resets the channel sequence number.
17=Resolve	Resolves an in-doubt channel situation, manually.
F11=Change view	Cycles around the three status panels.

Note: When using the WRKMQMCHST command, the channel status shown is SAVED channel status not CURRENT channel status. To see CURRENT channel status, use the WRKMQMCHL command.

Work-with-channel choices

The Work with Channels panel is reached with the command WRKMQMCHL, and it allows you to monitor the status of all channels listed, and to issue commands against selected channels.

The options available in the Work with Channel panel are:

Menu option	Description
F6=Create	Creates a channel.
2=Change	Changes the attributes of a channel.
3=Copy	Copies the attributes of a channel to a new channel.
4=Delete	Deletes a channel.
5=Display	Displays the current settings for the channel.
8=Work with status	Displays the channel status panels.
13=Ping	Runs the Ping facility to test the connection to the adjacent system by exchanging a fixed data message with the remote end.
14=Start	Starts the selected channel, or resets a disabled receiver channel.
15=End	Requests the channel to close down.

Menu option	Description
16=Reset	Requests the channel to reset the sequence numbers on this end of the link. The numbers must be equal at both ends for the channel to start.
17=Resolve	Requests the channel to resolve in-doubt messages without establishing connection to the other end.

Panel choices

The following choices are provided in the Work with MQM channels panel and the Work with Channel Status panel.

F6=Create

Use the Create option, or enter the CRTMQMCHL command from the command line, to obtain the Create Channel panel. There are examples of Create Channel panels, starting at Figure 55 on page 325.

With this panel, you create a new channel definition from a screen of fields filled with default values supplied by WebSphere MQ for i5/OS. Type the name of the channel, select the type of channel you are creating, and the communication method to be used.

When you press Enter, the panel is displayed. Type information in all the required fields in this panel, and the three pages making up the complete panel, and then save the definition by pressing Enter.

The channel name must be the same at both ends of the channel, and unique within the network. However, you should restrict the characters used to those that are valid for WebSphere MQ for i5/OS object names; see “Channel attributes” on page 71.

All panels have default values supplied by WebSphere MQ for i5/OS for some fields. You can customize these values, or you can change them when you are creating or copying channels. To customize the values, see the *WebSphere MQ for i5/OS System Administration*.

You can create your own set of channel default values by setting up dummy channels with the required defaults for each channel type, and copying them each time you want to create new channel definitions.

Table 31 shows the channel attributes for each type of channel. See “Channel attributes” on page 71 for details about the fields.

Table 31. Channel attribute fields per message channel type

Attribute field	Sender	Server	Receiver	Requester
Batch size	Yes	Yes	Yes	Yes
Channel name	Yes	Yes	Yes	Yes
Channel type	Yes	Yes	Yes	Yes
Connection name	Yes	Yes		Yes
Context			Yes	Yes
Disconnect interval	Yes	Yes		

Table 31. Channel attribute fields per message channel type (continued)

Attribute field	Sender	Server	Receiver	Requester
Heartbeat interval	Yes	Yes	Yes	Yes
Long retry wait interval	Yes	Yes		
Long retry count	Yes	Yes		
Maximum message length	Yes	Yes	Yes	Yes
Message channel agent name				Yes
Message exit user data	Yes	Yes	Yes	Yes
Message retry exit count			Yes	Yes
Message retry exit data			Yes	Yes
Message retry exit interval			Yes	Yes
Message retry exit name			Yes	Yes
Nonpersistent message speed	Yes	Yes	Yes	Yes
Receive exit	Yes	Yes	Yes	Yes
Receive exit user data	Yes	Yes	Yes	Yes
Security exit	Yes	Yes	Yes	Yes
Security exit user data	Yes	Yes	Yes	Yes
Send exit	Yes	Yes	Yes	Yes
Send exit user data	Yes	Yes	Yes	Yes
Sequence number wrap	Yes	Yes	Yes	Yes
Short retry wait interval	Yes	Yes		
Short retry count	Yes	Yes		
Transport type	Yes	Yes	Yes	Yes
Transmission queue	Yes	Yes		
Message exit	Yes	Yes	Yes	Yes

2=Change

Use the Change option, or the CHGMQMCHL command, to change an existing channel definition, except for the channel name. Simply type over the fields to be changed in the channel definition panel, and then save the updated definition by pressing Enter.

3=Copy

Use the Copy option, or the CPYMQMCHL command, to copy an existing channel. The Copy panel enables you to define the new channel name. However, you should restrict the characters used to those that are valid for WebSphere MQ for i5/OS object names; see the *WebSphere MQ for i5/OS System Administration*.

Press Enter on the Copy panel to display the details of current settings. You can change any of the new channel settings. Save the new channel definition by pressing Enter.

4=Delete

Use the Delete option to delete the selected channel. A panel is displayed to confirm or cancel your request.

5=Display

Use the Display option to display the current definitions for the channel. This choice displays the panel with the fields showing the current values of the parameters, and protected against user input.

8=Work with Status

The status column tells you whether the channel is active or inactive, and is displayed continuously in the Work with MQM Channels panel. Use option 8 (Work with Status) to see more status information displayed. Alternatively, this can be displayed from the command line with the WRKMQMCHST command. See “Work with channel status” on page 329.

- Channel name
- Communication connection name
- In-doubt status of channel (where appropriate)
- Last sequence number
- Transmission queue name (where appropriate)
- The in-doubt identifier (where appropriate)
- The last committed sequence number
- Logical unit of work identifier

13=Ping

Use the Ping option to exchange a fixed data message with the remote end. This gives some confidence to the system supervisor that the link is available and functioning.

Ping does not involve the use of transmission queues and target queues. It uses channel definitions, the related communication link, and the network setup.

It is available from sender and server channels, only. The corresponding channel is started at the far side of the link, and performs the start up parameter negotiation. Errors are notified normally.

The result of the message exchange is presented in the Ping panel for you, and is the returned message text, together with the time the message was sent, and the time the reply was received.

Ping with LU 6.2:

When Ping is invoked in WebSphere MQ for i5/OS, it is run with the USERID of the user requesting the function, whereas the normal way that a channel program is run is for the QMQM USERID to be taken for channel programs. The USERID flows to the receiving side and it must be valid on the receiving end for the LU 6.2 conversation to be allocated.

14=Start

The Start option is available for sender, server, and requester channels. It should not be necessary where a channel has been set up with queue manager triggering.

The Start option is also used for receiver channels that have a DISABLED or STOPPED status. Starting a receiver channel that is in DISABLED or STOPPED state resets the channel and allows it to be started from the remote channel.

When started, the sending MCA reads the channel definition file and opens the transmission queue. A channel start-up sequence is executed, which remotely starts the corresponding MCA of the receiver or server channel. When they have been started, the sender and server processes await messages arriving on the transmission queue and transmit them as they arrive.

When you use triggering, you will need to start the continuously running trigger process to monitor the initiation queue. The STRMQMCHLI command can be used for this.

At the far end of a channel, the receiving process may be started in response to a channel startup from the sending end. The method of doing this is different for LU 6.2 and TCP/IP connected channels:

- LU 6.2 connected channels do not require any explicit action at the receiving end of a channel.
- TCP connected channels require a listener process to be running continuously. This process awaits channel startup requests from the remote end of the link and starts the process defined in the channel definitions for that connection. When the remote system is i5/OS, you can use the STRMQMLSR command for this.

Use of the Start option always causes the channel to re-synchronize, where necessary.

For the start to succeed:

- Channel definitions, local and remote must exist. If there is no appropriate channel definition for a receiver or server-connection channel, a default one is created automatically if the channel is auto-defined. See “Channel auto-definition exit program” on page 393.
- The transmission queue must exist, be enabled for GETs, and have no other channels using it.
- MCAs, local and remote, must exist.
- The communication link must be available.
- The queue managers must be running, local and remote.
- The message channel must be inactive.

To transfer messages, remote queues and remote queue definitions *must* exist.

A message is returned to the panel confirming that the request to start a channel has been accepted. For confirmation that the Start process has succeeded, check the system log, or press F5 (refresh the screen).

15=End

Use the End option to request the channel to stop activity. The channel will not send any more messages until the operator starts the channel again. (For information about restarting stopped channels, see “Restarting stopped channels” on page 64.)

You can select the type of stop you require if you press F4 before Enter. You can choose IMMEDIATE, or CONTROLLED.

Stop immediate:

Normally, this option should not be used. It terminates the channel process. The channel does not complete processing the current batch of messages, and cannot, therefore, leave the channel in doubt. In general, it is recommended that the operators use the controlled stop option.

Stop controlled:

This choice requests the channel to close down in an orderly way; the current batch of messages is completed, and the syncpoint procedure is carried out with the other end of the channel.

16=Reset

The Reset option changes the message sequence number. Use it with care, and only after you have used the Resolve option to resolve any in-doubt situations. This option is available only at the sender or server channel. The first message starts the new sequence the next time the channel is started.

17=Resolve

Use the Resolve option when messages are held in-doubt by a sender or server, for example because one end of the link has terminated, and there is no prospect of it recovering. The Resolve option accepts one of two parameters: BACKOUT or COMMIT. Backout restores messages to the transmission queue, while Commit discards them.

The channel program does not try to establish a session with a partner. Instead, it determines the logical unit of work identifier (LUWID) which represents the in-doubt messages. It then issues, as requested, either:

- BACKOUT to restore the messages to the transmission queue; or
- COMMIT to delete the messages from the transmission queue.

For the resolution to succeed:

- The channel must be inactive
- The channel must be in doubt
- The channel type must be sender or server
- The channel definition, local, must exist
- The queue manager must be running, local

Preparing WebSphere MQ for i5/OS

This chapter describes the WebSphere MQ for i5/OS preparations required before DQM can be used. Communication preparations are described in “Setting up communication for WebSphere MQ for i5/OS” on page 342.

Before a channel can be started, the transmission queue must be defined as described in this chapter, and must be included in the message channel definition.

In addition, where needed, the triggering arrangement must be prepared with the definition of the necessary processes and queues.

Creating a transmission queue

You define a local queue with the Usage field attribute set to *TMQ, for each sending message channel.

If you want to make use of remote queue definitions, use the same command to create a queue of type *RMT, and Usage of *NORMAL.

To create a transmission queue, use the CRTMQMQ command from the command line to present you with the first queue creation panel; see Figure 66.

```
                Create MQM Queue (CRTMQMQ)

Type choices, press Enter.
Queue name . . . . .
Queue type . . . . . ____ *ALS, *LCL, *MDL, *RMT
Message Queue Manager name . . . *DFT _____
_____

F3=Exit  F4=Prompt  F5=Refresh  F12=Cancel  F13=How to use this display
F24=More keys

Bottom
+
```

Figure 66. Create a queue (1)

Type the name of the queue and specify the type of queue that you wish to create: Local, Remote, or Alias. For a transmission queue, specify Local (*LCL) on this panel and press Enter.

You are presented with the second page of the Create MQM Queue panel; see Figure 67 on page 338.

```

Create MQM Queue (CRTMQMQ)

Type choices, press Enter.

Queue name . . . . . > HURS.2.HURS.PRIORIT

Queue type . . . . . > *LCL          *ALS, *LCL, *MDL, *RMT
Message Queue Manager name . . . *DFT
Replace . . . . . *NO             *NO, *YES
Text 'description' . . . . . '
Put enabled . . . . . *YES         *SYSDFTQ, *NO, *YES
Default message priority . . . . 0          0-9, *SYSDFTQ
Default message persistence . . . *NO          *SYSDFTQ, *NO, *YES
Process name . . . . . '
Triggering enabled . . . . . *NO         *SYSDFTQ, *NO, *YES
Get enabled . . . . . *YES         *SYSDFTQ, *NO, *YES
Sharing enabled . . . . . *YES         *SYSDFTQ, *NO, *YES

More...
F3=Exit  F4=Prompt  F5=Refresh  F12=Cancel  F13=How to use this display
F24=More keys

```

Figure 67. Create a queue (2)

Change any of the default values shown. Press page down to scroll to the next screen; see Figure 68.

```

Create MQM Queue (CRTMQMQ)

Type choices, press Enter.

Default share option . . . . . *YES          *SYSDFTQ, *NO, *YES
Message delivery sequence . . . *PTY          *SYSDFTQ, *PTY, *FIFO
Harden backout count . . . . . *NO           *SYSDFTQ, *NO, *YES
Trigger type . . . . . *FIRST             *SYSDFTQ, *FIRST, *ALL...
Trigger depth . . . . . 1                1-999999999, *SYSDFTQ
Trigger message priority . . . . 0          0-9, *SYSDFTQ
Trigger data . . . . . '
Retention interval . . . . . 999999999    0-999999999, *SYSDFTQ
Maximum queue depth . . . . . 5000        1-24000, *SYSDFTQ
Maximum message length . . . . . 4194304  0-4194304, *SYSDFTQ
Backout threshold . . . . . 0            0-999999999, *SYSDFTQ
Backout requeue queue . . . . . '
Initiation queue . . . . . '

More...
F3=Exit  F4=Prompt  F5=Refresh  F12=Cancel  F13=How to use this display
F24=More keys

```

Figure 68. Create a queue (3)

Type *TMQ, for transmission queue, in the Usage field of this panel, and change any of the default values shown in the other fields.

```

Create MQM Queue (CRTMQMQ)

Type choices, press Enter.

Usage . . . . . *TMQ          *SYSDFTQ, *NORMAL, *TMQ
Queue depth high threshold . . . 80          0-100, *SYSDFTQ
Queue depth low threshold . . . 20          0-100, *SYSDFTQ
Queue full events enabled . . . *YES      *SYSDFTQ, *NO, *YES
Queue high events enabled . . . *YES      *SYSDFTQ, *NO, *YES
Queue low events enabled . . . *YES      *SYSDFTQ, *NO, *YES
Service interval . . . . . 999999999      0-999999999, *SYSDFTQ
Service interval events . . . *NONE     *SYSDFTQ, *HIGH, *OK, *NONE
Distribution list support . . . *NO       *SYSDFTQ, *NO, *YES
Cluster Name . . . . . *SYSDFTQ
Cluster Name List . . . . . *SYSDFTQ
Default Binding . . . . . *SYSDFTQ      *SYSDFTQ, *OPEN, *NOTFIXED

Bottom
F3=Exit  F4=Prompt  F5=Refresh  F12=Cancel  F13=How to use this display
F24=More keys

```

Figure 69. Create a queue (4)

When you are satisfied that the fields contain the correct data, press Enter to create the queue.

Triggering channels in WebSphere MQ for i5/OS

Implement triggering in WebSphere MQ for i5/OS by specifying the initiation queue to the channel initiator process and creating a suitable transmission queue.

An overview of triggering is given in “Triggering channels” on page 18, and it is described in depth in the WebSphere MQ Application Programming Guide. This section provides you with information specific to WebSphere MQ for i5/OS.

Triggering of channels in WebSphere MQ for i5/OS is implemented with the channel initiator process. A channel initiator process for the initiation queue SYSTEM.CHANNEL.INITQ is typically started automatically with the queue manager. However, you can choose to manually start a channel initiator for a different initiation queue, and the automatic startup of the channel initiator can be disabled by altering the queue manager SCHINIT attribute. You can manually start up to three channel initiator processes with the STRMQMCHLI command. For example:

```
STRMQMCHLI QNAME(MYINITQ)
```

Set up the transmission queue for the channel, specifying TRGENBL(*YES) and specifying the channel name in the TRIGDATA field. For example:

```
CRTMQMQ QNAME(MYXMITQ) QTYPE(*LCL) MQMNAME(MYQMGR) +
TRGENBL(*YES) INITQNAME(SYSTEM.CHANNEL.INITQ) +
USAGE(*TMQ) TRIGDATA(MYCHANNEL)
```

Channel programs

There are different types of channel programs (MCAs) available for use at the channels. The names are contained in the following table.

Table 32. Program and transaction names

Program name	Direction of connection	Communication
AMQCRSTA	Inbound	TCP
AMQCRS6A	Inbound	LU 6.2
AMQRMCLA	Outbound	Any

Channel states on i5/OS

Channel states are displayed on the Work with Channels panel (described in Figure 59 on page 327). There are some differences between the names of channel states on different versions of WebSphere MQ for i5/OS. In the following table, the state names shown for V4R2 correspond to the channel states described in Figure 30 on page 57. As shown in the table, some of these states have different names, or do not exist for earlier versions.

Table 33. Channel states on i5/OS

State name (V3R6)	State name (V3R2, V3R7, V4R2, V5R1)	Meaning
-	STARTING	Channel is ready to begin negotiation with target MCA
BINDING	BINDING	Establishing a session and initial data exchange
REQUESTING	REQUESTING	Requester channel initiating a connection
READY	RUNNING	Transferring or ready to transfer
PAUSED	PAUSED	Waiting for message-retry interval
CLOSING	STOPPING	Establishing whether to retry or stop
RETRYING	RETRYING	Waiting until next retry attempt
DISABLED	STOPPED	Channel stopped because of an error or because an end-channel command is issued
STOPPED	INACTIVE	Channel ended processing normally or channel never started
-	*None	No state (for server-connection channels only)
Note: The state *None applies only to V3R2 and V3R7.		

Other things to consider

Here are some other topics that you should consider when preparing WebSphere MQ for distributed queue management.

Undelivered-message queue

It is advisable that you have an application available to process the messages arriving on the undelivered-message queue (also known as the dead-letter queue or DLQ). The program could be triggered, or run at regular intervals. For more details, see the *WebSphere MQ for i5/OS System Administration* and the *WebSphere MQ Application Programming Guide*.

Queues in use

MCAs for receiver channels may keep the destination queues open even when messages are not being transmitted; this results in the queues appearing to be “in use”.

Maximum number of channels

You can specify the maximum number of channels allowed in your system and the maximum number that can be active at one time. You do this in the `qm.ini` file in directory `QIBM/UserData/mqm/qmgrs/queue manager name`. See Chapter 8, “Configuration file stanzas for distributed queuing,” on page 491.

Security of WebSphere MQ for i5/OS objects

This section deals with remote messaging aspects of security.

You need to provide users with authority to make use of the WebSphere MQ for i5/OS facilities, and this is organized according to actions to be taken with respect to objects and definitions. For example:

- Queue managers can be started and stopped by authorized users
- Applications need to connect to the queue manager, and have authority to make use of queues
- Message channels need to be created and controlled by authorized users

The message channel agent at a remote site needs to check that the message being delivered has derived from a user with authority to do so at this remote site. In addition, as MCAs can be started remotely, it may be necessary to verify that the remote processes trying to start your MCAs are authorized to do so. There are three possible ways for you to deal with this:

1. Decree in the channel definition that messages must contain acceptable *context* authority, otherwise they will be discarded.
2. Implement *user exit* security checking to ensure that the corresponding message channel is authorized. The security of the installation hosting the corresponding channel ensures that all users are properly authorized, so that you do not need to check individual messages.
3. Implement *user exit* message processing to ensure that individual messages are vetted for authorization.

Here are some facts about the way WebSphere MQ for i5/OS operates security:

- Users are identified and authenticated by i5/OS.
- Queue manager services invoked by applications are run with the authority of the queue manager user profile, but in the user’s process.
- Queue manager services invoked by user commands are run with the authority of the queue manager user profile.

System extensions and user-exit programs

A facility is provided in the channel definition to allow extra programs to be run at defined times during the processing of messages. These programs are not supplied with WebSphere MQ for i5/OS, but may be provided by each installation according to local requirements.

In order to run, such programs must have predefined names and be available on call to the channel programs. The names of the exit programs are included in the message channel definitions.

There is a defined control block interface for handing over control to these programs, and for handling the return of control from these programs.

The precise places where these programs are called, and details of control blocks and names, are to be found in Chapter 6, “Further intercommunication considerations,” on page 375.

Setting up communication for WebSphere MQ for i5/OS

DQM is a remote queuing facility for WebSphere MQ for i5/OS. It provides channel control programs for the WebSphere MQ for i5/OS queue manager which form the interface to communication links, controllable by the system operator. The channel definitions held by distributed-queuing management use these communication links.

When a distributed-queuing management channel is started, it tries to use the connection specified in the channel definition. For this to succeed, it is necessary for the connection to be defined and available. This chapter explains how to do this.

Deciding on a connection

There are two forms of communication between WebSphere MQ for i5/OS systems:

- i5/OS TCP

For TCP, a host address may be used, and these connections are set up as described in the *i5/OS Communication Configuration Reference*.

In the TCP environment, each distributed service is allocated a unique TCP address which may be used by remote machines to access the service. The TCP address consists of a host name/number and a port number. All queue managers will use such a number to communicate with each other via TCP.

- i5/OS SNA (LU 6.2)

This form of communication requires the definition of an i5/OS SNA logical unit type 6.2 (LU 6.2) that provides the physical link between the i5/OS system serving the local queue manager and the system serving the remote queue manager. Refer to the *i5/OS Communication Configuration Reference* for details on configuring communications in i5/OS.

Defining a TCP connection

The channel definition contains a field, CONNECTION NAME, that contains either the TCP network address of the target, in IPv4 dotted decimal form (for example 9.20.9.30) or IPv6 hexadecimal form (for example fe80:43e4:0204:acff:fe97:2c34:fde0:3485), or the host name (for example AS4HUR1). If the CONNECTION NAME is a host name, a name server or the i5/OS host table is used to convert the host name into a TCP host address.

A port number is required for a complete TCP address; if this is not supplied, the default port number 1414 is used. On the initiating end of a connection (sender, requester, and server channel types) it is possible to provide an optional port number for the connection, for example:

```
Connection name 9.20.9.30 (1555)
```

In this case the initiating end will attempt to connect to a receiving program at port 1555.

Receiving on TCP

Receiving channel programs are started in response to a startup request from the sending channel. To do this, a listener program has to be started to detect incoming network requests and start the associated channel. You start this listener program with the STRMQMLSR command.

You can start more than one listener for each queue manager. By default, the STRMQMLSR command uses port 1414 but you can override this. To override the default setting, add the following statements to the qm.ini file of the selected queue manager (in this example, the listener is required to use port 2500):

```
TCP:
  Port=2500
```

The qm.ini file is located in this IFS directory: /QIBM/UserData/mqm/qmgrs/*queue manager name*.

This new value is read only when the TCP listener is started. If you have a listener already running, this change is not be seen by that program. To use the new value, stop the listener and issue the STRMQMLSR command again. Now, whenever you use the STRMQMLSR command, the listener defaults to the new port.

Alternatively, you can specify a different port number on the STRMQMLSR command. For example:

```
STRMQMLSR MQMNAME(queue manager name) PORT(2500)
```

This change makes the listener default to the new port for the duration of the listener job.

Using the TCP SO_KEEPALIVE option:

If you want to use the SO_KEEPALIVE option (as discussed in “Checking that the other end of the channel is still available” on page 61) you must add the following entry to your queue manager configuration file (qm.ini in the IFS directory, /QIBM/UserData/mqm/qmgrs/*queue manager name*):

```
TCP:
  KeepAlive=yes
```

You must then issue the following command:

```
CFGTCP
```

Select option 3 (Change TCP Attributes). You can now specify a time interval in minutes. You can specify a value in the range 1 through 40320 minutes; the default is 120.

Using the TCP listener backlog option:

When receiving on TCP, a maximum number of outstanding connection requests is set. This can be considered a *backlog* of requests waiting on the TCP port for the listener to accept the request.

The default listener backlog value on i5/OS is 255. If the backlog reaches this value, the TCP connection is rejected and the channel will not be TCP: able to start.

For MCA channels, this results in the channel going into a RETRY state and retrying the connection at a later time.

For client connections, the client receives an MQRC_Q_MGR_NOT_AVAILABLE reason code from MQCONN and should retry the connection at a later time.

However, to avoid this error, you can add an entry in the qm.ini file:

```
ListenerBacklog = n
```

This overrides the default maximum number of outstanding requests (255) for the TCP listener.

Note: Some operating systems support a larger value than the default. If necessary, this can be used to avoid reaching the connection limit.

Defining an LU 6.2 connection

In WebSphere MQ for i5/OS, a mode name, TP name, and connection name of a fully-qualified LU 6.2 connection can be used.

For other versions of WebSphere MQ for i5/OS, a communications side information (CSI) object is required to define the LU 6.2 communications details for the sending end of a message channel. It is referred to in the CONNECTION NAME field of the Sender or Server channel definition for LU 6.2 connections. Further information on the communications side object is available in the *i5/OS APPC Communications Programmer's Guide*.

The initiated end of the link must have a routing entry definition to complement this CSI object. Further information on managing work requests from remote LU 6.2 systems is available in the *i5/OS Programming: Work Management Guide*.

See the *Multiplatform APPC Configuration Guide* and the following table for information.

Table 34. Settings on the local i5/OS system for a remote queue manager platform

Remote platform	TPNAME
z/OS, or OS/390 or MVS/ESA	The same as in the corresponding side information on the remote queue manager.
i5/OS	The same as the compare value in the routing entry on the i5/OS system.
HP OpenVMS	As specified in the OVMS Run Listener command.
Compaq NonStop Kernel	The same as the TPNAME specified in the receiver-channel definition.
UNIX systems	The invocable Transaction Program defined in the remote LU 6.2 configuration.

Table 34. Settings on the local i5/OS system for a remote queue manager platform (continued)

Remote platform	TPNAME
Windows	As specified in the Windows Run Listener command, or the invocable Transaction Program that was defined using TpSetup on Windows.

If you have more than one queue manager on the same machine, ensure that the TPnames in the channel definitions are unique.

Initiating end (Sending)

Use the CRTMQMCHL command to define a channel of transport type *LU62. For versions previous to WebSphere MQ for i5/OS V5.3, define the name of the CSI object that this channel will use in the CONNECTION NAME field. (See “Creating a channel” on page 324 for details of how to do this.) Use of the CSI object is optional in WebSphere MQ for i5/OS V5.3 or later.

The initiating end panel is shown in Figure Figure 70. You press F10 from the first panel displayed to obtain the complete panel as shown.

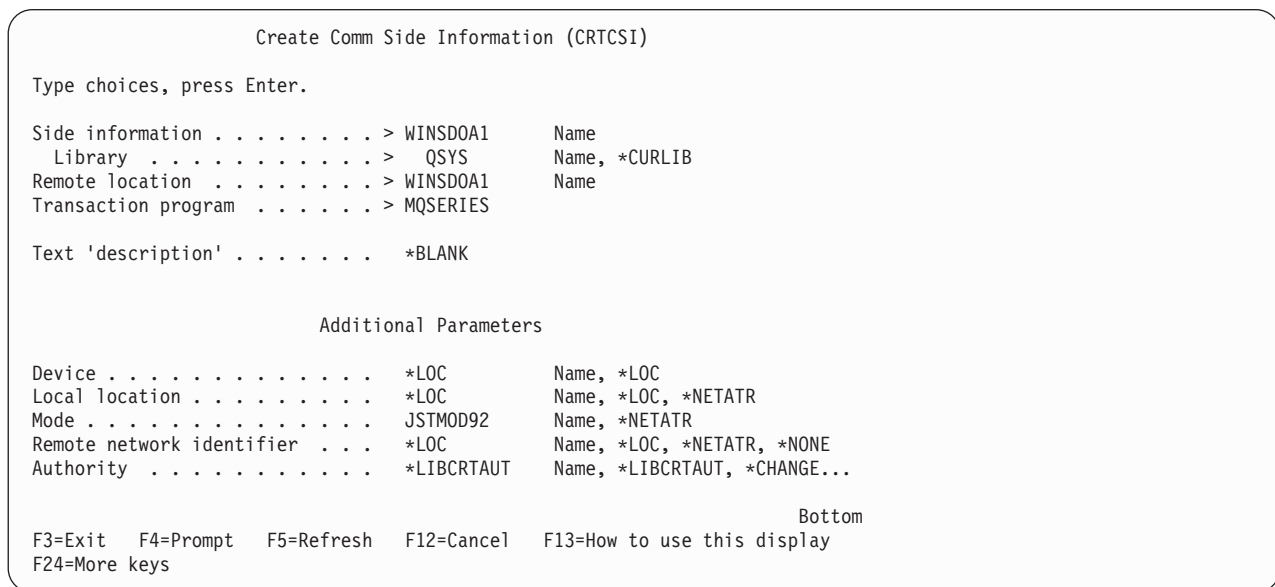


Figure 70. LU 6.2 communication setup panel - initiating end

Complete the initiating end fields as follows:

Side information

Give this definition a name that will be used to store the side information object to be created, for example, WINSDOA1.

Note: For LU 6.2, the link between the message channel definition and the communication connection is the **Connection name** field of the message channel definition at the sending end. This field contains the name of the CSI object.

Library

The name of the library where this definition will be stored.

The CSI object must be available in a library accessible to the program serving the message channel, for example, QSYS, QMQM, and QGPL.

If the name is incorrect, missing, or cannot be found then an error will occur on channel start up.

Remote location

Specifies the remote location name with which your program communicates.

In short, this required parameter contains the logical unit name of the partner at the remote system, as defined in the device description that is used for the communication link between the two systems.

The **Remote location** name can be found by issuing the command DSPNETA on the remote system and seeing the default local location name.

Transaction program

Specifies the name (up to 64 characters) of the transaction program on the remote system to be started. It may be a transaction process name, a program name, the channel name, or a character string that matches the **Compare value** in the routing entry.

This is a required parameter.

Note: To specify SNA service transaction program names, enter the hexadecimal representation of the service transaction program name. For example, to specify a service transaction program name whose hexadecimal representation is 21F0F0F1, you would enter X'21F0F0F1'.

More information on SNA service transaction program names is in the *SNA Transaction Programmer's Reference* manual for LU Type 6.2.

If the receiving end is another i5/OS system, the **Transaction program** name is used to match the CSI object at the sending end with the routing entry at the receiving end. This should be unique for each queue manager on the target i5/OS system. (See the **Program to call** parameter under "Initiated end (Receiver)" on page 348.) See also the **Comparison data: compare value** parameter in the Add Routing Entry panel.

Text description

A description (up to 50 characters) to remind you of the intended use of this connection.

Device

Specifies the name of the device description used for the remote system. The possible values are:

***LOC** The device is determined by the system.

Device-name

Specify the name of the device that is associated with the remote location.

Local location

Specifies the local location name. The possible values are:

***LOC** The local location name is determined by the system.

***NETATR**

The LCLLOCNAME value specified in the system network attributes is used.

Local-location-name

Specify the name of your location. Specify the local location if you want to indicate a specific location name for the remote location. The location name can be found by using the DSPNETA command.

Mode Specifies the mode used to control the session. This name is the same as the Common Programming Interface (CPI)- Communications Mode_Name. The possible values are:

***NETATR**

The mode in the network attributes is used.

BLANK

Eight blank characters are used.

Mode-name

Specify a mode name for the remote location.

Note: Because the mode determines the transmission priority of the communications session, it may be useful to define different modes depending on the priority of the messages being sent; for example MQMODE_HI, MQMODE_MED, and MQMODE_LOW. (You can have more than one CSI pointing to the same location.)

Remote network identifier

Specifies the remote network identifier used with the remote location. The possible values are:

***LOC** The remote network ID for the remote location is used.

***NETATR**

The remote network identifier specified in the network attributes is used.

***NONE**

The remote network has no name.

Remote-network-id

Specify a remote network ID. Use the DSPNETA command at the remote location to find the name of this network ID. It is the 'local network ID' at the remote location.

Authority

Specifies the authority you are giving to users who do not have specific authority to the object, who are not on an authorization list, and whose group profile has no specific authority to the object. The possible values are:

***LIBCRTAUT**

Public authority for the object is taken from the CRTAUT parameter of the specified library. This value is determined at create time. If the CRTAUT value for the library changes after the object is created, the new value does not affect existing objects.

***CHANGE**

Change authority allows the user to perform basic functions on the object, however, the user cannot change the object. Change authority provides object operational authority and all data authority.

***ALL** The user can perform all operations except those limited to the owner or controlled by authorization list management authority.

The user can control the object's existence and specify the security for the object, change the object, and perform basic functions on the object. The user can change ownership of the object.

***USE** Use authority provides object operational authority and read authority.

***EXCLUDE**
Exclude authority prevents the user from accessing the object.

Authorization-list

Specify the name of the authorization list whose authority is used for the side information.

Initiated end (Receiver)

Use the CRTMQMCHL command to define the receiving end of the message channel link with transport type *LU62. Leave the CONNECTION NAME field blank and ensure that the corresponding details match the sending end of the channel. (See "Creating a channel" on page 324 for details of how to do this.)

To enable the initiating end to start the receiving channel, add a routing entry to a subsystem at the initiated end. The subsystem must be the one that allocates the APPC device used in the LU 6.2 sessions and, therefore, it must have a valid communications entry for that device. The routing entry calls the program that starts the receiving end of the message channel.

Use the i5/OS commands (for example, ADDRTGE) to define the end of the link that is initiated by a communication session.

The initiated end panel is shown in Figure Figure 71.

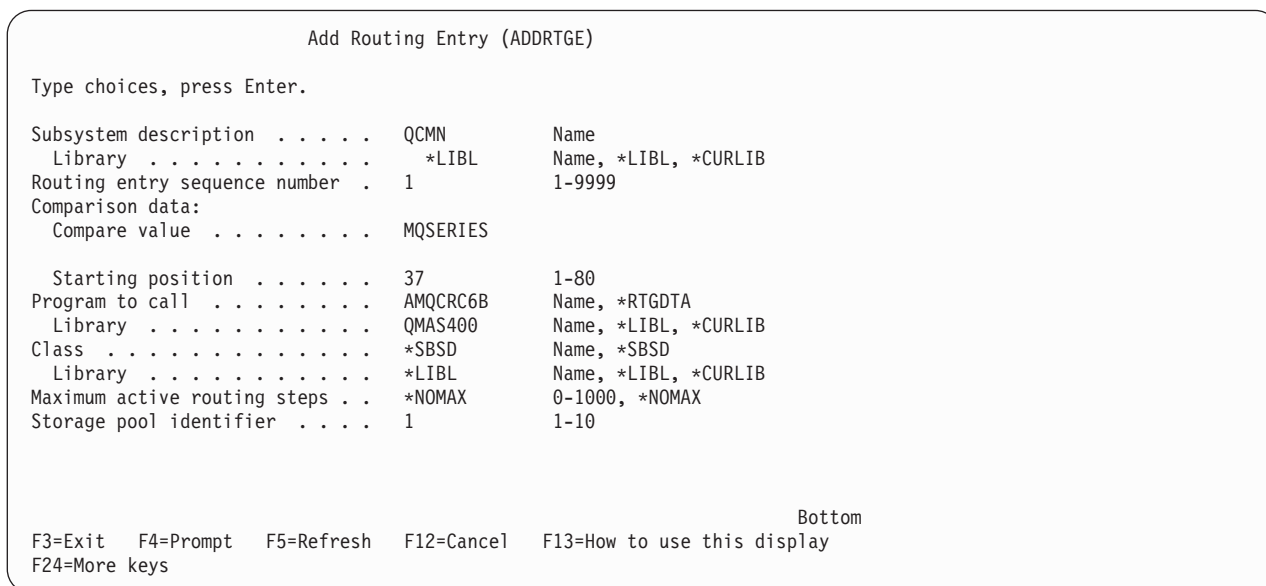


Figure 71. LU 6.2 communication setup panel - initiated end

Subsystem description

The name of your subsystem where this definition resides. Use the i5/OS WRKSBSD command to view and update the appropriate subsystem description for the routing entry.

Routing entry sequence number

A unique number in your subsystem to identify this communication definition. You can use values in the range 1 to 9999.

Comparison data: Compare value

A text string to compare with that received when the session is started by a **Transaction program** parameter, as shown in Figure 70 on page 345. The character string is derived from the Transaction program field of the sender CSI.

Comparison data: Starting position

The character position in the string where the comparison is to start.

Note: The starting position field is the character position in the string for comparison, and this is always 37.

Program to call

The name of the program that runs the inbound message program to be called to start the session.

The program, AMQCRC6A, is called for the default queue manager. This is a program supplied with WebSphere MQ for i5/OS that sets up the environment and then calls AMQCRS6A.

For additional queue managers:

- Each queue manager has a specific LU 6.2 invocable program located in its library. This program is called AMQCRC6B and is automatically generated when the queue manager is created.
- Each queue manager requires a specific routing entry with unique routing data to be added. This routing data should match the **Transaction program** name supplied by the requesting system (see “Initiating end (Sending)” on page 345).

An example of this is shown in Figure Figure 72:

Display Routing Entries

Subsystem description: QCMN Status: ACTIVE System: MY400

Type options, press Enter.
5=Display details

Opt	Seq Nbr	Program	Library	Compare Value	Start Pos
	10	*RTGDTA		'QZSCSRVR'	37
	20	*RTGDTA		'QZRCSRVR'	37
	30	*RTGDTA		'QZHQTRG'	37
	50	*RTGDTA		'QVPPRINT'	37
	60	*RTGDTA		'QNPSERVER'	37
	70	*RTGDTA		'QNMAPPINGD'	37
	80	QNMAREXECD	QSYS	'AREXECD'	37
	90	AMQCRC6A	QMQMBW	'MQSERIES'	37
	100	*RTGDTA		'QTFDWNLD'	37
	150	*RTGDTA		'QMFRQVVR'	37

F3=Exit F9=Display all detailed descriptions F12=Cancel

Figure 72. LU 6.2 communication setup panel - initiated end

In Figure Figure 72 sequence number 90 represents the default queue manager and provides compatibility with configurations from previous releases (that is, V3R2, V3R6, V3R7, and V4R2) of WebSphere MQ for

i5/OS. These releases allow one queue manager only. Sequence numbers 92 and 94 represent two additional queue managers called ALPHA and BETA that are created with libraries QMALPHA and QMBETA.

Note: You can have more than one routing entry for each queue manager by using different routing data. This gives the option of different job priorities depending on the classes used.

Class The name and library of the class used for the steps started through this routing entry. The class defines the attributes of the routing step's running environment and specifies the job priority. An appropriate class entry must be specified. Use, for example, the WRKCLS command to display existing classes or to create a new class. Further information on managing work requests from remote LU 6.2 systems is available in the *i5/OS Programming: Work Management Guide*.

Note on Work Management:

The AMQCRS6A job will not be able to take advantage of the normal i5/OS work management features that are documented in the WebSphere MQ for i5/OS System Administration Guide book because it is not started in the same way as other WebSphere MQ jobs. To change the run-time properties of the LU62 receiver jobs, you can do one of the following:

- Alter the class description that is specified on the routing entry for the AMQCRS6A job
- Change the job description on the communications entry

See the *i5/OS Programming: Work Management Guide* for more information about configuring Communication Jobs.

Example configuration - IBM WebSphere MQ for i5/OS

This chapter gives an example of how to set up communication links from WebSphere MQ for i5/OS to WebSphere MQ products on the following platforms:

- Windows
- AIX
- Compaq Tru64 UNIX
- HP-UX
- Solaris
- Linux
- z/OS, OS/390, or MVS/ESA
- VSE/ESA

First it describes the parameters needed for an LU 6.2 connection, then it describes "Establishing an LU 6.2 connection" on page 355 and "Establishing a TCP connection" on page 360.

Once the connection is established, you need to define some channels to complete the configuration. This is described in "WebSphere MQ for i5/OS configuration" on page 362.

See "Example configuration chapters in this book" on page 101 for background information about this chapter and how to use it.

Configuration parameters for an LU 6.2 connection

Table 35 presents a worksheet listing all the parameters needed to set up communication from i5/OS system to one of the other WebSphere MQ platforms. The worksheet shows examples of the parameters, which have been tested in a working environment, and leaves space for you to fill in your own values. An explanation of the parameter names follows the worksheet. Use the worksheet in this chapter in conjunction with the worksheet in the chapter for the platform to which you are connecting.

Configuration worksheet

Use the following worksheet to record the values you will use for this configuration. Where numbers appear in the Reference column they indicate that the value must match that in the appropriate worksheet elsewhere in this book. The examples that follow in this chapter refer back to the values in the ID column of this table. The entries in the Parameter Name column are explained in “Explanation of terms” on page 353.

Table 35. Configuration worksheet for SNA on an i5/OS system

ID	Parameter Name	Reference	Example Used	User Value
<i>Definition for local node</i>				
1	Local network ID		NETID	
2	Local control point name		AS400PU	
3	LU name		AS400LU	
4	LAN destination address		10005A5962EF	
5	Subsystem description		QCMN	
6	Line description		TOKENRINGL	
7	Resource name		LIN041	
8	Local Transaction Program name		MQSERIES	
<i>Connection to a Windows system</i>				
The values in this section must match those used in Table 13 on page 130, as indicated.				
9	Network ID	2	NETID	
10	Control point name	3	WINNTCP	
11	LU name	5	WINNTLU	
12	Controller description		WINNTCP	
13	Device		WINNTLU	
14	Side information		NTCPIC	
15	Transaction Program	7	MQSERIES	
16	LAN adapter address	9	08005AA5FAB9	
17	Mode	17	#INTER	
<i>Connection to an AIX system</i>				
The values in this section must match those used in Table 17 on page 156, as indicated.				
9	Network ID	1	NETID	
10	Control point name	2	AIXPU	
11	LU name	4	AIXLU	

Table 35. Configuration worksheet for SNA on an i5/OS system (continued)

ID	Parameter Name	Reference	Example Used	User Value
12	Controller description		AIXPU	
13	Device		AIXLU	
14	Side information		AIXCPIC	
15	Transaction Program	6	MQSERIES	
16	LAN adapter address	8	123456789012	
17	Mode	14	#INTER	
Connection to an HP-UX system				
The values in this section must match those used in Table 19 on page 172, as indicated.				
9	Network ID	4	NETID	
10	Control point name	2	HPUXPU	
11	LU name	5	HPUXLU	
12	Controller description		HPUXPU	
13	Device		HPUXLU	
14	Side information		HPUXCPIC	
15	Transaction Program	7	MQSERIES	
16	LAN adapter address	8	100090DC2C7C	
17	Mode	17	#INTER	
Connection to a Solaris system				
The values in this section must match those used in Table 21 on page 195, as indicated.				
9	Network ID	2	NETID	
10	Control point name	3	SOLARPU	
11	LU name	7	SOLARLU	
12	Controller description		SOLARPU	
13	Device		SOLARLU	
14	Side information		SOLCPIC	
15	Transaction Program	8	MQSERIES	
16	LAN adapter address	5	08002071CC8A	
17	Mode	17	#INTER	
Connection to a Linux (x86 platform) system				
The values in this section must match those used in Configuration worksheet for Communications Server for Linux, as indicated.				
9	Network ID	4	NETID	
10	Control point name	2	LINUXPU	
11	LU name	5	LINUXLU	
12	Controller description		LINUXPU	
13	Device		LINUXLU	
14	Side information		LXCPIC	
15	Transaction Program	7	MQSERIES	
16	LAN adapter address	8	08005AC6DF33	

Table 35. Configuration worksheet for SNA on an i5/OS system (continued)

ID	Parameter Name	Reference	Example Used	User Value
17	Mode	6	#INTER	
<i>Connection to an z/OS system</i>				
The values in this section must match those used in Table 27 on page 267, as indicated.				
9	Network ID	2	NETID	
10	Control point name	3	MVSPU	
11	LU name	4	MVSLU	
12	Controller description		MVSPU	
13	Device		MVSLU	
14	Side information		MVSPIC	
15	Transaction Program	7	MQSERIES	
16	LAN adapter address	8	400074511092	
17	Mode	6	#INTER	
<i>Connection to a VSE/ESA system</i>				
The values in this section must match those used in your VSE/ESA system.				
9	Network ID	1	NETID	
10	Control point name	2	VSEPU	
11	LU name	3	VSELU	
12	Controller description		VSEPU	
13	Device		VSELU	
14	Side information		VSEPIC	
15	Transaction Program	4	MQ01	MQ01
16	LAN adapter address	5	400074511092	
17	Mode		#INTER	

Explanation of terms

1 2 3 See “How to find network attributes” on page 354 for the details of how to find the configured values.

4 LAN destination address

The hardware address of the i5/OS system token-ring adapter. You can find the value using the command DSPLIND *Line description* (6).

5 Subsystem description

This is the name of any i5/OS subsystem that will be active while using the queue manager. The name QCMN has been used because this is the i5/OS communications subsystem.

6 Line description

If this has been specified it is indicated in the Description field of the resource Resource name. See “How to find the value of Resource name” on page 354 for details. If the value is not specified you will need to create a line description.

7 Resource name

See “How to find the value of Resource name” on page 354 for details of how to find the configured value.

8 Local Transaction Program name

WebSphere MQ applications trying to converse with this workstation will specify a symbolic name for the program to be run at the receiving end. This will have been defined on the channel definition at the sender. For simplicity, wherever possible use a transaction program name of MQSERIES, or in the case of a connection to VSE/ESA, where the length is limited to 4 bytes, use MQTP.

See Table 34 on page 344 for more information.

12 Controller description

This is an alias for the Control Point name (or Node name) of the partner system. For convenience we have used the actual name of the partner in this example.

13 Device

This is an alias for the LU of the partner system. For convenience we have used the LU name of the partner in this example.

14 Side information

This is the name given to the CPI-C side information profile. You specify your own 8-character name for this.

How to find network attributes:

The local node has been partially configured as part of the i5/OS installation. To display the current network attributes enter the command DSPNETA.

If you need to change these values use the command CHGNETA. An IPL may be required to apply your changes.

```
Display Network Attributes
System: AS400PU
Current system name . . . . . : AS400PU
Pending system name . . . . . :
Local network ID . . . . . : NETID
Local control point name . . . . . : AS400PU
Default local location . . . . . : AS400LU
Default mode . . . . . : BLANK
APPN node type . . . . . : *ENDNODE
Data compression . . . . . : *NONE
Intermediate data compression . . . . . : *NONE
Maximum number of intermediate sessions . . . . . : 200
Route addition resistance . . . . . : 128
Server network ID/control point name . . . . . : NETID NETCP

More...

Press Enter to continue.
F3=Exit F12=Cancel
```

Check that the values for **Local network ID** (1), **Local control point name** (2), and **Default local location** (3), correspond to the values on your worksheet.

How to find the value of Resource name:

Type WRKHDWRSC TYPE(*CMN) and press Enter. The Work with Communication Resources panel is displayed. The value for **Resource name** is found as the Token-Ring Port. It is LIN041 in this example.

```

Work with Communication Resources                               System:  AS400PU
Type options, press Enter.
 2=Edit   4=Remove   5=Work with configuration description
 7=Add configuration description ...

Opt  Resource      Configuration
      CC02         Description  Type  Description
      LIN04         Description  Type  Description
      LIN041       TOKENRINGL  2636  Token-Ring Port

F3=Exit      F5=Refresh  F6=Print  F11=Display resource addresses/statuses
F12=Cancel   F23=More options
Bottom

```

Establishing an LU 6.2 connection

This section describes how to establish an LU 6.2 connection.

Local node configuration

To configure the local node, you need to:

1. Create a line description
2. Add a routing entry

Creating a line description:

1. If the line description has not already been created use the command CRTLINTRN.
2. Specify values for **Line description** (6) and **Resource name** (7).

Create Line Desc (Token-Ring) (CRTLINTRN)

Type choices, press Enter.

Line description	TOKENRINGL	Name
Resource name	LIN041	Name, *NWID
NWI type	*FR	*FR, *ATM
Online at IPL	*YES	*YES, *NO
Vary on wait	*NOWAIT	*NOWAIT, 15-180 (1 second)
Maximum controllers	40	1-256
Attached NWI	*NONE	Name, *NONE

Bottom

F3=Exit F4=Prompt F5=Refresh F10=Additional parameters F12=Cancel
F13=How to use this display F24=More keys
Parameter LIND required. +

Adding a routing entry:

1. Type the command ADDRTGE and press Enter.

Add Routing Entry (ADDRTGE)

Type choices, press Enter.

Subsystem description	QCMN	Name
Library	*LIBL	Name, *LIBL, *CURLIB
Routing entry sequence number . .	1	1-9999
Comparison data:		
Compare value	'MQSERIES'	
Starting position	37	1-80
Program to call	AMQCRC6B	Name, *RTGDTA
Library	QMAS400	Name, * LI BL, *CURLIB
Class	*SBSD	Name, *SBSD
Library	*LIBL	Name, *LIBL, *CURLIB
Maximum active routing steps . .	*NOMAX	0-1000, *NOMAX
Storage pool identifier	1	1-10

Bottom

F3=Exit F4=Prompt F5=Refresh F12=Cancel F13=How to use this display
F24=More keys
Parameter SBSDB required. +

2. Specify your value for **Subsystem description** (5), and the values shown here for **Routing entry sequence number**, **Compare value** (8), **Starting position**, **Program to call**, and the **Library** containing the program to call.
3. Type the command STRSBS *subsystem description* (5) and press Enter.

Connection to partner node

This example is for a connection to a Windows system, but the steps are the same for other nodes. The steps are:

1. Create a controller description.
2. Create a device description.

3. Create CPI-C side information.
4. Add a communications entry for APPC.
5. Add a configuration list entry.

Creating a controller description:

1. At a command line type CRTCTLAPPC and press Enter.

```

Create Ctl Desc (APPC) (CRTCTLAPPC)

Type choices, press Enter.

Controller description . . . . . WINNTCP           Name
Link type . . . . . *LAN                *FAX, *FR, *IDLC,
*LAN...
Online at IPL . . . . . *NO              *YES, *NO

                                                    Bottom
F3=Exit  F4=Prompt  F5=Refresh  F10=Additional parameters  F12=Cancel
F13=How to use this display  F24=More keys
Parameter CTLD required.                                     +

```

2. Specify a value for **Controller description** (12), set **Link type** to *LAN, and set **Online at IPL** to *NO.
3. Press Enter twice, followed by F10.

```

Create Ctl Desc (APPC) (CRTCTLAPPC)

Type choices, press Enter.

Controller description . . . . . > WINNTCP           Name
Link type . . . . . > *LAN                *FAX, *FR, *IDLC, *LAN...
Online at IPL . . . . . > *NO              *YES, *NO
APPN-capable . . . . . *YES                *YES, *NO
Switched line list . . . . . TOKENRINGL      Name
+ for more values
Maximum frame size . . . . . *LINKTYPE      265-16393, 256, 265, 512...
Remote network identifier . . . . . NETID      Name, *NETATR, *NONE, *ANY
Remote control point . . . . . WINNTCP       Name, *ANY
Exchange identifier . . . . .                00000000-FFFFFFFF
Initial connection . . . . . *DIAL           *DIAL, *ANS
Dial initiation . . . . . *LINKTYPE          *LINKTYPE, *IMMED, *DELAY
LAN remote adapter address . . . . . 10005AFC5D83 000000000001-FFFFFFFFFFFF
APPN CP session support . . . . . *YES        *YES, *NO
APPN node type . . . . . *ENDNODE            *ENDNODE, *LENNODE...
APPN transmission group number 1            1-20, *CALC

                                                    More...
F3=Exit  F4=Prompt  F5=Refresh  F12=Cancel  F13=How to use this display
F24=More keys

```

4. Specify values for **Switched line list** (6), **Remote network identifier** (9), **Remote control point** (10), and **LAN remote adapter address** (16).

5. Press Enter.

Creating a device description:

1. Type the command CRTDEVAPPC and press Enter.

```
                Create Device Desc (APPC) (CRTDEVAPPC)

Type choices, press Enter.

Device description . . . . . WINNTLU      Name
Remote location . . . . . WINNTLU      Name
Online at IPL . . . . . *YES           *YES, *NO
Local location . . . . . AS400LU      Name, *NETATR
Remote network identifier . . . . . NETID  Name, *NETATR, *NONE
Attached controller . . . . . WINNTCP   Name
Mode . . . . . *NETATR                Name, *NETATR
                + for more values
Message queue . . . . . QSYSOPR       Name, QSYSOPR
  Library . . . . . *LIBL             Name, *LIBL, *CURLIB
APPN-capable . . . . . *YES           *YES, *NO
Single session:
  Single session capable . . . . . *NO   *NO, *YES
  Number of conversations . . . . .      1-512

                                                    Bottom
F3=Exit  F4=Prompt  F5=Refresh  F10=Additional parameters  F12=Cancel
F13=How to use this display  F24=More keys
Parameter DEVD required.                                                    +
```

2. Specify values for **Device description** (13), **Remote location** (11), **Local location** (3), **Remote network identifier** (9), and **Attached controller** (12).

Note: You can avoid having to create controller and device descriptions manually by taking advantage of i5/OS's auto-configuration service. Consult the i5/OS documentation for details.

Creating CPI-C side information:

1. Type CRTCSI and press F10.

```

Create Comm Side Information (CRTCSI)

Type choices, press Enter.

Side information . . . . . NTCPIC      Name
Library . . . . . *CURLIB      Name, *CURLIB
Remote location . . . . . WINNTLU     Name
Transaction program . . . . . MQSERIES

Text 'description' . . . . . *BLANK

Additional Parameters

Device . . . . . *LOC          Name, *LOC
Local location . . . . . AS400LU     Name, *LOC, *NETATR
Mode . . . . . #INTER          Name, *NETATR
Remote network identifier . . . . . NETID      Name, *LOC, *NETATR, *NONE
Authority . . . . . *LIBCRTAUT      Name, *LIBCRTAUT, *CHANGE...

Bottom
F3=Exit  F4=Prompt  F5=Refresh  F12=Cancel  F13=How to use this display
F24=More keys
Parameter CSI required.

```

2. Specify values for **Side information** (14), **Remote location** (11), **Transaction program** (15), **Local location** (3), **Mode**, and **Remote network identifier** (9).
3. Press Enter.

Adding a communications entry for APPC:

1. At a command line type ADDCMNE and press Enter.

```

Add Communications Entry (ADDCMNE)

Type choices, press Enter.

Subsystem description . . . . . QCMN      Name
Library . . . . . *LIBL          Name, *LIBL, *CURLIB
Device . . . . . WINNTLU         Name, generic*, *ALL...
Remote location . . . . .          Name
Job description . . . . . *USRPRF  Name, *USRPRF, *SBSD
Library . . . . .          Name, *LIBL, *CURLIB
Default user profile . . . . . *NONE  Name, *NONE, *SYS
Mode . . . . . *ANY             Name, *ANY
Maximum active jobs . . . . . *NOMAX  0-1000, *NOMAX

Bottom
F3=Exit  F4=Prompt  F5=Refresh  F12=Cancel  F13=How to use this display
F24=More keys
Parameter SBSD required.

```

2. Specify values for **Subsystem description** (5) and **Device** (13), and press Enter.

Adding a configuration list entry:

1. Type ADDCFGLE *APPNRMT and press F4.

Add Configuration List Entries (ADDCFGLE)

Type choices, press Enter.

```
Configuration list type . . . . > *APPNRMT      *APPNLCL, *APPNRMT...
APPN remote location entry:
Remote location name . . . . . WINNTLU        Name, generic*, *ANY
Remote network identifier . . . NETID         Name, *NETATR, *NONE
Local location name . . . . . AS400LU        Name, *NETATR
Remote control point . . . . . WINNTCP       Name, *NONE
Control point net ID . . . . . NETID         Name, *NETATR, *NONE
Location password . . . . . *NONE
Secure location . . . . . *NO              *YES, *NO
Single session . . . . . *NO              *YES, *NO
Locally controlled session . . *NO          *YES, *NO
Pre-established session . . . *NO          *YES, *NO
Entry 'description' . . . . . *BLANK
Number of conversations . . . 10            1-512
      + for more values

Bottom
F3=Exit  F4=Prompt  F5=Refresh  F12=Cancel  F13=How to use this display
F24=More keys
```

2. Specify values for **Remote location name** (11), **Remote network identifier** (9), **Local location name** (3), **Remote control point** (10), and **Control point net ID** (9).
3. Press Enter.

What next?

The LU 6.2 connection is now established. You are ready to complete the configuration. Go to “WebSphere MQ for i5/OS configuration” on page 362.

Establishing a TCP connection

If TCP is already configured there are no extra configuration tasks. The following panels guide you through the steps that may be required if TCP/IP is not configured.

Adding a TCP/IP interface

1. At a command line type ADDTCPIFC and press Enter.

```

Add TCP/IP Interface (ADDTCPIFC)

Type choices, press Enter.

Internet address . . . . . 19.22.11.55
Line description . . . . . TOKENRINGL Name, *LOOPBACK
Subnet mask . . . . . 255.255.0.0
Type of service . . . . . *NORMAL *MINDELAY, *MAXTHRPUT..
Maximum transmission unit . . . *LIND 576-16388, *LIND
Autostart . . . . . *YES *YES, *NO
PVC logical channel identifier
+ for more values
X.25 idle circuit timeout . . . 60 1-600
X.25 maximum virtual circuits . 64 0-64
X.25 DDN interface . . . . . *NO *YES, *NO
TRLAN bit sequencing . . . . . *MSB *MSB, *LSB

Bottom
F3=Exit F4=Prompt F5=Refresh F12=Cancel F13=How to use this display
F24=More keys

```

- Specify this machine's **Internet address** and **Line description**, and a **Subnet mask**.
- Press Enter.

Adding a TCP/IP loopback interface

- At a command line type ADDTCPIFC and press Enter.

```

Add TCP Interface (ADDTCPIFC)

Type choices, press Enter.

Internet address . . . . . 127.0.0.1
Line description . . . . . *LOOPBACK Name, *LOOPBACK
Subnet mask . . . . . 255.0.0.0
Type of service . . . . . *NORMAL *MINDELAY, *MAXTHRPUT..
Maximum transmission unit . . . *LIND 576-16388, *LIND
Autostart . . . . . *YES *YES, *NO
PVC logical channel identifier
+ for more values
X.25 idle circuit timeout . . . 60 1-600
X.25 maximum virtual circuits . 64 0-64
X.25 DDN interface . . . . . *NO *YES, *NO
TRLAN bit sequencing . . . . . *MSB *MSB, *LSB

Bottom
F3=Exit F4=Prompt F5=Refresh F12=Cancel F13=How to use this display
F24=More keys

```

- Specify the values for **Internet address**, **Line description**, and **Subnet mask**.

Adding a default route

- At a command line type ADDTCPRTE and press Enter.

Add TCP Route (ADDTCP RTE)

Type choices, press Enter.

```
Route destination . . . . . *DFTRROUTE
Subnet mask . . . . . *NONE
Type of service . . . . . *NORMAL          *MINDELAY, *MAXTHRPUT.
Next hop . . . . . 19.2.3.4
Maximum transmission unit . . . 576          576-16388, *IFC
```

Bottom

F3=Exit F4=Prompt F5=Refresh F12=Cancel F13=How to use this display
F24=More keys
Command prompting ended when user pressed F12.

2. Fill in with values appropriate to your network and press Enter to create a default route entry.

What next?

The TCP connection is now established. You are ready to complete the configuration. Go to “WebSphere MQ for i5/OS configuration.”

WebSphere MQ for i5/OS configuration

Start the TCP channel listener using the command STRMQMLSR.

Start any sender channel using the command STRMQMCHL
CHLNAME(*channel_name*).

Use the WRKMQMQ command to display the WebSphere MQ configuration menu.

Note: AMQ* errors are placed in the log relating to the job that found the error. Use the WRKACTJOB command to display the list of jobs. Under the subsystem name QSYSWRK, locate the job and enter 5 against it to work with that job. WebSphere MQ logs are prefixed ‘AMQ’.

Basic configuration

1. First you need to create a queue manager. To do this, type CRTMQM and press Enter.

```

Create Message Queue Manager (CRTMQM)

Type choices, press Enter.

Message Queue Manager name . . .

Text 'description' . . . . . *BLANK

Trigger interval . . . . . 999999999 0-999999999
Undelivered message queue . . . *NONE

Default transmission queue . . . *NONE

Maximum handle limit . . . . . 256 1-999999999
Maximum uncommitted messages . . 1000 1-10000
Default Queue manager . . . . . *NO *YES, *NO

Bottom
F3=Exit F4=Prompt F5=Refresh F12=Cancel F13=How to use this display
F24=More keys

```

2. In the **Message Queue Manager name** field, type AS400. In the **Undelivered message queue** field, type DEAD.LETTER.QUEUE.
3. Press Enter.
4. Now start the queue manager by entering STRMQM MQMNAME(AS400).
5. Create the undelivered message queue using the following parameters. (For details and an example refer to “Defining a queue” on page 367.)

```

Local Queue
Queue name : DEAD.LETTER.QUEUE
Queue type : *LCL

```

Channel configuration

This section details the configuration to be performed on the i5/OS queue manager to implement the channel described in Figure 32 on page 101.

Examples are given for connecting WebSphere MQ for i5/OS and WebSphere MQ for Windows. If you wish connect to WebSphere MQ on another platform, use the appropriate values from the table in place of those for Windows

Note:

1. The words in **bold** are user-specified and reflect the names of WebSphere MQ objects used throughout these examples. If you change the names used here, ensure that you also change the other references made to these objects throughout this book. All others are keywords and should be entered as shown.
2. The WebSphere MQ channel ping command (PNGMQMCHL) runs interactively, whereas starting a channel causes a batch job to be submitted. If a channel ping completes successfully but the channel will not start, this indicates that the network and WebSphere MQ definitions are probably correct, but that the i5/OS environment for the batch job is not. For example, make sure that QSYS2 is included in the system portion of the library list and not just your personal library list.

For details and examples of how to create the objects listed refer to “Defining a queue” on page 367 and “Defining a channel” on page 367.

Table 36. Configuration worksheet for WebSphere MQ for i5/OS

ID	Parameter Name	Reference	Example Used	User Value
<i>Definition for local node</i>				
A	Queue Manager Name		AS400	
B	Local queue name		AS400.LOCALQ	
<i>Connection to WebSphere MQ for Windows</i>				
The values in this section of the table must match those used in Table 14 on page 146, as indicated.				
C	Remote queue manager name	A	WINNT	
D	Remote queue name		WINNT.REMOTEQ	
E	Queue name at remote system	B	WINNT.LOCALQ	
F	Transmission queue name		WINNT	
G	Sender (SNA) channel name		AS400.WINNT.SNA	
H	Sender (TCP/IP) channel name		AS400.WINNT.TCP	
I	Receiver (SNA) channel name	G	WINNT.AS400.SNA	
J	Receiver (TCP/IP) channel name	H	WINNT.AS400.TCP	
<i>Connection to WebSphere MQ for AIX</i>				
The values in this section of the table must match those used in Table 18 on page 168, as indicated.				
C	Remote queue manager name	A	AIX	
D	Remote queue name		AIX.REMOTEQ	
E	Queue name at remote system	B	AIX.LOCALQ	
F	Transmission queue name		AIX	
G	Sender (SNA) channel name		AS400.AIX.SNA	
H	Sender (TCP/IP) channel name		AS400.AIX.TCP	
I	Receiver (SNA) channel name	G	AIX.AS400.SNA	
J	Receiver (TCP) channel name	H	AIX.AS400.TCP	
<i>Connection to MQSeries for Compaq Tru64 UNIX</i>				
The values in this section of the table must match those used in your Compaq Tru64 UNIX system.				
C	Remote queue manager name	A	DECUX	
D	Remote queue name		DECUX.REMOTEQ	
E	Queue name at remote system	B	DECUX.LOCALQ	
F	Transmission queue name		DECUX	
H	Sender (TCP) channel name		DECUX.AS400.TCP	
J	Receiver (TCP) channel name	H	AS400.DECUX.TCP	
<i>Connection to WebSphere MQ for HP-UX</i>				
The values in this section of the table must match those used in Table 20 on page 191, as indicated.				
C	Remote queue manager name	A	HPUX	
D	Remote queue name		HPUX.REMOTEQ	
E	Queue name at remote system	B	HPUX.LOCALQ	
F	Transmission queue name		HPUX	
G	Sender (SNA) channel name		AS400.HPUX.SNA	
H	Sender (TCP) channel name		AS400.HPUX.TCP	

Table 36. Configuration worksheet for WebSphere MQ for i5/OS (continued)

ID	Parameter Name	Reference	Example Used	User Value
I	Receiver (SNA) channel name	G	HPUX.AS400.SNA	
J	Receiver (TCP) channel name	H	HPUX.AS400.TCP	
Connection to WebSphere MQ for Solaris				
The values in this section of the table must match those used in Table 22 on page 212, as indicated.				
C	Remote queue manager name	A	SOLARIS	
D	Remote queue name		SOLARIS.REMOTEQ	
E	Queue name at remote system	B	SOLARIS.LOCALQ	
F	Transmission queue name		SOLARIS	
G	Sender (SNA) channel name		AS400.SOLARIS.SNA	
H	Sender (TCP/IP) channel name		AS400.SOLARIS.TCP	
I	Receiver (SNA) channel name	G	SOLARIS.AS400.SNA	
J	Receiver (TCP/IP) channel name	H	SOLARIS.AS400.TCP	
Connection to WebSphere MQ for Linux				
The values in this section of the table must match those used in Table 24 on page 235, as indicated.				
C	Remote queue manager name	A	LINUX	
D	Remote queue name		LINUX.REMOTEQ	
E	Queue name at remote system	B	LINUX.LOCALQ	
F	Transmission queue name		LINUX	
G	Sender (SNA) channel name		AS400.LINUX.SNA	
H	Sender (TCP/IP) channel name		AS400.LINUX.TCP	
I	Receiver (SNA) channel name	G	LINUX.AS400.SNA	
J	Receiver (TCP/IP) channel name	H	LINUX.AS400.TCP	
Connection to WebSphere MQ for z/OS				
The values in this section of the table must match those used in Table 28 on page 272, as indicated.				
C	Remote queue manager name	A	MVS	
D	Remote queue name		MVS.REMOTEQ	
E	Queue name at remote system	B	MVS.LOCALQ	
F	Transmission queue name		MVS	
G	Sender (SNA) channel name		AS400.MVS.SNA	
H	Sender (TCP) channel name		AS400.MVS.TCP	
I	Receiver (SNA) channel name	G	MVS.AS400.SNA	
J	Receiver (TCP) channel name	H	MVS.AS400.TCP	
Connection to MQSeries for VSE/ESA				
The values in this section of the table must match those used in your VSE/ESA system.				
C	Remote queue manager name	A	VSE	
D	Remote queue name		VSE.REMOTEQ	
E	Queue name at remote system	B	VSE.LOCALQ	
F	Transmission queue name		VSE	
G	Sender channel name		AS400.VSE.SNA	

Table 36. Configuration worksheet for WebSphere MQ for i5/OS (continued)

ID	Parameter Name	Reference	Example Used	User Value
I	Receiver channel name	G	VSE.AS400.SNA	

WebSphere MQ for i5/OS sender-channel definitions using SNA:

```

Local Queue
    Queue name : WINNT                F
    Queue type : *LCL
    Usage      : *TMQ

Remote Queue
    Queue name : WINNT.REMOTEQ        D
    Queue type : *RMT
    Remote queue : WINNT.LOCALQ       E
Remote Queue Manager : WINNT         C
    Transmission queue : WINNT        F

Sender Channel
    Channel Name : AS400.WINNT.SNA    G
    Channel Type : *SDR
    Transport type : *LU62
    Connection name : WINNTPIC        14
    Transmission queue : WINNT        F
    
```

WebSphere MQ for i5/OS receiver-channel definitions using SNA:

```

Local Queue
    Queue name : AS400.LOCALQ         B
    Queue type : *LCL

Receiver Channel
    Channel Name : WINNT.AS400.SNA    I
    Channel Type : *RCVR
    Transport type : *LU62
    
```

WebSphere MQ for i5/OS sender-channel definitions using TCP:

```

Local Queue
    Queue name : WINNT                F
    Queue type : *LCL
    Usage      : *TMQ

Remote Queue
    Queue name : WINNT.REMOTEQ        D
    Queue type : *RMT
    Remote queue : WINNT.LOCALQ       E
Remote Queue Manager : WINNT         C
    Transmission queue : WINNT        F

Sender Channel
    Channel Name : AS400.WINNT.TCP    H
    Channel Type : *SDR
    Transport type : *TCP
    Connection name : WINNT.tcpip.hostname
    Transmission queue : WINNT        F
    
```

WebSphere MQ for i5/OS receiver-channel definitions using TCP:

```

Local Queue
    Queue name : AS400.LOCALQ         B
    Queue type : *LCL
    
```

```
Receiver Channel
Channel Name : WINNT.AS400.TCP      J
Channel Type : *RCVR
Transport type : *TCP
```

Defining a queue

Type CRTMQMQ on the command line.

```
                Create MQM Queue (CRTMQMQ)
Type choices, press Enter.
Queue name . . . . .
Queue type . . . . .          *ALS, *LCL, *RMT

                                                Bottom
F3=Exit  F4=Prompt  F5=Refresh  F12=Cancel  F13=How to use this display
F24=More keys
Parameter QNAME required.
```

Fill in the two fields of this panel and press Enter. This causes another panel to appear, with entry fields for the other parameters you have. Defaults can be taken for all other queue attributes.

Defining a channel

Type CRTMQMCHL on the command line.

```
                Create MQM Channel (CRTMQMCHL)

Type choices, press Enter.

Channel name . . . . .
Channel type . . . . .          *RCVR, *SDR, *SVR, *RQSTR

                                                                    Bottom
F3=Exit  F4=Prompt  F5=Refresh  F12=Cancel  F13=How to use this display
F24=More keys
Parameter CHLNAME required.
```

Fill in the two fields of this panel and press Enter. Another panel is displayed on which you can specify the values for the other parameters given earlier. Defaults can be taken for all other channel attributes.

Message channel planning example for WebSphere MQ for i5/OS

This chapter provides a detailed example of how to connect two i5/OS queue managers together so that messages can be sent between them. The example illustrates the preparations needed to allow an application using queue manager QM1 to put messages on a queue at queue manager QM2. An application running on QM2 can retrieve these messages, and send responses to a reply queue on QM1.

The example illustrates the use of TCP/IP connections. The example assumes that channels are to be triggered to start when the first message arrives on the transmission queue they are servicing. You must start the channel initiator in order for triggering to work. To do this, use the STRMQMCHLI command.

This example uses SYSTEM.CHANNEL.INITQ as the initiation queue. This queue is already defined by WebSphere MQ. You can use a different initiation queue, but you will have to define it yourself and specify the name of the queue when you start the channel initiator.

What the example shows

The example uses the WebSphere MQ for i5/OS command language.

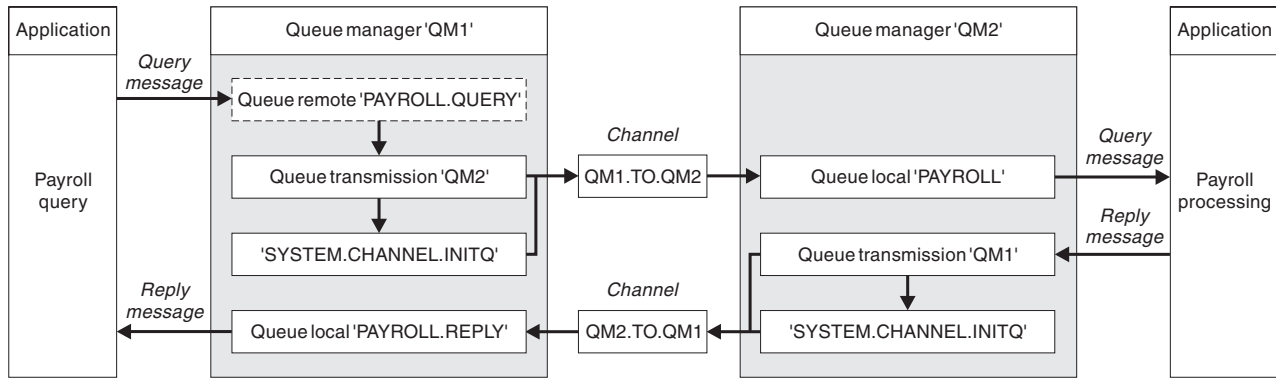


Figure 73. The message channel example for WebSphere MQ for i5/OS

It involves a payroll query application connected to queue manager QM1 that sends payroll query messages to a payroll processing application running on queue manager QM2. The payroll query application needs the replies to its queries sent back to QM1. The payroll query messages are sent from QM1 to QM2 on a sender-receiver channel called QM1.TO.QM2, and the reply messages are sent back from QM2 to QM1 on another sender-receiver channel called QM2.TO.QM1. Both of these channels are triggered to start as soon as they have a message to send to the other queue manager.

The payroll query application puts a query message to the remote queue "PAYROLL.QUERY" defined on QM1. This remote queue definition resolves to the local queue "PAYROLL" on QM2. In addition, the payroll query application specifies that the reply to the query is sent to the local queue "PAYROLL.REPLY" on QM1. The payroll processing application gets messages from the local queue "PAYROLL" on QM2, and sends the replies to wherever they are required; in this case, local queue "PAYROLL.REPLY" on QM1.

Both queue managers are assumed to be running on i5/OS. In the example definitions, QM1 has a host address of 9.20.9.31 and is listening on port 1411, and QM2 has a host address of 9.20.9.32 and is listening on port 1412. The example assumes that these are already defined on your i5/OS system, and are available for use.

The object definitions that need to be created on QM1 are:

- Remote queue definition, PAYROLL.QUERY
- Transmission queue definition, QM2 (default=remote queue manager name)
- Sender channel definition, QM1.TO.QM2
- Receiver channel definition, QM2.TO.QM1
- Reply-to queue definition, PAYROLL.REPLY

The object definitions that need to be created on QM2 are:

- Local queue definition, PAYROLL
- Transmission queue definition, QM1 (default=remote queue manager name)
- Sender channel definition, QM2.TO.QM1
- Receiver channel definition, QM1.TO.QM2

The connection details are supplied in the CONNAME attribute of the sender channel definitions.

You can see a diagram of the arrangement in Figure 73 on page 369.

Queue manager QM1 example

The following object definitions allow applications connected to queue manager QM1 to send request messages to a queue called PAYROLL on QM2, and to receive replies on a queue called PAYROLL.REPLY on QM1.

All the object definitions have been provided with the TEXT attributes. The other attributes supplied are the minimum required to make the example work. The attributes that are not supplied take the default values for queue manager QM1.

Run the following commands on queue manager QM1:

Remote queue definition

The CRTMQMQ command with the following attributes:

QNAME	'PAYROLL.QUERY'
QTYPE	*RMT
TEXT	'Remote queue for QM2'
PUTENBL	*YES
TMQNAME	'QM2' (default = remote queue manager name)
RMTQNAME	'PAYROLL'
RMTMQMNAME	'QM2'

Note: The remote queue definition is not a physical queue, but a means of directing messages to the transmission queue, QM2, so that they can be sent to queue manager QM2.

Transmission queue definition

The CRTMQMQ command with the following attributes:

QNAME	QM2
QTYPE	*LCL
TEXT	'Transmission queue to QM2'
USAGE	*TMQ
PUTENBL	*YES
GETENBL	*YES
TRGENBL	*YES
TRGTYPE	*FIRST
INITQNAME	SYSTEM.CHANNEL.INITQ
TRIGDATA	QM1.TO.QM2

When the first message is put on this transmission queue, a trigger message is sent to the initiation queue, SYSTEM.CHANNEL.INITQ. The channel initiator gets the message from the initiation queue and starts the channel identified in the named process.

Sender channel definition

The CRTMQMCHL command with the following attributes:

CHLNAME	QM1.TO.QM2
CHLTYPE	*SDR
TRPTYPE	*TCP
TEXT	'Sender channel to QM2'
TMQNAME	QM2
CONNNAME	'9.20.9.32(1412)'

Receiver channel definition

The CRTMQMCHL command with the following attributes:

CHLNAME	QM2.TO.QM1
CHLTYPE	*RCVR
TRPTYPE	*TCP
TEXT	'Receiver channel from QM2'

Reply-to queue definition

The CRTMQMQ command with the following attributes:

QNAME	PAYROLL.REPLY
QTYPE	*LCL
TEXT	'Reply queue for replies to query messages sent to QM2'
PUTENBL	*YES
GETENBL	*YES

The reply-to queue is defined as PUT(ENABLED). This ensures that reply messages can be put to the queue. If the replies cannot be put to the reply-to queue, they are sent to the dead-letter queue on QM1 or, if this queue is not available, remain on transmission queue QM1 on queue manager QM2. The queue has been defined as GET(ENABLED) to allow the reply messages to be retrieved.

Queue manager QM2 example

The following object definitions allow applications connected to queue manager QM2 to retrieve request messages from a local queue called PAYROLL, and to put replies to these request messages to a queue called PAYROLL.REPLY on queue manager QM1.

You do not need to provide a remote queue definition to enable the replies to be returned to QM1. The message descriptor of the message retrieved from local queue PAYROLL contains both the reply-to queue and the reply-to queue manager names. Therefore, as long as QM2 can resolve the reply-to queue manager name to that of a transmission queue on queue manager QM2, the reply message can be sent. In this example, the reply-to queue manager name is QM1 and so queue manager QM2 simply requires a transmission queue of the same name.

All the object definitions have been provided with the TEXT attribute and are the minimum required to make the example work. The attributes that are not supplied take the default values for queue manager QM2.

Run the following commands on queue manager QM2:

Local queue definition

The CRTMQMQ command with the following attributes:

QNAME	PAYROLL
QTYPE	*LCL
TEXT	'Local queue for QM1 payroll details'
PUTENBL	*YES
GETENBL	*YES

This queue is defined as PUT(ENABLED) and GET(ENABLED) for the same reason as the reply-to queue definition on queue manager QM1.

Transmission queue definition

The CRTMQMQ command with the following attributes:

QNAME	QM1
QTYPE	*LCL
TEXT	'Transmission queue to QM1'
USAGE	*TMQ
PUTENBL	*YES
GETENBL	*YES
TRGENBL	*YES
TRGTYPE	*FIRST
INITQNAME	SYSTEM.CHANNEL.INITQ
TRIGDATA	QM2.TO.QM1

When the first message is put on this transmission queue, a trigger message is sent to the initiation queue, SYSTEM.CHANNEL.INITQ. The channel initiator gets the message from the initiation queue and starts the channel identified in the trigger data.

Sender channel definition

The CRTMQMCHL command with the following attributes:

CHLNAME	QM2.TO.QM1
CHLTYPE	*SDR
TRPTYPE	*TCP
TEXT	'Sender channel to QM1'
TMQNAME	QM1
CONNNAME	'9.20.9.31(1411)'

Receiver channel definition

The CRTMQMCHL command with the following attributes:

CHLNAME	QM1.TO.QM2
CHLTYPE	*RCVR
TRPTYPE	*TCP
TEXT	'Receiver channel from QM1'

Running the example

When you have created the required objects, you must:

- Start the channel initiator for both queue managers
- Start the listener for both queue managers

The applications can then send messages to each other. The channels are triggered to start by the first message arriving on each transmission queue, so you do not need to issue the STRMQMCHL command.

For details about starting a channel initiator and a listener see "Monitoring and controlling channels on i5/OS" on page 321.

Expanding this example

This example can be expanded by:

- Adding more queue and channel definitions to allow other applications to send messages between the two queue managers.
- Adding user exit programs on the channels to allow for link encryption, security checking, or additional message processing.
- Using queue manager aliases and reply-to queue aliases to understand more about how these can be used in the organization of your queue manager network.

For a version of this example that uses MQSC commands, see “Message channel planning example for z/OS” on page 275.

Chapter 6. Further intercommunication considerations

Channel-exit programs

This chapter discusses WebSphere MQ channel-exit programs. This is product-sensitive programming interface information. The following topics are covered:

- “What are channel-exit programs?”
- “Writing and compiling channel-exit programs” on page 394

Message channel agents (MCAs) can also call data-conversion exits; these are discussed in the WebSphere MQ Application Programming Guide.

What are channel-exit programs?

Channel-exit programs are called at defined places in the processing carried out by MCA programs.

Some of these user-exit programs work in complementary pairs. For example, if a user-exit program is called by the sending MCA to encrypt the messages for transmission, the complementary process must be functioning at the receiving end to reverse the process.

The different types of channel-exit program are described below. Table 37 shows the types of channel exit that are available for each channel type.

Table 37. Channel exits available for each channel type

Channel Type	Message exit	Message- retry exit	Receive exit	Security exit	Send exit	Auto-definition exit
Sender channel	Yes		Yes	Yes	Yes	
Server channel	Yes		Yes	Yes	Yes	
Cluster- sender channel	Yes		Yes	Yes	Yes	Yes
Receiver channel	Yes	Yes	Yes	Yes	Yes	Yes
Requester channel	Yes	Yes	Yes	Yes	Yes	
Cluster- receiver channel	Yes	Yes	Yes	Yes	Yes	Yes
Client- connection channel			Yes	Yes	Yes	
Server- connection channel			Yes	Yes	Yes	Yes

Notes:

1. On z/OS, the auto-definition exit applies to cluster-sender and cluster-receiver channels only.

If you are going to run channel exits on a client, you cannot use the MQSERVER environment variable. Instead, create and reference a client channel definition table as described in the WebSphere MQ Clients book.

Processing overview

On startup, the MCAs exchange a startup dialog to synchronize processing. Then they switch to a data exchange that includes the security exits; these must end successfully for the startup phase to complete and to allow messages to be transferred.

The security check phase is a loop, as shown in Figure 74.

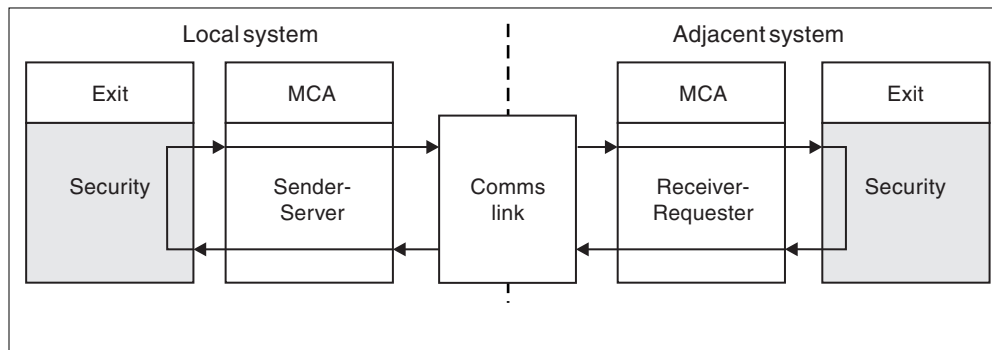


Figure 74. Security exit loop

During the message transfer phase, the sending MCA gets messages from a transmission queue, calls the message exit, calls the send exit, and then sends the message to the receiving MCA, as shown in Figure 75 on page 377.

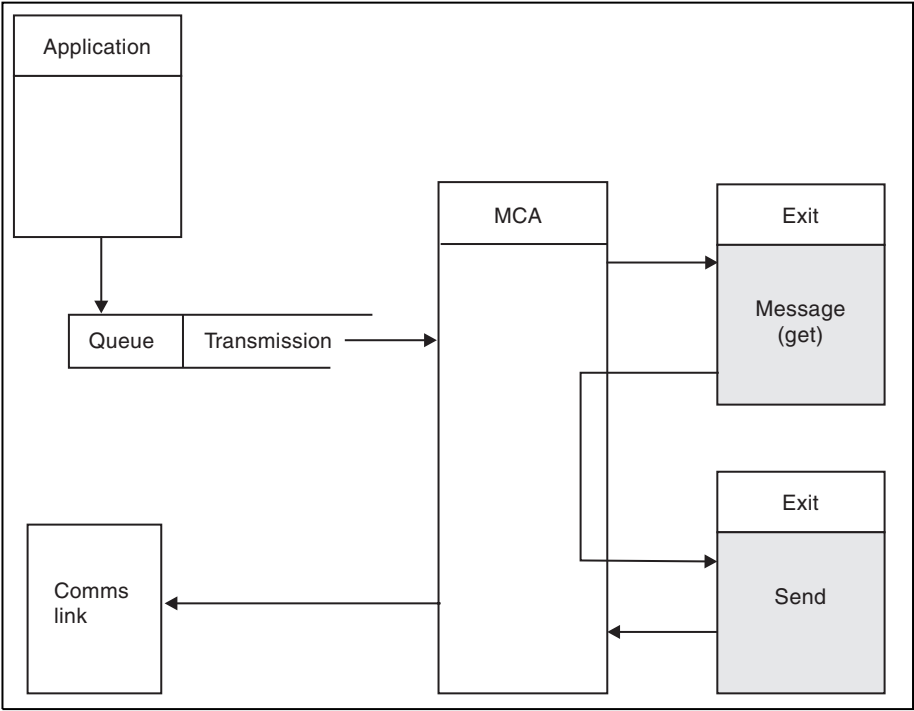


Figure 75. Example of a send exit at the sender end of message channel

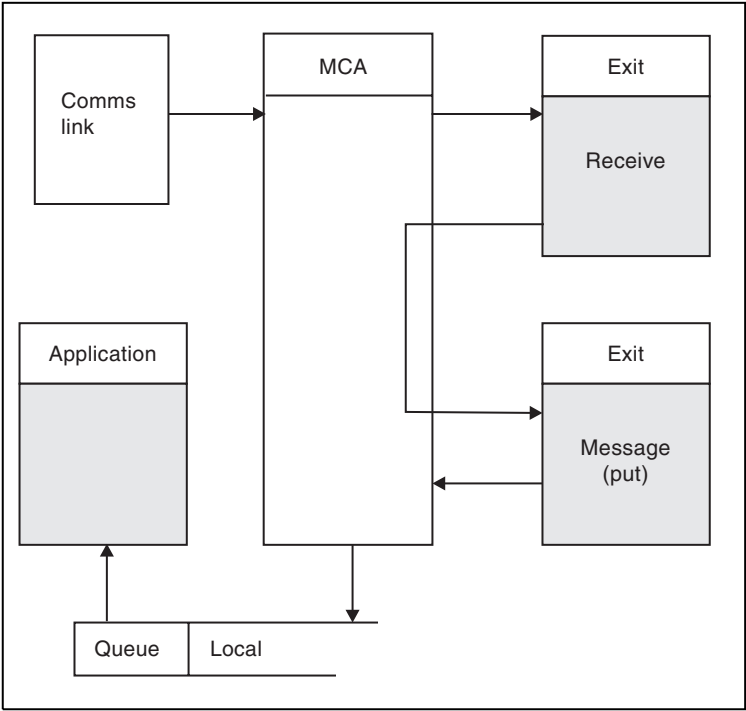


Figure 76. Example of a receive exit at the receiver end of message channel

The receiving MCA receives a message from the communications link, calls the receive exit, calls the message exit, and then puts the message on the local queue, as shown in Figure 76 on page 377. (The receive exit can be called more than once before the message exit is called.)

Channel security exit programs

You can use security exit programs to verify that the partner at the other end of a channel is genuine. To specify that a channel must use a security exit, specify the exit name in the SCYEXIT field of the channel definition.

A security exit must be written in C.

Channel security exit programs are called at the following places in an MCA's processing cycle:

- At MCA initiation and termination.
- Immediately after the initial data negotiation is finished on channel startup. The receiver or server end of the channel may initiate a security message exchange with the remote end by providing a message to be delivered to the security exit at the remote end. It may also decline to do so. The exit program is re-invoked to process any security message received from the remote end.
- Immediately after the initial data negotiation is finished on channel startup. The sender or requester end of the channel processes a security message received from the remote end, or initiates a security exchange when the remote end cannot. The exit program is re-invoked to process all subsequent security messages that may be received.

A requester channel never gets called with MQXCC_INIT_SEC. The channel notifies the server that it has a security exit program, and the server then has the opportunity to initiate a security exit. If it does not have one, it sends a null security flow to allow the requester to call its exit program.

Note: You are recommended to avoid sending zero-length security messages.

Examples of the data exchanged by security-exit programs are illustrated in figures Figure 77 on page 379 through Figure 81 on page 383. These examples show the sequence of events that occur involving the receiver's security exit (left-hand column) and the sender's security exit (right-hand column). Successive rows in the figures represent the passage of time. In some cases, the events at the receiver and sender are not correlated, and therefore can occur at the same time or at different times. In other cases, an event at one exit program results in a complementary event occurring later at the other exit program. For example, in Figure 77 on page 379:

1. The receiver and sender are each invoked with MQXR_INIT, but these invocations are not correlated and can therefore occur at the same time or at different times.
2. The receiver is next invoked with MQXR_INIT_SEC, but returns MQXCC_OK which requires no complementary event at the sender exit.
3. The sender is next invoked with MQXR_INIT_SEC. This is not correlated with the invocation of the receiver with MQXR_INIT_SEC. The sender returns MQXCC_SEND_SEC_MSG, which causes a complementary event at the receiver exit.

4. The receiver is subsequently invoked with MQXR_SEC_MSG, and returns MQXCC_SEND_SEC_MSG, which causes a complementary event at the sender exit.
5. The sender is subsequently invoked with MQXR_SEC_MSG, and returns MQXCC_OK which requires no complementary event at the receiver exit.

Receiver exit	Sender exit
Invoked with MQXR_INIT Responds with MQXCC_OK	Invoked with MQXR_INIT Responds with MQXCC_OK
Invoked with MQXR_INIT_SEC Responds with MQXCC_OK	
	Invoked with MQXR_INIT_SEC Responds with MQXCC_SEND_SEC_MSG
Invoked with MQXR_SEC_MSG Responds with MQXCC_SEND_SEC_MSG	
<i>Message transfer begins</i>	Invoked with MQXR_SEC_MSG Responds with MQXCC_OK

Figure 77. Sender-initiated exchange with agreement

Receiver exit	Sender exit
Invoked with MQXR_INIT Responds with MQXCC_OK	Invoked with MQXR_INIT Responds with MQXCC_OK
Invoked with MQXR_INIT_SEC Responds with MQXCC_OK	
	Invoked with MQXR_INIT_SEC Responds with MQXCC_SEND_SEC_MSG
Invoked with MQXR_SEC_MSG Responds with MQXCC_OK	
	Invoked with MQXR_SEC_MSG Responds with MQXCC_SUPPRESS_FUNCTION <i>Channel closes</i>
Invoked with MQXR_TERM Responds with MQXCC_OK	Invoked with MQXR_TERM Responds with MQXCC_OK

Figure 78. Sender-initiated exchange with no agreement

Receiver exit	Sender exit
Invoked with MQXR_INIT Responds with MQXCC_OK	Invoked with MQXR_INIT Responds with MQXCC_OK
Invoked with MQXR_INIT_SEC Responds with MQXCC_SEND_SEC_MSG	
	Invoked with MQXR_SEC_MSG Responds with MQXCC_SEND_SEC_MSG
Invoked with MQXR_SEC_MSG Responds with MQXCC_OK <i>Message transfer begins</i>	
Invoked with MQXR_TERM Responds with MQXCC_OK	Invoked with MQXR_TERM Responds with MQXCC_OK

Figure 79. Receiver-initiated exchange with agreement

Receiver exit	Sender exit
Invoked with MQXR_INIT Responds with MQXCC_OK	Invoked with MQXR_INIT Responds with MQXCC_OK
Invoked with MQXR_INIT_SEC Responds with MQXCC_SEND_SEC_MSG	
	Invoked with MQXR_SEC_MSG Responds with MQXCC_OK
Invoked with MQXR_SEC_MSG Responds with MQXCC_SUPPRESS_FUNCTION	
<i>Channel closes</i>	

Figure 80. Receiver-initiated exchange with no agreement

Figure Figure 81 on page 383 illustrates the use of security exits in a client connection, using the WebSphere MQ Object Authority Manager to authenticate a user. Either SecurityParmsPtr or SecurityParmsOffset is set in the MQCNO structure on the client and there are security exits at both ends of the channel. After the normal security message exchange has ended, as described for figure Figure 77 on page 379, and the channel is ready to run, the MQCSP structure accessed from the MQCXP SecurityParms field is passed to the security exit on the client. The exit type is set to MQXR_SEC_PARMS. The security exit can elect to do nothing to the user identifier and password, or it can alter either or both of them. The data returned from the exit is then sent to the server-connection end of the channel. The MQCSP structure is rebuilt on the server-connection end of the channel and is passed to the server-connection security exit accessed from the MQCXP SecurityParms field. The security exit receives and processes this data. This processing will normally be to reverse any change made to the userid and password fields in the client exit. The resulting MQCSP structure is referenced using SecurityParmsPtr in the MQCNO structure on the queue manager system.

If SecurityParmsPtr or SecurityParmsOffset are set in the MQCNO structure and there is a security exit at only one end of the channel, the security exit will receive and process the MQCSP structure. Actions such as encryption are inappropriate for a single user exit, as there is no exit to perform the complementary action.

If SecurityParmsPtr and SecurityParmsOffset are not set in the MQCNO structure and there is a security exit at either or both ends of the channel, the security exit or exits will be called. Either security exit can return its own MQCSP structure, addressed through the SecurityParmsPtr; the security exit is not called again until it is terminated (ExitReason of MQXR_TERM). The exit writer can free the memory used for the MQCSP at that stage.

Server-connection exit	Client-connection exit
Invoked with MQXR_INIT Responds with MQXCC_OK	Invoked with MQXR_INIT Responds with MQXCC_OK
Invoked with MQXR_INIT_SEC Responds with MQXCC_OK	
	Invoked with MQXR_INIT_SEC Responds with MQXCC_SEND_SEC_MSG
Invoked with MQXR_SEC_MSG Responds with MQXCC_SEND_SEC_MSG	
	Invoked with MQXR_SEC_MSG Responds with MQXCC_OK
	Invoked with MQXR_SEC_PARMS
Invoked with MQXR_SEC_PARMS	
<i>Data transfer begins</i>	

Figure 81. Client connection-initiated exchange with agreement for client connection using security parameters

The channel security exit program is passed an agent buffer containing the security data, excluding any transmission headers, generated by the security exit. This may be any suitable data so that either end of the channel is able to perform security validation.

The security exit program at both the sending and receiving end of the message channel may return one of four response codes to any call:

- Security exchange ended with no errors
- Suppress the channel and close down

Note:

1. The channel security exits usually work in pairs. When you define the appropriate channels, make sure that compatible exit programs are named for both ends of the channel.

2. In i5/OS, security exit programs that have been compiled with “Use adopted authority” (USEADPAUT=*YES) have the ability to adopt QMQM or QMQMADM authority. Take care that the exit does not exploit this feature to pose a security risk to your system.
3. On an SSL channel on which the other end of the channel provides a certificate, the security exit receives the Distinguished Name of the subject of this certificate in the MQCD field accessed by SSLPeerNamePtr and the Distinguished Name of the issuer in the MQCXP field accessed by SSLRemCertIssNamePtr. Uses to which this name can be put are:
 - to restrict access over the SSL channel.
 - to change MCAUSER based on the name

Writing a security exit:

Figure 82 illustrates how to write a security exit.

```
void MQENTRY MQStart() {;}
void MQENTRY EntryPoint (PMQVOID pChannelExitParms,
                        PMQVOID pChannelDefinition,
                        PMQLONG pDataLength,
                        PMQLONG pAgentBufferLength,
                        PMQVOID pAgentBuffer,
                        PMQLONG pExitBufferLength,
                        PMQPTR pExitBufferAddr)
{
  PMQCXP pParms = (PMQCXP)pChannelExitParms;
  PMQCD pChDef = (PMQCD)pChannelDefinition;
  /* TODO: Add Security Exit Code Here */
}
```

Figure 82. Security exit skeleton code

The standard MQ Entry Point MQStart must exist, but is not required to perform any function. The name of the function (EntryPoint in this example) can be changed, but the function must be exported when the library is compiled and linked. The MQCXP and MQCD structures are passed to the exit as null pointers, so they must be cast before accessing fields. For general information on calling channel exits and the use of parameters, see “MQ_CHANNEL_EXIT – Channel exit” on page 406. These parameters are used in a security exit as follows:

PMQVOID pChannelExitParm

input/output

Pointer to MQCXP structure - cast to PMQCXP to access fields. This structure is used to communicate between the Exit and MCA. The following fields in the MQCXP are of particular interest for Security Exits:

ExitReason

Tells the Security Exit the current state in the security exchange and should be used when deciding what action to take.

ExitResponse

The response to the MCA which dictates the next stage in the security exchange.

ExitResponse2

Extra control flags to govern how the MCA interprets the Security Exit’s response.

ExitUserArea

16 bytes (maximum) of storage which can be used by the Security Exit to maintain state between calls.

ExitData

Contains the data specified in the SCYDATA field of the channel definition (32 bytes padded to the right with blanks).

PMQVOID pChannelDefinition

input/output

Pointer to MQCD structure - cast to PMQCD to access fields. This contains the definition of the channel. The following fields in the MQCD are of particular interest for Security Exits:

ChannelName

The channel name (20 bytes padded to the right with blanks).

ChannelType

A code defining the channel type.

MCA User Identifier

This group of 3 fields is initialized to the value of the MCAUSER field specified in the channel definition. Any user identifier specified by the Security Exit in these fields is used for authentication by the OAM (not applicable to CLNTCONN channels).

MCAUserIdentifier

First 12 bytes of identifier padded to the right with blanks.

LongMCAUserIdPtr

Pointer to a buffer containing the full length identifier (not guaranteed null terminated) takes priority over MCAUserIdentifier.

LongMCAUserIdLength

Length of string pointed to by LongMCAUserIdPtr - must be set if LongMCAUserIdPtr is set.

Remote User Identifier

Only applies to CLNTCONN/SVRCONN channel pairs. If no CLNTCONN Security Exit is defined then these 3 fields are initialized by the client MCA, so they may contain a user identifier from the environment of the client which can be used by a SVRCONN Security Exit for authentication and when specifying the MCA User Identifier. If a CLNTCONN Security Exit is defined then these fields are not initialized and can be set by the CLNTCONN Security Exit, or security messages can be used to pass a user identifier from Client to Server.

RemoteUserIdentifier

First 12 Bytes of identifier padded to the right with blanks.

LongRemoteUserIdPtr

Pointer to a buffer containing the full length identifier (not guaranteed null terminated) takes priority over RemoteUserIdentifier.

LongRemoteUserIdLength

Length of string pointed to by LongRemoteUserIdPtr - must be set if LongRemoteUserIdPtr is set.

PMQLONG pDataLength

input/output

Pointer to MQLONG. Contains the length of any Security Exit contained in the AgentBuffer upon invocation of the Security Exit. Must be set by a Security Exit to the length of any message being sent in the AgentBuffer or ExitBuffer.

PMQLONG pAgentBufferLength

input

Pointer to MQLONG. The length of the data contained in the AgentBuffer on invocation of the Security Exit.

PMQVOID pAgentBuffer

input/output

On invocation of the Security Exit this points to any message sent from the partner exit. If ExitReason2 in the MQCXP structure has the MQXR2_USE_AGENT_BUFFER flag set (default) then a Security Exit should set this to point to any message data being sent.

PMQLONG pExitBufferLength

input/output

Pointer to MQLONG. This is initialized to 0 on the first invocation of a Security Exit and the value returned is maintained between calls to the Security Exit during a security exchange.

PMQPTR pExitBufferAddr

input/output

This is initialized to a null pointer on the first invocation of a Security Exit and the value returned is maintained between calls to the Security Exit during a security exchange. If the MQXR2_USE_EXIT_BUFFER flag is set in the ExitReason2 in the MQCXP structure then a Security Exit should set this to point to any message data being sent.

Differences in behavior between security exits defined on CLNTCONN/SVRCONN channel pairs and other channel pairs:

Security exits can be defined on all types of channel. However, the behavior of security exits defined on CLNTCONN/SVRCONN channel pairs is slightly different to security exits defined on other channel pairs.

A Security Exit on a CLNTCONN channel can set the Remote User Identifier in the channel definition for processing by a partner CLNTCONN exit, or for OAM authorization if no SVRCONN Security Exit is defined and the MCAUSER field of the SVRCONN is not set. If no CLNTCONN Security Exit is defined then the Remote User Identifier in the channel definition is set to a user identifier from the client environment (which may be blank) by the client MCA.

A security exchange between Security Exits defined on a CLNTCONN and SVRCONN channel pair completes successfully when the SVRCONN Security Exit returns an ExitResponse of MQXCC_OK. A security exchange between other channel pairs completes successfully when the Security Exit which initiated the exchange returns an ExitResponse of MQXCC_OK.

However, the MQXCC_SEND_AND_REQUEST_SEC_MSG ExitResponse code can be used to force continuation of the security exchange: If an ExitResponse of MQXCC_SEND_AND_REQUEST_SEC_MSG is returned by a CLNTCONN or SVRCONN Security Exit then the partner exit must respond by sending a security message (not MQXCC_OK or a null response) or the channel will terminate. For Security Exits defined on other types of channel an ExitResponse of MQXCC_OK

returned in response to a MQXCC_SEND_AND_REQUEST_SEC_MSG from the partner Security Exit results in continuation of the security exchange as if a null response was returned and not in termination of the channel.

Channel send and receive exit programs

You can use the send and receive exits to perform tasks such as data compression and decompression. You can specify a list of send and receive exit programs to be run in succession.

Channel send and receive exit programs are called at the following places in an MCA's processing cycle:

- The send and receive exit programs are called for initialization at MCA initiation and for termination at MCA termination.
- The send exit program is invoked at either end of the channel, immediately before a transmission is sent over the link.
- The receive exit program is invoked at either end of the channel, immediately after a transmission has been taken from the link.

There may be many transmissions for one message transfer, and there could be many iterations of the send and receive exit programs before a message reaches the message exit at the receiving end.

The channel send and receive exit programs are passed an agent buffer containing the transmission data as sent or received from the communications link. For send exit programs, the first eight bytes of the buffer are reserved for use by the MCA, and must not be changed. If the program returns a different buffer, then these first eight bytes must exist in the new buffer. The format of data presented to the exit programs is not defined.

A good response code must be returned by send and receive exit programs. Any other response will cause an MCA abnormal end (abend).

Note: Do not issue an MQGET, MQPUT, or MQPUT1 call within syncpoint from a send or receive exit.

Note:

1. Send and receive exits usually work in pairs. For example a send exit may compress the data and a receive exit decompress it, or a send exit may encrypt the data and a receive exit decrypt it. When you define the appropriate channels, make sure that compatible exit programs are named for both ends of the channel.
2. If compression is turned on for the channel, the exits will be passed compressed data.
3. Channel send and receive exits may be called for message segments other than for application data, for example, status messages. They are not called during the startup dialog, nor the security check phase.
4. Although message channels send messages in one direction only, channel-control data flows in both directions, and these exits are available in both directions, also. However, some of the initial channel startup data flows are exempt from processing by any of the exits.
5. There are circumstances in which send and receive exits could be invoked out of sequence; for example, if you are running a series of exit programs or if you are also running security exits. Then, when the receive exit is first called upon

to process data, it may receive data that has not passed through the corresponding send exit. If the receive exit were just to perform the operation, for example decompression, without first checking that it was really required, the results would be unexpected.

You should code your send and receive exits in such a way that the receive exit can check that the data it is receiving has been processed by the corresponding send exit. The recommended way to do this is to code your exit programs so that:

- The send exit sets the value of the ninth byte of data to 0 and shifts all the data along one byte, before performing the operation. (The first eight bytes are reserved for use by the MCA.)
- If the receive exit receives data that has a 0 in byte 9, it knows that the data has come from the send exit. It removes the 0, performs the complementary operation, and shifts the resulting data back by one byte.
- If the receive exit receives data that has something other than 0 in byte 9, it assumes that the send exit has not run, and sends the data back to the caller unchanged.

When using security exits, if the channel is ended by the security exit it is possible that a send exit may be called without the corresponding receive exit. One way to prevent this from being a problem is to code the security exit to set a flag, in MQCD.SecurityUserData or MQCD.SendUserData, for example, when the exit decides to end the channel. Then the send exit should check this field, and process the data only if the flag is not set. This prevents the send exit from unnecessarily altering the data, and thus prevents any conversion errors that could occur if the security exit received altered data.

6. In the case of MQI channels for clients, byte 10 of the agent buffer identifies the API call in use when the send or receive exit is called. This is useful for identifying which channel flows include user data and may require processing such as encryption or digital signing.

Table 38 shows the data that appears in byte 10 of the channel flow when an API call is being processed.

Note: These are not the only values of this byte. There are other *reserved* values.

Table 38. Identifying API calls

API call	Value of byte 10 for request	Value of byte 10 for reply
MQCONN (1 on page 389, 2 on page 389)	X'81'	X'91'
MQDISC (1 on page 389)	X'82'	X'92'
MQOPEN (3 on page 389)	X'83'	X'93'
MQCLOSE	X'84'	X'94'
MQGET (4 on page 389)	X'85'	X'95'
MQPUT (4 on page 389)	X'86'	X'96'
MQPUT1 request (4 on page 389)	X'87'	X'97'
MQSET request	X'88'	X'98'
MQINQ request	X'89'	X'99'
MQCMIT request	X'8A'	X'9A'
MQBACK request	X'8B'	X'9B'
xa_start request	X'A1'	X'B1'

Table 38. Identifying API calls (continued)

API call	Value of byte 10 for request	Value of byte 10 for reply
xa_end request	X'A2'	X'B2'
xa_open request	X'A3'	X'B3'
xa_close request	X'A4'	X'B4'
xa_prepare request	X'A5'	X'B5'
xa_commit request	X'A6'	X'B6'
xa_rollback request	X'A7'	X'B7'
xa_forget request	X'A8'	X'B8'
xa_recover request	X'A9'	X'B9'
xa_complete request	X'AA'	X'BA'
Notes: <ol style="list-style-type: none"> 1. The connection between the client and server is initiated by the client application using MQCONN. Therefore, for this command in particular, there will be several other network flows. This also applies to MQDISC that terminates the network connection. 2. MQCONNX is treated in the same way as MQCONN for the purposes of the client-server connection. 3. If a large distribution list is opened, there may be more than one network flow per MQOPEN call in order to pass all of the required data to the SVRCONN MCA. 4. Large messages can exceed the transmission segment size. If this happens there can be a large number of network flows resulting from a single API call. 		

Channel send exit programs — reserving space

You can use send and receive exits to transform the data before transmission. Channel send exit programs can add their own data about the transformation by reserving space in the transmission buffer. This data is processed by the receive exit program and then removed from the buffer. For example, you might want to encrypt the data and add a security key for decryption.

How you reserve space and use it:

When the send exit program is called for initialization, set the *ExitSpace* field of MQXCP to the number of bytes to be reserved. See “MQXCP – Channel exit parameter” on page 458 for details. *ExitSpace* can be set only during initialization, that is when *ExitReason* has the value MQXR_INIT. When the send exit is invoked immediately before transmission, with *ExitReason* set to MQXR_XMIT, *ExitSpace* bytes are reserved in the transmission buffer. *ExitSpace* is not supported on z/OS.

The send exit need not use all of the reserved space. It can use less than *ExitSpace* bytes or, if the transmission buffer is not full, the exit can use more than the amount reserved. When setting the value of *ExitSpace*, you must leave at least 1 KB for message data in the transmission buffer. Note that channel performance can be affected if reserved space is used for large amounts of data.

What happens at the receiving end of the channel:

Channel receive exit programs must be set up to be compatible with the corresponding send exits. Receive exits must know the number of bytes in the reserved space and must remove the data in that space.

Multiple send exits:

You can specify a list of send and receive exit programs to be run in succession. WebSphere MQ maintains a total for the space reserved by all of the send exits. This total space must leave at least 1 KB for message data in the transmission buffer.

The following example shows how space is allocated for three send exits, called in succession:

1. When called for initialization:
 - Send exit A reserves 1 KB.
 - Send exit B reserves 2 KB.
 - Send exit C reserves 3 KB.
2. The maximum transmission size is 32 KB and the user data is 5 KB long.
3. Exit A is called with 5 KB of data; up to 27 KB are available, because 5KB is reserved for exits B and C. Exit A adds 1KB, the amount it reserved.
4. Exit B is called with 6 KB of data; up to 29 KB are available, because 3KB is reserved for exit C. Exit B adds 1KB, less than the 2KB it reserved.
5. Exit C is called with 7 KB of data; up to 32 KB are available. Exit C adds 10K, more than the 3KB it reserved. This is valid, because the total amount of data, 17 KB, is less than the 32KB maximum.

Channel message exit programs

You can use the channel message exit for the following:

- Encryption on the link
- Validation of incoming user IDs
- Substitution of user IDs according to local policy
- Message data conversion
- Journaling
- Reference message handling

On WebSphere MQ for i5/OS, UNIX systems, z/OS, and Windows systems, and with WebSphere MQ clients, you can specify a list of message exit programs to be run in succession.

Channel message exit programs are called at the following places in an MCA's processing cycle:

- At MCA initiation and termination
- Immediately after a sending MCA has issued an MQGET call
- Before a receiving MCA issues an MQPUT call

The message exit is passed an agent buffer containing the transmission queue header, MQXQH, and the application message text as retrieved from the queue. (The format of MQXQH is given in the WebSphere MQ Application Programming Reference book.) If you use reference messages, that is messages that contain only a header which points to some other object that is to be sent, the message exit recognizes the header, MQRMH. It identifies the object, retrieves it in whatever way is appropriate appends it to the header, and passes it to the MCA for transmission to the receiving MCA. At the receiving MCA, another message exit recognizes that this is a reference message, extracts the object, and passes the

header on to the destination queue. See the WebSphere MQ Application Programming Guide for more information about reference messages and some sample message exits that handle them.

Message exits can return the following responses:

- Send the message (GET exit). The message may have been changed by the exit. (This returns MQXCC_OK.)
- Put the message on the queue (PUT exit). The message may have been changed by the exit. (This returns MQXCC_OK.)
- Do not process the message. The message is placed on the dead-letter queue (undelivered message queue) by the MCA.
- Close the channel.
- Bad return code, which causes the MCA to abend.

Note:

1. Message exits are called just once for every complete message transferred, even when the message is split into parts.
2. In UNIX systems, if you provide a message exit for any reason the automatic conversion of user IDs to lowercase characters does not operate. See “User IDs on UNIX systems” on page 118.
3. An exit runs in the same thread as the MCA itself. It also runs inside the same unit of work (UOW) as the MCA because it uses the same connection handle. Therefore, any calls made under syncpoint are committed or backed out by the channel at the end of the batch. For example, one channel message exit program can send notification messages to another and these messages will only be committed to the queue when the batch containing the original message is committed.

Therefore, it is possible to issue syncpoint MQI calls from a channel message exit program.

Message conversion outside the message exit:

Before calling the message exit, the receiving MCA performs some conversions on the message. This section describes the algorithms used to perform the conversions.

Which headers are processed:

A conversion routine runs in the receiver’s MCA before the message exit is called. The conversion routine begins with the MQXQH header at the top of the message. The conversion routine then processes through the chained headers that follow the MQXQH, performing conversion where necessary. The chained headers can extend beyond the offset contained in the HeaderLength parameter of the MQCXP data that is passed to the receiver’s message exit. The following headers will be converted in-place:

- MQXQH (format name “MQXMIT ”)
- MQMD (this is part of the MQXQH and has no format name)
- MQMDE (format name “MQHMDE ”)
- MQDH (format name “MQHDIST ”)
- MQWIH (format name “MQHWIH ”)

The following headers will not be converted, but will be stepped over as the MCA continues to process the chained headers:

- MQDLH (format name "MQDEAD ")
- any headers with format names beginning with the three characters 'MQH' (eg. "MQHRF ") that are not otherwise mentioned above

How the headers are processed:

The Format parameter of each MQ header is read by the MCA. The Format parameter is 8 bytes within the header, which are 8 single-byte characters containing a name.

The MCA then interprets the data following each header as being of the named type. If the Format is the name of a header type eligible for MQ data conversion, it will be converted. If it is another name indicating non-MQ data (for example MQFMT_NONE or MQFMT_STRING) then the MCA stops processing the headers at this point.

What is the MQCXP HeaderLength?:

The HeaderLength parameter in the MQCXP data supplied to a message exit is the total length of the MQXQH (which includes the MQMD), MQMDE and MQDH headers at the start of the message. These headers are chained using the 'Format' names and lengths as per the general discussion above.

MQWIH:

As noted above, the chained headers can extend beyond the HeaderLength into the user data area. The MQWIH header, if it is present, is one of those that will appear beyond the HeaderLength.

If there is an MQWIH header in the chained headers, it will be converted in-place before the receiver's message exit is called.

Channel message retry exit program

The channel message-retry exit is called when an attempt to open the target queue is unsuccessful. You can use the exit to determine under which circumstances to retry, how many times to retry, and how frequently.

This exit is also called at the receiving end of the channel at MCA initiation and termination.

The channel message-retry exit is passed an agent buffer containing the transmission queue header, MQXQH, and the application message text as retrieved from the queue. The format of MQXQH is given in the WebSphere MQ Application Programming Reference book.

The exit is invoked for all reason codes; the exit determines for which reason codes it wants the MCA to retry, for how many times, and at what intervals. (The value of the message-retry count set when the channel was defined is passed to the exit in the MQCD, but the exit can ignore this.)

The MsgRetryCount field in MQCXP is incremented by the MCA each time the exit is invoked, and the exit returns either MQXCC_OK with the wait time contained in the MsgRetryInterval field of MQCXP, or MQXCC_SUPPRESS_FUNCTION. Retries continue indefinitely until the exit returns MQXCC_SUPPRESS_FUNCTION in the

ExitResponse field of MQCXP. See “MQCXP – Channel exit parameter” on page 458 for information about the action taken by the MCA for these completion codes.

If all the retries are unsuccessful, the message is written to the dead-letter queue. If there is no dead-letter queue available, the channel stops.

If you do not define a message-retry exit for a channel and a failure occurs that is likely to be temporary, for example MQRC_Q_FULL, the MCA uses the message-retry count and message-retry intervals set when the channel was defined. If the failure is of a more permanent nature and you have not defined an exit program to handle it, the message is written to the dead-letter queue.

Channel auto-definition exit program

The channel auto-definition exit can be used when a request is received to start a receiver or server-connection channel but no definition for that channel exists (not for WebSphere MQ for z/OS). It can also be called on all platforms for cluster-sender and cluster-receiver channels to allow definition modification for an instance of the channel.

The channel auto-definition exit can be called on all platforms except z/OS when a request is received to start a receiver or server-connection channel but no channel definition exists. You can use it to modify the supplied default definition for an automatically defined receiver or server-connection channel, SYSTEM.AUTO.RECEIVER or SYSTEM.AUTO.SVRCON. See “Auto-definition of receiver and server-connection channels” on page 54 for a description of how channel definitions can be created automatically.

The channel auto-definition exit can also be called when a request is received to start a cluster-sender channel. It can be called for cluster-sender and cluster-receiver channels to allow definition modification for this instance of the channel. In this case, the exit also applies to WebSphere MQ for z/OS. A common use of the channel auto-definition exit is to change the names of message exits (MSGEXIT, RCVEXIT, SCYEXIT, and SENDEXIT) because exit names have different formats on different platforms. If no channel auto-definition exit is specified, the default behavior on z/OS is to examine a distributed exit name of the form *[path]/libraryname(function)* and take up to 8 chars of function, if present, or libraryname. Note that on z/OS, a channel auto-definition exit program must alter the fields addressed by MsgExitPtr, MsgUserDataPtr, SendExitPtr, SendUserDataPtr, ReceiveExitPtr, and ReceiveUserDataPtr, rather than the MsgExit, MsgUserData, SendExit, SendUserData, ReceiveExit and ReceiveUserData fields themselves.

For more information, see *WebSphere MQ Queue Manager Clusters*.

As with other channel exits, the parameter list is:

MQ_CHANNEL_AUTO_DEF_EXIT (ChannelExitParms, ChannelDefinition)

ChannelExitParms are described in “MQCXP – Channel exit parameter” on page 458. ChannelDefinition is described in “MQCD – Channel definition” on page 413.

MQCD contains the values that are used in the default channel definition if they are not altered by the exit. The exit can modify only a subset of the fields; see “MQ_CHANNEL_AUTO_DEF_EXIT – Channel auto-definition exit” on page 410. However, attempting to change other fields does not cause an error.

The channel auto-definition exit returns a response of either MQXCC_OK or MQXCC_SUPPRESS_FUNCTION. If neither of these is returned, the MCA continues processing as though MQXCC_SUPPRESS_FUNCTION were returned. That is, the auto-definition is abandoned, no new channel definition is created and the channel cannot start.

Writing and compiling channel-exit programs

Channel exits must be named in the channel definition. You can do this when you first define the channels, or you can add the information later using, for example, the MQSC command ALTER CHANNEL. You can also give the channel exit names in the MQCD channel data structure. The format of the exit name depends on your WebSphere MQ platform; see “MQCD – Channel definition” on page 413 or the WebSphere MQ Script (MQSC) Command Reference book for information.

If the channel definition does not contain a user-exit program name, the user exit is not called.

The channel auto-definition exit is the property of the queue manager, not the individual channel. In order for this exit to be called, it must be named in the queue manager definition. To alter a queue manager definition, use the MQSC command ALTER QMGR.

User exits and channel-exit programs are able to make use of all MQI calls, except as noted in the sections that follow. To get the connection handle, an MQCONN must be issued, even though a warning, MQRC_ALREADY_CONNECTED, is returned because the channel itself is connected to the queue manager.

For exits on client-connection channels, the queue manager to which the exit tries to connect depends on how the exit was linked. If the exit was linked with MQM.LIB (or QMQM/LIBMQM on i5/OS) and you do not specify a queue manager name on the MQCONN call, the exit will try to connect to the default queue manager on your system. If the exit was linked with MQM.LIB (or QMQM/LIBMQM on i5/OS) and you specify the name of the queue manager that was passed to the exit through the QMgrName field of MQCD, the exit tries to connect to that queue manager. If the exit was linked with MQIC.LIB or any other library, the MQCONN call will fail whether you specify a queue manager name or not.

Note: You are recommended to avoid issuing the following MQI calls in channel-exit programs:

- MQCMIT
- MQBACK
- MQBEGIN
- MQDISC
- MQCONNX with MQCNO_HANDLE_SHARE_BLOCK or MQCNO_HANDLE_SHARE_NO_BLOCK options

MQCONNX with MQCNO_HANDLE_SHARE_BLOCK or MQCNO_HANDLE_SHARE_NO_BLOCK options returns a new shared handle on each call. If used inside an exit, the handle must be disconnected before returning from the exit. Otherwise, this can result in connection handles building up, and eventually agent threads will increase.

An exit runs in the same thread as the MCA itself and uses the same connection handle. So, it runs inside the same UOW as the MCA and any calls made under syncpoint are committed or backed out by the channel at the end of the batch.

Therefore, a channel message exit could send notification messages that will only be committed to that queue when the batch containing the original message is committed. So, it is possible to issue syncpoint MQI calls from a channel message exit.

A channel exit can change fields in the MQCD. However, these changes are not generally acted on, except in the circumstances listed. If a channel exit program changes a field in the MQCD data structure, the new value is generally ignored by the WebSphere MQ channel process. However, the new value remains in the MQCD and is passed to any remaining exits in an exit chain and to any conversation sharing the channel instance. For more information, see “Changing MQCD fields in a channel exit” on page 456

Also, for programs written in C, non-reentrant C library function should not be used in a channel-exit program.

If you use multiple channel exit libraries simultaneously, problems can arise on some UNIX platforms if the code for two different exits contains identically-named functions. When a channel exit is loaded, the dynamic loader resolves function names in the exit library to the addresses where the library is loaded. If two exit libraries define separate functions which happen to have identical names, this resolution process might incorrectly resolve the function names of one library to use the functions of another. If this problem occurs, specify to the linker that it must only export the required exit and MQStart functions, as these will be unaffected. Other functions should be given local visibility so that they will not be used by functions outside their own exit library. Consult your linker’s documentation for more information.

All exits are called with a channel exit parameter structure (MQCXP), a channel definition structure (MQCD), a prepared data buffer, data length parameter, and buffer length parameter. The buffer length must not be exceeded:

- For message exits, you should allow for the largest message required to be sent across the channel, plus the length of the MQXQH structure.
- For send and receive exits, the largest buffer you should allow for is as follows:

LU 6.2:

- 32 KB

TCP:

- i5/OS 16 KB
- Others 32 KB

Note: The maximum usable length may be 2 bytes less than this. Check the value returned in MaxSegmentLength for details. For more information on MaxSegmentLength, see MaxSegmentLength.

NetBIOS:

- 64 KB

SPX:

- 64 KB

Note: Receive exits on sender channels and sender exits on receiver channels use 2 KB buffers for TCP.

- For security exits, the distributed queuing facility allocates a buffer of 4000 bytes.

It is permissible for the exit to return an alternate buffer, together with the relevant parameters. See “Channel-exit programs” on page 375 for call details.

Note: Before using a channel-exit program for the first time on WebSphere MQ on UNIX systems, i5/OS, or Windows, you should relink it with threaded libraries to make it thread-safe.

WebSphere MQ for z/OS

The exits are invoked as if by a z/OS LINK, in:

- Non-authorized problem program state
- Primary address space control mode
- Non-cross-memory mode
- Non-access register mode
- 31-bit addressing mode

The link-edited modules must be placed in the data set specified by the CSQXLIB DD statement of the channel initiator address space procedure; the names of the load modules are specified as the exit names in the channel definition.

When writing channel exits for z/OS, the following rules apply:

- Exits must be written in assembler or C; if C is used, it must conform to the C systems programming environment for system exits, described in the *z/OS C/C++ Programming Guide*.
- Exits are loaded from the non-authorized libraries defined by a CSQXLIB DD statement. Providing CSQXLIB has DISP=SHR, exits can be updated while the channel initiator is running, with the new version used when the channel is restarted.
- Exits must be reentrant, and capable of running anywhere in virtual storage.
- Exits must reset the environment, on return, to that at entry.
- Exits must free any storage obtained, or ensure that it will be freed by a subsequent exit invocation.

For storage that is to persist between invocations, use the z/OS STORAGE service; there is no suitable service in C.

- All MQI calls except MQCMIT/CSQBCMT and MQBACK/CSQBBAK are allowed. They must be contained after MQCONN (with a blank queue manager name). If these calls are used, the exit must be link-edited with the stub CSQXSTUB.

The exception to this rule is that security channel exits may issue commit and backout MQI calls. To do this, code the verbs CSQXCMT and CSQXBK in place of MQCMIT/CSQBCMT and MQBACK/CSQBBAK.

- Exits should not use any system services that could cause a wait, because this would severely impact the handling of some or all of the other channels. Many channels are run under a single TCB typically, if you do something in an exit that causes a wait and you do not use MQXWAIT, it will cause *all* these

channels to wait. This will not give any functional problems, but might have an adverse effect on performance. Most SVCs involve waits, so you should avoid them, except for the following:

- GETMAIN/FREEMAIN/STORAGE
- LOAD/DELETE

In general, therefore, SVCs, PCs, and I/O should be avoided. Instead, the MQXWAIT call should be used.

- Exits should not issue ESTAEs or SPIEs, apart from in any subtasks they attach. This is because their error handling might interfere with the error handling performed by WebSphere MQ. This means that WebSphere MQ might not be able to recover from an error, or that your exit program might not receive all the error information.
- The MQXWAIT call (see “MQXWAIT – Wait in exit” on page 412) provides a wait service that allows waiting for I/O and other events; if this service is used, exits must not use the linkage stack.

For I/O and other facilities that do not provide non-blocking facilities or an ECB to wait on, a separate subtask should be ATTACHED, and its completion waited for by MQXWAIT; because of the overhead that this technique incurs, it is recommended that this be used only by the security exit.

- The MQDISC MQI call will not cause an implicit commit to occur within the exit program. A commit of the channel process is performed only when the channel protocol dictates.

The following exit samples are provided with WebSphere MQ for z/OS:

CSQ4BAX0

This sample is written in assembler, and illustrates the use of MQXWAIT.

CSQ4BCX1 and CSQ4BCX2

These samples are written in C and illustrate how to access the parameters.

WebSphere MQ for i5/OS

The exit is a program object written in the ILE C, ILE RPG, or ILE COBOL language. The exit program names and their libraries are named in the channel definition.

Observe the following conditions when creating and compiling an exit program:

- The program must be made thread safe and created with the ILE C, ILE RPG, or ILE COBOL compiler. For ILE RPG you must specify the THREAD(*SERIALIZE) control specification, and for ILE COBOL you must specify SERIALIZE for the THREAD option of the PROCESS statement. The programs must also be bound to the threaded WebSphere MQ libraries: QMQM/LIBMQM_R in the case of ILE C and ILE RPG, and AMQ0STUB_R in the case of ILE COBOL. For additional information about making RPG or COBOL applications thread safe, refer to the appropriate Programmer’s Guide for the language.
- WebSphere MQ for i5/OS requires that the exit programs are enabled for teraspace support. (Teraspace is a form of shared memory introduced in OS/400® V4R4.) In the case of the ILE RPG and COBOL compilers, any programs compiled on OS/400 V4R4 or later are so enabled. In the case of C, the programs must be compiled with the TERASPACE(*YES *TSIFC) options specified on CRTCMOD or CRTBNDC commands.
- An exit returning a pointer to its own buffer space must ensure that the object pointed to exists beyond the timespan of the channel-exit program. In other

words, the pointer cannot be the address of a variable on the program stack, nor of a variable in the program heap. Instead, the pointer must be obtained from the system. An example of this is a user space created in the user exit. To ensure that any data area allocated by the channel-exit program is still available for the MCA when the program ends, the channel exit must run in the caller's activation group or a named activation group. Do this by setting the ACTGRP parameter on CRTPGM to a user-defined value or *CALLER. If the program is created in this way, the channel-exit program can allocate dynamic memory and pass a pointer to this memory back to the MCA.

WebSphere MQ for Windows server, WebSphere MQ client for Windows

The exit is a DLL that must be written in C.

- On a WebSphere MQ for Windows server, specify the path name of the directory that holds the exit program on the Exits page of the queue manager properties (accessed from the WebSphere MQ Services snap-in).

If the exit is on a Windows client, specify the path name on the All Queue Managers page of the WebSphere MQ properties (accessed from the WebSphere MQ services snap-in).

The WebSphere MQ Services snap-in is described in the WebSphere MQ System Administration Guide manual.

The default exit path for 32-bit exits is <install location>\exits, and for 64-bit exits it is <install location>\exits64. You can alter the default location using WebSphere MQ Explorer.

Define a dummy MQStart() routine in the exit and specify MQStart as the entry point in the library. Figure 83 shows how to set up an entry to your program:

```
#include <cmqc.h>
#include <cmqxc.h>

void MQStart() {;} /* dummy entry point - for consistency only */
void MQENTRY ChannelExit ( PMQCXP  pChannelExitParms,
                           PMQCD   pChannelDefinition,
                           PMQLONG pDataLength,
                           PMQLONG pAgentBufferLength,
                           PMQVOID pAgentBuffer,
                           PMQLONG pExitBufferLength,
                           PMQPTR  pExitBufferAddr)

{
... Insert code here
}
```

Figure 83. Sample source code for a channel exit on Windows

In order to access the fields pointed to by pChannelExitParms and pChannelDefinition you need to insert the following lines in your exit program:

```
...
/* Variable definitions */
...
    PMQCXP      pParms;
    PMQCD       pChDef;
...
/* Code */
...
    pParms = (PMQCXP)pChannelExitParms;
    pChDef = (PMQCD)pChannelDefinition;
```

The pointers pParms and pChDef can then be dereferenced to access individual fields.

When writing channel exits for these products using Visual C++, you should do the following:

- Add MQMVX.LIB to project as a source file¹.
- Change the box labelled “Use Run-Time Library” from “Multithreaded” to “Multithreaded using DLL” in the project settings under C/C++ code generation.
- Do not change the box labelled “Entry-Point Symbol”. This box can be found in the project settings, under the Link tab, when you select Category and then Output.
- Write your own .DEF file; an example of this is shown in Figure 84.

```
LIBRARY exit
PROTMODE
DESCRIPTION 'Provides Retry and Channel exits'
CODE SHARED    LOADONCALL
DATA NONSHARED MULTIPLE
HEAPSIZE    4096
STACKSIZE   8192
EXPORTS    Retry
```

Figure 84. Sample DEF file for Windows

WebSphere MQ for AIX

Note: Before you use an existing user exit for the first time on WebSphere MQ for AIX, you must recompile it to enable it to take advantage of thread-safe system calls. If your user exits use thread-unsafe system calls, you will need to modify them before using them on this platform.

The exit is a dynamically loaded object that must be written in C. To ensure that it can be loaded when required, put 32-bit exits in /var/mqm/exits and 64-bit exits in /var/mqm/exits64.

These are the default paths for exits in the ExitPath stanza of the 'qm.ini' file or the ClientExitPath stanza of the WebSphere MQ client configuration file and can be changed if required. Exits on the server use the 'qm.ini' file, those on the client use the WebSphere MQ client configuration file.

If both mqs.ini and WebSphere MQ client configuration file values are encountered, the WebSphere MQ client configuration file value is used; if there is no WebSphere MQ client configuration file value, the mqs.ini value is used if it is set.

Alternatively you can specify the full path name in the MQCD if MQCONN is used or in the DEFINE CHANNEL command.

1. MQMVX.LIB is used for data conversion and is not available on client products.

Define a dummy MQStart() routine in the exit and specify MQStart as the entry point in the module. Figure 85 shows how to set up an entry to your program:

```
#include <cmqc.h>
#include <cmqxc.h>

void MQStart() {;} /* dummy entry point - for consistency only */
void MQENTRY ChannelExit ( PMQVOID pChannelExitParms,
                           PMQVOID pChannelDefinition,
                           PMQLONG pDataLength,
                           PMQLONG pAgentBufferLength,
                           PMQVOID pAgentBuffer,
                           PMQLONG pExitBufferLength,
                           PMQPTR pExitBufferAddr)
{
... Insert code here
}
```

Figure 85. Sample source code for a channel exit on AIX

Figure 86 shows the compiler and linker commands for channel-exit programs on AIX.

```
$ xlc_r -q64 -e MQStart -bE:exit.exp -bM:SRE -o /var/mqm/exits64/exit
exit.c -I/usr/mqm/inc -L/usr/mqm/lib64 -lmqm_r
```

Figure 86. Sample compiler and linker commands for channel exits on AIX

In this case **exit** is the library name and **ChannelExit** is the function name. The export file is called **exit.exp**. These names are used by the channel definition to reference the exit program using the format described in Exit name fields. See also the MSGEXIT parameter of the Define Channel command in WebSphere MQ Script (MQSC) Command Reference

Figure 87 shows a sample export file for this make file.

Note: All functions that will be called by WebSphere MQ must be exported.

```
#!
channelExit
MQStart
```

Figure 87. Sample export file for AIX

On the client, a 32-bit or 64-bit exit can be used. This must be linked to mqic_r.

WebSphere MQ for HP-UX

Note: Before you use an existing user exit for the first time on WebSphere MQ for HP-UX, you must recompile it to enable it to take advantage of thread-safe system calls. If your user exits use thread-unsafe system calls, you will need to modify them before using them on this platform.

The exit is a dynamically loaded object that must be written in C. To ensure that it can be loaded when required, put 32-bit exits in /var/mqm/exits and 64-bit exits in /var/mqm/exits64.

These are the default paths for exits in the ExitPath stanza of the 'qm.ini' file or the ClientExitPath stanza of the WebSphere MQ client configuration file and can be

changed if required. Exits on the server use the 'qm.ini' file, those on the client use the WebSphere MQ client configuration file.

If both mqs.ini and WebSphere MQ client configuration file values are encountered, the WebSphere MQ client configuration file value is used; if there is no WebSphere MQ client configuration file value, the mqs.ini value is used if it is set.

Alternatively you can specify the full path name in the MQCD if MQCONN is used or in the DEFINE CHANNEL command.

Define a dummy MQStart() routine in the exit and specify MQStart as the entry point in the module. Figure 88 shows how to set up an entry to your program:

```
#include <cmqc.h>
#include <cmqxc.h>

void MQStart() {;} /* dummy entry point - for consistency only */
void MQENTRY ChannelExit ( PMQVOID pChannelExitParms,
                           PMQVOID pChannelDefinition,
                           PMQLONG pDataLength,
                           PMQLONG pAgentBufferLength,
                           PMQVOID pAgentBuffer,
                           PMQLONG pExitBufferLength,
                           PMQPTR pExitBufferAddr)
{
  ... Insert code here
}
```

Figure 88. Sample source code for a channel exit on HP-UX

```
$ cc89 +DD64 +z -c -D_HPUX_SOURCE -o exit.o exit.c -I/opt/mqm/inc
$ ld -b +noenvvar exit.o +ee MQStart +ee ChannelExit -o
/var/mqm/exits64/exit -L/opt/mqm/lib64 -L/usr/lib/pa20_64 -lmqm_r -lpthread
$ rm exit.o
```

Figure 89. Sample compiler and linker commands for channel exits on HP-UX

On the client, a 32-bit or 64-bit exit can be used. This must be linked to mqic_r.

WebSphere MQ for Solaris

Note: Before you use an existing user exit for the first time on WebSphere MQ for Solaris, you must recompile it to enable it to take advantage of thread-safe system calls. If your user exits use thread-unsafe system calls, you will need to modify them before using them on this platform.

The exit is a dynamically loaded object that must be written in C. To ensure that it can be loaded when required, put 32-bit exits in /var/mqm/exits and 64-bit exits in /var/mqm/exits64.

These are the default paths for exits in the ExitPath stanza of the 'qm.ini' file or the ClientExitPath stanza of the WebSphere MQ client configuration file and can be changed if required. Exits on the server use the 'qm.ini' file, those on the client use the WebSphere MQ client configuration file.

If both mqs.ini and WebSphere MQ client configuration file values are encountered, the WebSphere MQ client configuration file value is used; if there is no WebSphere MQ client configuration file value, the mqs.ini value is used if it is set.

Alternatively you can specify the full path name in the MQCD if MQCONN is used or in the DEFINE CHANNEL command.

Define a dummy MQStart() routine in the exit and specify MQStart as the entry point in the module. Figure 90 shows how to set up an entry to your program:

```
#include <cmqc.h>
#include <cmqxc.h>

void MQStart() {;} /* dummy entry point */
void MQENTRY ChannelExit ( PMQVOID pChannelExitParms,
                          PMQVOID pChannelDefinition,
                          PMQLONG pDataLength,
                          PMQLONG pAgentBufferLength,
                          PMQVOID pAgentBuffer,
                          PMQLONG pExitBufferLength,
                          PMQPTR pExitBufferAddr)
{
... Insert code here
}
```

Figure 90. Sample source code for a channel exit on Solaris

Figure 91 shows the compiler and linker commands for channel-exit programs on Solaris.

```
$ cc -xarch=v9 -mt -G -o /var/mqm/exits64/exit exit.c -I/opt/mqm/inc
-L/opt/mqm/lib64 -R/opt/mqm/lib64 -R/usr/lib/64 -lmqm -lmqmcs -lmqmzse -lsocket
-lnsl -ldl
```

Figure 91. Sample compiler and linker commands for channel exits on Solaris

In this case **exit** is the library name and **ChannelExit** is the function name. These names are used by the channel definition to reference the exit program using the format described in Exit name fields. See also the MSGEXIT parameter of the Define Channel command in WebSphere MQ Script (MQSC) Command Reference

On the client, a 32-bit or 64-bit exit can be used. This must be linked to mqic_r.

WebSphere MQ for Linux

Note: Before you use an existing user exit for the first time on WebSphere MQ for Linux, you must recompile it to enable it to take advantage of thread-safe system calls. If your user exits use thread-unsafe system calls, you will need to modify them before using them on this platform.

The exit is a dynamically loaded object that must be written in C. To ensure that it can be loaded when required, put 32-bit exits in /var/mqm/exits and 64-bit exits in /var/mqm/exits64.

These are the default paths for exits in the ExitPath stanza of the 'qm.ini' file or the ClientExitPath stanza of the WebSphere MQ client configuration file and can be changed if required. Exits on the server use the 'qm.ini' file, those on the client use the WebSphere MQ client configuration file.

If both mqs.ini and WebSphere MQ client configuration file values are encountered, the WebSphere MQ client configuration file value is used; if there is no WebSphere MQ client configuration file value, the mqs.ini value is used if it is set.

Alternatively you can specify the full path name in the MQCD if MQCONN is used or in the DEFINE CHANNEL command.

Define a dummy MQStart() routine in the exit and specify MQStart as the entry point in the module. Figure 92 shows how to set up an entry to your program:

```
#include <cmqc.h>
#include <cmqxc.h>
void MQStart() {;} /* dummy entry point */
void MQENTRY ChannelExit ( PMQVOID pChannelExitParms,
PMQVOID pChannelDefinition,
PMQLONG pDataLength,
PMQLONG pAgentBufferLength,
PMQVOID pAgentBuffer,
PMQLONG pExitBufferLength,
PMQPTR pExitBufferAddr)
{
... Insert code here
}
```

Figure 92. Sample source code for a channel exit on Linux

Figure 93 shows the compiler and linker commands for channel-exit programs on Linux:

```
$ gcc -m64 -shared -fPIC -o /var/mqm/exits64/exit exit.c -I/opt/mqm/inc
-L/opt/mqm/lib64 -Wl,-rpath=/opt/mqm/lib64 -Wl,-rpath=/usr/lib64 -lmqm_r
```

Figure 93. Sample compiler and linker commands for channel-exits on Linux platforms where the queue manager is 64-bit

In this case **exit** is the library name and **ChannelExit** is the function name. These names are used by the channel definition to reference the exit program using the format described in Exit name fields. See also the MSGEXIT parameter of the Define Channel command in WebSphere MQ Script (MQSC) Command Reference

On the client, a 32-bit or 64-bit exit can be used. This must be linked to mqic_r. For WebSphere MQ for Linux (x86 platform), if you use a 32-bit server platform, then use the compiler and linker commands shown in Figure 94.

```
$ gcc -shared -fPIC -o /var/mqm/exits/exit exit.c -I/opt/mqm/inc
-L/opt/mqm/lib -Wl,-rpath=/opt/mqm/lib -Wl,-rpath=/usr/lib -lmqm_r
```

Figure 94. Sample compiler and linker commands for channel-exits on Linux platforms where the queue manager is 32-bit

SSPI security exit

WebSphere MQ for Windows supplies a security exit for both the WebSphere MQ client and the WebSphere MQ server. This is a channel-exit program that provides authentication for WebSphere MQ channels by using the Security Services Programming Interface (SSPI). The SSPI provides the integrated security facilities of Windows.

The security packages are loaded from either security.dll or secur32.dll. These DLLs are supplied with your operating system.

One-way authentication is provided on Windows, using NTLM authentication services. Two way authentication is provided on Windows 2000, using Kerberos authentication services.

The security exit program is supplied in source and object format. You can use the object code as it is, or you can use the source code as a starting point to create your own user-exit programs. For more information on using the object or source code of the SSPI security exit, see WebSphere MQ Application Programming Guide

Implications of sharing conversations

Channel instances can be shared in some circumstances. This topic discusses the interaction of the SharingConversations fields in the MQCD and MQCXP structures and the implications for channel exits.

In an environment where sharing conversations is not permitted, conversations cannot share a channel instance.

In an environment where sharing conversations is permitted, conversations can share a channel instance. Sharing conversations is controlled at a channel instance level by two fields, both called SharingConversations. One of these is part of the channel definition (MQCD) structure and the other is part of the channel exit parameter (MQCXP) structure. The SharingConversations field in the MQCD is an integer value, determining the maximum number of conversations that can share a channel instance associated with the channel. The SharingConversations fields in the MQCXP is a boolean value, indicating whether the channel instance can be shared.

If the first channel exit program to run on a channel instance sets the MQCD SharingConversations field to zero, the exit program is the only one to run on this channel instance; the channel instance is never shared. Further exit programs that set this field to zero similarly run on their own channel instance.

If the first channel exit program to run on a channel instance sets the MQCD SharingConversations field to a number greater than zero then MQCXP SharingConversations is set to TRUE, indicating that further conversations can be started on the same channel instance. If a new user-supplied channel definition matches the user-supplied definition of this already existing channel (that is, its definition before any alteration by a channel exit program) then the existing channel instance is used. The channel exit program cannot choose to use a separate channel instance. If there are no existing channels matching the new channel definition, a new channel instance is created and MQCXP SharingConversations is set to FALSE.

When you write a channel exit program, consider whether it will run on a channel instance that might be shared. If the channel instance might be shared, consider the impact on other instances of the channel exit of changing MQCD fields, which have common values across all the sharing conversations.

Example

Sharing conversations is enabled.

You are using a client-connection channel definition which specifies an exit program.

The first time that this channel starts, the exit program alters some of the MQCD parameters when it is initialized. These are acted on by the channel, so the definition that the channel is running with is now different from the one that was originally supplied. The MQCXP SharingConversations parameter is set to TRUE.

The next time that the application connects using this channel, the conversation runs on the channel instance which was started previously, because it has the same original channel definition. The conversation is now running on a channel instance which was set up using the definition altered by the exit on the first conversation. When the exit program is initialized for the second conversation, although it can alter MQCD fields, they are *not* acted on by the channel. These same characteristics apply to any subsequent conversations which share the channel instance.

Each exit instance within a channel instance shares the same MQCD. Once the channel instance is established, if exit programs try to alter MQCD fields they might encounter problems because other instances of exit programs running on the channel instance could be attempting to alter the same fields at the same time. If this situation could arise with your exit programs, you must serialize access to the MQCD in your exit code.

If you are working with a channel which is defined to share conversations, but you do not want sharing to occur on a particular channel instance, set the MQCD value of SharingConversations to 1 or 0 when you initialize a channel exit on the first conversation on the channel instance. See “SharingConversations (MQLONG)” on page 442 for an explanation of the values of SharingConversations.

Channel-exit calls and data structures

This topic provides reference information about the special WebSphere MQ calls and data structures that you can use when you write channel exit programs.

This is product-sensitive programming interface information. You can write WebSphere MQ user exits in the following programming languages:

Platform	Programming languages
WebSphere MQ for z/OS	Assembler and C (which must conform to the C system programming environment for system exits, described in the <i>z/OS C/C++ Programming Guide</i> .)
WebSphere MQ for i5/OS	C, COBOL, and RPG II
All other WebSphere MQ platforms	C

You can also write user exits in Java™ for use only with Java and JMS applications. For more information about using channel exits with the WebSphere MQ classes for Java and WebSphere MQ classes for JMS, see Using channel exits.

You cannot write WebSphere MQ user exits in TAL or Visual Basic. However, a declaration for the MQCD structure is provided in Visual Basic for use on the MQCONN call from an MQ client program.

In a number of cases in the descriptions that follow, parameters are arrays or character strings whose size is not fixed. For these, a lowercase “n” is used to represent a numeric constant. When the declaration for that parameter is coded, the “n” must be replaced by the numeric value required. For further information

about the conventions used in these descriptions, see the WebSphere MQ Application Programming Reference book.

The calls are:

- “MQ_CHANNEL_EXIT – Channel exit”
- “MQ_CHANNEL_AUTO_DEF_EXIT – Channel auto-definition exit” on page 410
- “MQXWAIT – Wait in exit” on page 412

The data structures are:

- “MQCD – Channel definition” on page 413
- “MQCXP – Channel exit parameter” on page 458
- “MQXWD – Exit wait descriptor” on page 475

Data definition files

Data definition files are supplied with WebSphere MQ for each of the supported programming languages. For details of these, see the chapter “Header files” in WebSphere MQ Constants.

MQ_CHANNEL_EXIT – Channel exit

This call definition is provided solely to describe the parameters that are passed to each of the channel exits called by the Message Channel Agent. No entry point called MQ_CHANNEL_EXIT is actually provided by the queue manager; the name MQ_CHANNEL_EXIT is of no special significance since the names of the channel exits are provided in the channel definition MQCD.

There are five types of channel exit:

- Channel security exit
- Channel message exit
- Channel send exit
- Channel receive exit
- Channel message-retry exit

The parameters are similar for each type of exit, and the description given here applies to all of them, except where specifically noted.

Syntax

MQ_CHANNEL_EXIT (*ChannelExitParms, ChannelDefinition, DataLength, AgentBufferLength, AgentBuffer, ExitBufferLength, ExitBufferAddr*)

Parameters

The MQ_CHANNEL_EXIT call has the following parameters.

ChannelExitParms (MQCXP) – input/output:

Channel exit parameter block.

This structure contains additional information relating to the invocation of the exit. The exit sets information in this structure to indicate how the MCA should proceed.

ChannelDefinition (MQCD) – input/output:

Channel definition.

This structure contains parameters set by the administrator to control the behavior of the channel.

DataLength (MQLONG) – input/output:

Length of data.

When the exit is invoked, this contains the length of data in the *AgentBuffer* parameter. The exit must set this to the length of the data in either the *AgentBuffer* or the *ExitBufferAddr* (as determined by the *ExitResponse2* field in the *ChannelExitParms* parameter) that is to proceed.

The data depends on the type of exit:

- For a channel security exit, when the exit is invoked this contains the length of any security message in the *AgentBuffer* field, if *ExitReason* is MQXR_SEC_MSG. It is zero if there is no message. The exit must set this field to the length of any security message to be sent to its partner if it sets *ExitResponse* to MQXCC_SEND_SEC_MSG or MQXCC_SEND_AND_REQUEST_SEC_MSG. The message data is in either *AgentBuffer* or *ExitBufferAddr*.
The content of security messages is the sole responsibility of the security exits.
- For a channel message exit, when the exit is invoked this contains the length of the message (including the transmission queue header). The exit must set this field to the length of the message in either *AgentBuffer* or *ExitBufferAddr* that is to proceed.
- For a channel send or channel receive exit, when the exit is invoked this contains the length of the transmission. The exit must set this field to the length of the transmission in either *AgentBuffer* or *ExitBufferAddr* that is to proceed.

If a security exit sends a message, and there is no security exit at the other end of the channel, or the other end sets an *ExitResponse* of MQXCC_OK, the initiating exit is re-invoked with MQXR_SEC_MSG and a null response (*DataLength*=0).

AgentBufferLength (MQLONG) – input:

Length of agent buffer.

This can be greater than *DataLength* on invocation.

For channel message, send, and receive exits, any unused space on invocation can be used by the exit to expand the data in place. If this is done, the *DataLength* parameter must be set appropriately by the exit.

In the C programming language, this parameter is passed by address.

AgentBuffer (MQBYTE×AgentBufferLength) – input/output:

Agent buffer.

The contents of this depend upon the exit type:

- For a channel security exit, on invocation of the exit it contains a security message if *ExitReason* is MQXR_SEC_MSG. If the exit wishes to send a security message back, it can either use this buffer or its own buffer (*ExitBufferAddr*).
- For a channel message exit, on invocation of the exit this contains:
 - The transmission queue header (MQXQH), which includes the message descriptor (which itself contains the context information for the message), immediately followed by
 - The message data

If the message is to proceed, the exit can do one of the following:

- Leave the contents of the buffer untouched
 - Modify the contents in place (returning the new length of the data in *DataLength*; this must not be greater than *AgentBufferLength*)
 - Copy the contents to the *ExitBufferAddr*, making any required changes
- Any changes that the exit makes to the transmission queue header are not checked; however, erroneous modifications may mean that the message cannot be put at the destination.

- For a channel send or receive exit, on invocation of the exit this contains the transmission data. The exit can do one of the following:
 - Leave the contents of the buffer untouched
 - Modify the contents in place (returning the new length of the data in *DataLength*; this must not be greater than *AgentBufferLength*)
 - Copy the contents to the *ExitBufferAddr*, making any required changesNote that the first 8 bytes of the data must not be changed by the exit.

ExitBufferLength (MQLONG) – input/output:

Length of exit buffer.

On the first invocation of the exit, this is set to zero. Thereafter whatever value is passed back by the exit, on each invocation, is presented to the exit next time it is invoked. The value is not used by the MCA.

Note: This parameter should not be used by exits written in programming languages which do not support the pointer data type.

ExitBufferAddr (MQPTR) – input/output:

Address of exit buffer.

This is a pointer to the address of a buffer of storage managed by the exit, where it can choose to return message or transmission data (depending upon the type of exit) to the agent if the agent's buffer is or may not be large enough, or if it is more convenient for the exit to do so.

On the first invocation of the exit, the address passed to the exit is null. Thereafter whatever address is passed back by the exit, on each invocation, is presented to the exit the next time it is invoked.

Note: This parameter should not be used by exits written in programming languages that do not support the pointer data type.

Usage notes

1. The function performed by the channel exit is defined by the provider of the exit. The exit, however, must conform to the rules defined here and in the associated control block, the MQCXP.
2. The *ChannelDefinition* parameter passed to the channel exit may be one of several versions. See the *Version* field in the MQCD structure for more information.
3. If the channel exit receives an MQCD structure with the *Version* field set to a value greater than MQCD_VERSION_1, the exit should use the *ConnectionName* field in MQCD, in preference to the *ShortConnectionName* field.
4. In general, channel exits are allowed to change the length of message data. This may arise as a result of the exit adding data to the message, or removing data from the message, or compressing or encrypting the message. However, special restrictions apply if the message is a segment that contains only part of a logical message. In particular, there must be no net change in the length of the message as a result of the actions of complementary sending and receiving exits.

For example, it is permissible for a sending exit to shorten the message by compressing it, but the complementary receiving exit must restore the original length of the message by decompressing it, so that there is no net change in the length of the message.

This restriction arises because changing the length of a segment would cause the offsets of later segments in the message to be incorrect, and this would inhibit the queue manager's ability to recognize that the segments formed a complete logical message.

C invocation

```
exitname (&ChannelExitParms, &ChannelDefinition,  
         &DataLength, &AgentBufferLength, AgentBuffer,  
         &ExitBufferLength, &ExitBufferAddr);
```

The parameters passed to the exit are declared as follows:

```
MQCXP  ChannelExitParms; /* Channel exit parameter block */  
MQCD   ChannelDefinition; /* Channel definition */  
MQLONG DataLength; /* Length of data */  
MQLONG AgentBufferLength; /* Length of agent buffer */  
MQBYTE AgentBuffer[n]; /* Agent buffer */  
MQLONG ExitBufferLength; /* Length of exit buffer */  
MQPTR  ExitBufferAddr; /* Address of exit buffer */
```

COBOL invocation

```
CALL 'exitname' USING CHANNELEXITPARMS, CHANNELDEFINITION,  
                     DATALENGTH, AGENTBUFFERLENGTH, AGENTBUFFER,  
                     EXITBUFFERLENGTH, EXITBUFFERADDR.
```

The parameters passed to the exit are declared as follows:

```
** Channel exit parameter block  
01 CHANNELEXITPARMS.  
   COPY CMQCXPV.  
** Channel definition  
01 CHANNELDEFINITION.  
   COPY CMQCDV.  
** Length of data  
01 DATALENGTH          PIC S9(9) BINARY.  
** Length of agent buffer  
01 AGENTBUFFERLENGTH   PIC S9(9) BINARY.  
** Agent buffer  
01 AGENTBUFFER          PIC X(n).
```

```

** Length of exit buffer
01 EXITBUFFERLENGTH PIC S9(9) BINARY.
** Address of exit buffer
01 EXITBUFFERADDR POINTER.

```

RPG invocation (ILE)

```

C*..1.....2.....3.....4.....5.....6.....7..
C          CALLP      exitname(MQCXP : MQCD : DATLEN :
C                               ABUFL : ABUF : EBUFL :
C                               EBUF)

```

The prototype definition for the call is:

```

D*..1.....2.....3.....4.....5.....6.....7..
Dexitname      PR          EXTPROC('exitname')
D* Channel exit parameter block
D MQCXP                160A
D* Channel definition
D MQCD                 1328A
D* Length of data
D DATLEN                10I 0
D* Length of agent buffer
D ABUFL                10I 0
D* Agent buffer
D ABUF                  *   VALUE
D* Length of exit buffer
D EBUFL                10I 0
D* Address of exit buffer
D EBUF                  *

```

System/390 assembler invocation

```

CALL EXITNAME, (CHANNELEXITPARMS, CHANNELDEFINITION, DATALENGTH, X
               AGENTBUFFERLENGTH, AGENTBUFFER, EXITBUFFERLENGTH, X
               EXITBUFFERADDR)

```

The parameters passed to the exit are declared as follows:

CHANNELEXITPARMS	CMQCXPA	,	Channel exit parameter block
CHANNELDEFINITION	CMQCDA	,	Channel definition
DATALENGTH	DS	F	Length of data
AGENTBUFFERLENGTH	DS	F	Length of agent buffer
AGENTBUFFER	DS	CL(n)	Agent buffer
EXITBUFFERLENGTH	DS	F	Length of exit buffer
EXITBUFFERADDR	DS	F	Address of exit buffer

MQ_CHANNEL_AUTO_DEF_EXIT – Channel auto-definition exit

This call definition is provided solely to describe the parameters that are passed to the channel auto-definition exit called by the Message Channel Agent. No entry point called MQ_CHANNEL_AUTO_DEF_EXIT is actually provided by the queue manager; the name MQ_CHANNEL_AUTO_DEF_EXIT is of no special significance because the names of the auto-definition exits are provided in the queue manager.

Syntax

MQ_CHANNEL_AUTO_DEF_EXIT (*ChannelExitParms*, *ChannelDefinition*)

Parameters

The MQ_CHANNEL_AUTO_DEF_EXIT call has the following parameters.

ChannelExitParms (MQCXP) – input/output:

Channel exit parameter block.

This structure contains additional information relating to the invocation of the exit. The exit sets information in this structure to indicate how the MCA should proceed.

ChannelDefinition (MQCD) – input/output:

Channel definition.

This structure contains parameters set by the administrator to control the behavior of channels which are created automatically. The exit sets information in this structure to modify the default behavior set by the administrator.

The MQCD fields listed below must not be altered by the exit:

- *ChannelName*
- *ChannelType*
- *StrucLength*
- *Version*

If other fields are changed, the value set by the exit must be valid. If the value is not valid, an error message is written to the error log file or displayed on the console (as appropriate to the environment).

Usage notes

1. The function performed by the channel exit is defined by the provider of the exit. The exit, however, must conform to the rules defined here and in the associated control block, the MQCXP.
2. The *ChannelExitParms* parameter passed to the channel auto-definition exit is an MQCXP structure. The version of MQCXP passed depends on the environment in which the exit is running; see the description of the *Version* field in “MQCXP – Channel exit parameter” on page 458 for details.
3. The *ChannelDefinition* parameter passed to the channel auto-definition exit is an MQCD structure. The version of MQCD passed depends on the environment in which the exit is running; see the description of the *Version* field in “MQCD – Channel definition” on page 413 for details.

C invocation

```
exitname (&ChannelExitParms, &ChannelDefinition);
```

The parameters passed to the exit are declared as follows:

```
MQCXP ChannelExitParms; /* Channel exit parameter block */
MQCD ChannelDefinition; /* Channel definition */
```

COBOL invocation

```
CALL 'exitname' USING CHANNELEXITPARMS, CHANNELDEFINITION.
```

The parameters passed to the exit are declared as follows:

```

** Channel exit parameter block
01 CHANNELEXITPARMS.
   COPY CMQXPV.
** Channel definition
01 CHANNELDEFINITION.
   COPY CMQCDV.

```

RPG invocation (ILE)

```

C*..1.....2.....3.....4.....5.....6.....7..
C          CALLP          exitname(MQCXP : MQCD)

```

The prototype definition for the call is:

```

D*..1.....2.....3.....4.....5.....6.....7..
Dexitname      PR          EXTPROC('exitname')
D* Channel exit parameter block
D MQCXP                160A
D* Channel definition
D MQCD                 1328A

```

System/390 assembler invocation

```
CALL EXITNAME,(CHANNELEXITPARMS,CHANNELDEFINITION)
```

The parameters passed to the exit are declared as follows:

```

CHANNELEXITPARMS  CMQCXPA  , Channel exit parameter block
CHANNELDEFINITION CMQCDA   , Channel definition

```

MQXWAIT – Wait in exit

The MQXWAIT call waits for an event to occur. It can be used only from a channel exit on z/OS.

The use of MQXWAIT helps to avoid performance problems that might otherwise occur if a channel exit does something that causes a wait. The event MQXWAIT is waiting on is signalled by an MVS ECB (event control block). The ECB is described in the MQXWD control block description .

For more information on the use of MQXWAIT and writing channel-exit programs, see “WebSphere MQ for z/OS” on page 396

Syntax

```
MQXWAIT (Hconn, WaitDesc, CompCode, Reason)
```

Parameters

The MQXWAIT call has the following parameters.

Hconn (MQHCONN) – input

Connection handle.

This handle represents the connection to the queue manager. The value of *Hconn* was returned by a previous MQCONN call issued in the same or earlier invocation of the exit.

WaitDesc (MQXWD) – input/output

Wait descriptor.

This describes the event to wait for. See “MQXWD – Exit wait descriptor” on page 475 for details of the fields in this structure.

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.

MQRC_ADAPTER_NOT_AVAILABLE

(2204, X'89C') Adapter not available.

MQRC_OPTIONS_ERROR

(2046, X'7FE') Options not valid or not consistent.

MQRC_XWAIT_CANCELED

(2107, X'83B') MQXWAIT call canceled.

MQRC_XWAIT_ERROR

(2108, X'83C') Invocation of MQXWAIT call not valid.

For more information on these reason codes, see the WebSphere MQ Application Programming Reference.

C invocation

```
MQXWAIT (Hconn, &WaitDesc, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHCONN  Hconn;    /* Connection handle */
MQXWD    WaitDesc; /* Wait descriptor */
MQLONG   CompCode; /* Completion code */
MQLONG   Reason;   /* Reason code qualifying CompCode */
```

System/390[®] assembler invocation

```
CALL MQXWAIT,(HCONN,WAITDESC,COMP CODE,REASON)
```

Declare the parameters as follows:

```
HCONN      DS      F  Connection handle
WAITDESC   CMQXWDA ,  Wait descriptor
COMP CODE  DS      F  Completion code
REASON     DS      F  Reason code qualifying COMP CODE
```

MQCD – Channel definition

The MQCD structure contains the parameters which control execution of a channel. It is passed to each channel exit that is called from a Message Channel Agent (MCA). See MQ_CHANNEL_EXIT. The description in section relates both to message channels and to MQI channels.

Exit name fields

When an exit is called, the relevant field from *SecurityExit*, *MsgExit*, *SendExit*, *ReceiveExit*, and *MsgRetryExit* contains the name of the exit currently being invoked. The meaning of the name in these fields depends on the environment in which the MCA is running. Except where noted below, the name is left-justified within the field, with no embedded blanks; the name is padded with blanks to the length of the field. In the descriptions that follow, square brackets ([]) denote optional information:

UNIX systems

The exit name is the name of a dynamically-loadable module or library, suffixed with the name of a function residing in that library. The function name must be enclosed in parentheses. The library name can optionally be prefixed with a directory path:

`[path]library(function)`

The name is limited to a maximum of 128 characters.

z/OS The exit name is the name of a load module that is valid for specification on the EP parameter of the LINK or LOAD macro. The name is limited to a maximum of 8 characters.

Windows

The exit name is the name of a dynamic-link library, suffixed with the name of a function residing in that library. The function name must be enclosed in parentheses. The library name can optionally be prefixed with a directory path and drive:

`[d:][path]library(function)`

The name is limited to a maximum of 128 characters.

i5/OS The exit name is a 10-byte program name followed by a 10-byte library name. If the names are less than 10 bytes long, each name is padded with blanks to make it 10 bytes. The library name can be *LIBL except when calling a channel auto-definition exit, in which case a fully qualified name is required.

Fields

This topic lists all the fields in the MQCD structure and describes each field.

ChannelName (MQCHAR20):

Channel definition name.

There must be a channel definition of the same name at the remote machine to be able to communicate.

The name must use only the characters:

- Uppercase A–Z
- Lowercase a–z
- Numerics 0–9
- Period (.)
- Forward slash (/)
- Underscore (_)

- Percent sign (%)

and be padded to the right with blanks. Leading or embedded blanks are not allowed.

The length of this field is given by MQ_CHANNEL_NAME_LENGTH.

Version (MQLONG):

Structure version number.

The value depends on the environment:

MQCD_VERSION_1

Version-1 channel definition structure.

MQCD_VERSION_2

Version-2 channel definition structure.

This value is not used by any current WebSphere MQ product.

MQCD_VERSION_3

Version-3 channel definition structure.

The field has this value on MQSeries Version 2 in the following environments: HP OpenVMS, Compaq NonStop Kernel, and UNIX systems not listed elsewhere.

MQCD_VERSION_4

Version-4 channel definition structure.

This value is not used by any current WebSphere MQ product.

MQCD_VERSION_5

Version-5 channel definition structure.

The field has this value on MQSeries for OS/390 Version 5 Release 2.

MQCD_VERSION_6

Version-6 channel definition structure.

This is not the current MQCD structure version of any existing WebSphere MQ product. However, a Version-6 MQCD structure can be passed to MQCONN using the ClientConnOffset or ClientConnPtr fields of the MQCNO structure.

On the distributed platforms MQCD_VERSION_6 is the default Version in the MQCD_DEFAULT and MQCD_CLIENT_CONN_DEFAULT initializers. If you want to reference the MQCD_VERSION_7, MQCD_VERSION_8, or MQCD_VERSION_9 fields of the MQCD, explicitly initialize the MQCD Version field to MQCD_VERSION_7, MQCD_VERSION_8, or MQCD_VERSION_9 as appropriate.

MQCD_VERSION_7

Version-7 channel definition structure.

The field has this value on WebSphere MQ Version 5 Release 3 in the following environments: AIX, HP-UX, Solaris, Windows, and on WebSphere MQ for z/OS Version 5 Release 3 and Version 5 Release 3.1.

MQCD_VERSION_8

Version-8 channel definition structure.

The field has this value on WebSphere MQ Version 6.0 on all platforms.

MQCD_VERSION_9

Version-9 channel definition structure.

The field has this value on WebSphere MQ Version 7.0 on all platforms.

Fields that exist only in the more-recent versions of the structure are identified as such in the descriptions of the fields. The following constant specifies the version number of the current version:

MQCD_CURRENT_VERSION

Current version of channel definition structure.

The value of this constant depends on the environment (see above). For WebSphere MQ Version 7, the declarations of MQCD provided in the header, COPY, and INCLUDE files for the supported programming languages contain the additional fields, but the initial value provided for the Version field is MQCD_VERSION_6. To use the additional fields, the application must set the version number to MQCD_CURRENT_VERSION. Applications which are intended to be portable between several environments should use a more-recent version MQCD only if all of those environments support that version.

Note: When a new version of the MQCD structure is introduced, the layout of the existing part is not changed. The exit should therefore check that the version number is equal to or greater than the lowest version which contains the fields that the exit needs to use.

ChannelType (MQLONG):

Channel type.

It is one of the following:

MQCHT_SENDER

Sender.

MQCHT_SERVER

Server.

MQCHT_RECEIVER

Receiver.

MQCHT_REQUESTER

Requester.

MQCHT_CLNTCONN

Client connection.

MQCHT_SVRCONN

Server-connection (for use by clients).

MQCHT_CLUSSDR

Cluster sender.

MQCHT_CLUSRCVR

Cluster receiver.

TransportType (MQLONG):

Transport type.

Transmission protocol to be used.

Note that the value will not have been checked if the channel was initiated from the other end.

The value is one of the following:

MQXPT_LU62

LU 6.2 transport protocol.

MQXPT_TCP

TCP/IP transport protocol.

MQXPT_NETBIOS

NetBIOS transport protocol.

This value is supported in the following environments: Windows.

MQXPT_SPX

SPX transport protocol.

This value is supported in the following environments: Windows, plus WebSphere MQ clients connected to these systems.

Desc (MQCHAR64):

Channel description.

This is a field that may be used for descriptive commentary. The content of the field is of no significance to Message Channel Agents. However, it should contain only characters that can be displayed. It cannot contain any null characters; if necessary, it is padded to the right with blanks. In a DBCS installation, the field can contain DBCS characters (subject to a maximum field length of 64 bytes).

Note: If this field contains characters that are not in the queue manager's character set (as defined by the *CodedCharSetId* queue manager attribute), those characters may be translated incorrectly if this field is sent to another queue manager.

The length of this field is given by MQ_CHANNEL_DESC_LENGTH.

QMgrName (MQCHAR48):

Queue-manager name.

For channels with a *ChannelType* other than MQCHT_CLNTCONN, this is the name of the queue manager that an exit can connect to, which on UNIX systems and Windows, is always nonblank.

The length of this field is given by MQ_Q_MGR_NAME_LENGTH.

XmitQName (MQCHAR48):

Transmission queue name.

The name of the transmission queue from which messages are retrieved.

This field is relevant only for channels with a *ChannelType* of MQCHT_SENDER or MQCHT_SERVER.

The length of this field is given by MQ_Q_NAME_LENGTH.

ShortConnectionName (MQCHAR20):

First 20 bytes of connection name.

If the *Version* field is MQCD_VERSION_1, *ShortConnectionName* contains the full connection name.

If the *Version* field is MQCD_VERSION_2 or greater, *ShortConnectionName* contains the first 20 characters of the connection name. The full connection name is given by the *ConnectionName* field; *ShortConnectionName* and the first 20 characters of *ConnectionName* are identical.

See *ConnectionName* for details of the contents of this field.

Note: The name of this field was changed for MQCD_VERSION_2 and subsequent versions of MQCD; the field was previously called *ConnectionName*.

The length of this field is given by MQ_SHORT_CONN_NAME_LENGTH.

MCAName (MQCHAR20):

Reserved.

This is a reserved field; its value is blank.

The length of this field is given by MQ_MCA_NAME_LENGTH.

ModeName (MQCHAR8):

LU 6.2 Mode name.

This field is relevant only if the transmission protocol (*TransportType*) is MQXPT_LU62, and the *ChannelType* is not MQCHT_SVRCONN or MQCHT_RECEIVER.

This field is always blank. The information is contained in the communications Side Object instead.

The length of this field is given by MQ_MODE_NAME_LENGTH.

TpName (MQCHAR64):

LU 6.2 transaction program name.

This field is relevant only if the transmission protocol (*TransportType*) is MQXPT_LU62, and the *ChannelType* is not MQCHT_SVRCONN or MQCHT_RECEIVER.

This field is always blank on platforms on which the information is contained in the communications Side Object instead.

The length of this field is given by MQ_TP_NAME_LENGTH.

BatchSize (MQLONG):

Batch size.

The maximum number of messages that can be sent through a channel before synchronizing the channel.

This field is not relevant for channels with a *ChannelType* of MQCHT_SVRCONN or MQCHT_CLNTCONN.

DiscInterval (MQLONG):

Disconnect interval.

The maximum time in seconds for which the channel waits for a message to arrive on the transmission queue, before terminating the channel. A value of zero causes the MCA to wait indefinitely.

For server-connection channels using the TCP protocol, the interval represents the client inactivity disconnect value, specified in seconds. If a server-connection has received no communication from its partner client for this duration, it will terminate the connection. The server-connection inactivity interval only applies between MQ API calls from a client, so no client will be disconnected during a long-running MQGET with wait call.

This attribute is not applicable for server-connection channels using protocols other than TCP.

This field is relevant only for channels with a *ChannelType* of MQCHT_SENDER, MQCHT_SERVER, MQCHT_CLUSSDR, MQCHT_CLUSRCVR or MQCHT_SVRCONN.

ShortRetryCount (MQLONG):

Short retry count.

This is the maximum number of attempts that are made to connect to the remote machine, at intervals specified by *ShortRetryInterval*, before the (normally longer) *LongRetryCount* and *LongRetryInterval* are used.

This field is relevant only for channels with a *ChannelType* of MQCHT_SENDER, MQCHT_SERVER, MQCHT_CLUSSDR, or MQCHT_CLUSRCVR.

ShortRetryInterval (MQLONG):

Short retry wait interval.

This is the maximum number of seconds to wait before reattempting connection to the remote machine. Note that the interval between retries may be extended if the channel has to wait to become active.

This field is relevant only for channels with a *ChannelType* of MQCHT_SENDER, MQCHT_SERVER, MQCHT_CLUSSDR, or MQCHT_CLUSRCVR.

LongRetryCount (MQLONG):

Long retry count.

This count is used after the count specified by *ShortRetryCount* has been exhausted. It specifies the maximum number of further attempts that are made to connect to the remote machine, at intervals specified by *LongRetryInterval*, before logging an error to the operator.

This field is relevant only for channels with a *ChannelType* of MQCHT_SENDER, MQCHT_SERVER, MQCHT_CLUSSDR, or MQCHT_CLUSRCVR.

LongRetryInterval (MQLONG):

Long retry wait interval.

This is the maximum number of seconds to wait before reattempting connection to the remote machine. Note that the interval between retries may be extended if the channel has to wait to become active.

This field is relevant only for channels with a *ChannelType* of MQCHT_SENDER, MQCHT_SERVER, MQCHT_CLUSSDR, or MQCHT_CLUSRCVR.

SecurityExit (MQCHARn):

Channel security exit name.

If this name is nonblank, the exit is called at the following times:

- Immediately after establishing a channel.
Before any messages are transferred, the exit is given the opportunity to instigate security flows to validate connection authorization.
- Upon receipt of a response to a security message flow.
Any security message flows received from the remote processor on the remote machine are given to the exit.
- At initialization and termination of the channel.

See "Exit name fields" on page 414 for a description of the content of this field in various environments.

The length of this field is given by MQ_EXIT_NAME_LENGTH.

Note: The value of this constant is environment specific.

MsgExit (MQCHARn):

Channel message exit name.

If this name is nonblank, the exit is called at the following times:

- Immediately after a message has been retrieved from the transmission queue (sender or server), or immediately before a message is put to a destination queue (receiver or requester).
The exit is given the entire application message and transmission queue header for modification.
- At initialization and termination of the channel.

This field is not relevant for channels with a *ChannelType* of MQCHT_SVRCONN or MQCHT_CLNTCONN; a message exit is never invoked for such channels.

See “Exit name fields” on page 414 for a description of the content of this field in various environments.

The length of this field is given by MQ_EXIT_NAME_LENGTH.

Note: The value of this constant is environment specific.

SendExit (MQCHARn):

Channel send exit name.

If this name is nonblank, the exit is called at the following times:

- Immediately before data is sent out on the network.
The exit is given the complete transmission buffer before it is transmitted. The contents of the buffer can be modified as required.
- At initialization and termination of the channel.

See “Exit name fields” on page 414 for a description of the content of this field in various environments.

The length of this field is given by MQ_EXIT_NAME_LENGTH.

Note: The value of this constant is environment specific.

ReceiveExit (MQCHARn):

Channel receive exit name.

If this name is nonblank, the exit is called at the following times:

- Immediately before the received network data is processed.
The exit is given the complete transmission buffer as received. The contents of the buffer can be modified as required.
- At initialization and termination of the channel.

See “Exit name fields” on page 414 for a description of the content of this field in various environments.

The length of this field is given by MQ_EXIT_NAME_LENGTH.

Note: The value of this constant is environment specific.

SeqNumberWrap (MQLONG):

Highest allowable message sequence number.

When this value is reached, sequence numbers wrap to start again at 1.

This value is non-negotiable and must match in both the local and remote channel definitions.

This field is not relevant for channels with a *ChannelType* of MQCHT_SVRCONN or MQCHT_CLNTCONN.

MaxMsgLength (MQLONG):

Maximum message length.

Specifies the maximum message length that can be transmitted on the channel. This is compared with the value for the remote channel and the actual maximum is the lower of the two values.

PutAuthority (MQLONG):

Put authority.

Specifies whether the user identifier in the context information associated with a message should be used to establish authority to put the message to the destination queue.

This field is relevant only for channels with a *ChannelType* of MQCHT_REQUESTER, MQCHT_RECEIVER, or MQCHT_CLUSRCVR. It is one of the following:

MQPA_DEFAULT

Default user identifier is used.

MQPA_CONTEXT

Context user identifier is used.

MQPA_ALTERNATE_OR_MCA

The user ID from the UserIdentifier field of the message descriptor is used. Any user ID received from the network is not used. This value is supported only on z/OS.

MQPA_ONLY_MCA

The default user ID is used. Any user ID received from the network is not used. This value is supported only on z/OS.

DataConversion (MQLONG):

Data conversion.

This specifies whether the sending message channel agent should attempt conversion of the application message data if the receiving message channel agent is unable to perform this conversion. This applies only to messages that are not segments of logical messages; the MCA never attempts to convert messages which are segments.

This field is relevant only for channels with a *ChannelType* of MQCHT_SENDER, MQCHT_SERVER, MQCHT_CLUSSDR, or MQCHT_CLUSRCVR. It is one of the following:

MQCDC_SENDER_CONVERSION

Conversion by sender.

MQCDC_NO_SENDER_CONVERSION

No conversion by sender.

SecurityUserData (MQCHAR32):

Channel security exit user data.

This is passed to the channel security exit in the *ExitData* field of the *ChannelExitParms* parameter (see MQ_CHANNEL_EXIT).

This field initially contains the data that was set in the channel definition. However, during the lifetime of this MCA instance, any changes made to the contents of this field by an exit of any type are preserved by the MCA, and made visible to subsequent invocations of exits (regardless of type) for this MCA instance. This applies to exits on different conversations. Such changes have no effect on the channel definition used by other MCA instances. Any characters (including binary data) can be used.

The length of this field is given by `MQ_EXIT_DATA_LENGTH`.

This field is not relevant in WebSphere MQ for i5/OS.

MsgUserData (MQCHAR32):

Channel message exit user data.

This is passed to the channel message exit in the *ExitData* field of the *ChannelExitParms* parameter (see `MQ_CHANNEL_EXIT`).

This field initially contains the data that was set in the channel definition. However, during the lifetime of this MCA instance, any changes made to the contents of this field by an exit of any type are preserved by the MCA, and made visible to subsequent invocations of exits (regardless of type) for this MCA instance. Such changes have no effect on the channel definition used by other MCA instances. Any characters (including binary data) can be used.

The length of this field is given by `MQ_EXIT_DATA_LENGTH`.

This field is not relevant in WebSphere MQ for i5/OS.

SendUserData (MQCHAR32):

Channel send exit user data.

This is passed to the channel send exit in the *ExitData* field of the *ChannelExitParms* parameter (see `MQ_CHANNEL_EXIT`).

This field initially contains the data that was set in the channel definition. However, during the lifetime of this MCA instance, any changes made to the contents of this field by an exit of any type are preserved by the MCA, and made visible to subsequent invocations of exits (regardless of type) for this MCA instance. This applies to exits on different conversations. Such changes have no effect on the channel definition used by other MCA instances. Any characters (including binary data) can be used.

The length of this field is given by `MQ_EXIT_DATA_LENGTH`.

This field is not relevant in WebSphere MQ for i5/OS.

ReceiveUserData (MQCHAR32):

Channel receive exit user data.

This is passed to the channel receive exit in the *ExitData* field of the *ChannelExitParms* parameter (see `MQ_CHANNEL_EXIT`).

This field initially contains the data that was set in the channel definition. However, during the lifetime of this MCA instance, any changes made to the contents of this field by an exit of any type are preserved by the MCA, and made visible to subsequent invocations of exits (regardless of type) for this MCA instance. This applies to exits on different conversations. Such changes have no effect on the channel definition used by other MCA instances. Any characters (including binary data) can be used.

The length of this field is given by `MQ_EXIT_DATA_LENGTH`.

This field is not relevant in WebSphere MQ for i5/OS.

The following fields in this structure are not present if *Version* is less than `MQCD_VERSION_2`.

UserIdentifier (MQCHAR12):

User identifier.

This is used by the message channel agent when attempting to initiate a secure SNA session with a remote message channel agent.

This field can be nonblank only on UNIX systems and Windows, and is relevant only for channels with a *ChannelType* of `MQCHT_SENDER`, `MQCHT_SERVER`, `MQCHT_REQUESTER` or `MQCHT_CLNTCONN`. On z/OS this field is not relevant.

The length of this field is given by `MQ_USER_ID_LENGTH`, however only the first 10 characters are used.

This field is not present when *Version* is less than `MQCD_VERSION_2`.

Password (MQCHAR12):

Password.

This is used by the message channel agent when attempting to initiate a secure SNA session with a remote message channel agent.

This field can be nonblank only on UNIX systems, and Windows, and is relevant only for channels with a *ChannelType* of `MQCHT_SENDER`, `MQCHT_SERVER`, `MQCHT_REQUESTER` or `MQCHT_CLNTCONN`. On z/OS this field is not relevant.

The length of this field is given by `MQ_PASSWORD_LENGTH`, however only the first 10 characters are used.

This field is not present if *Version* is less than `MQCD_VERSION_2`.

MCAUserIdentifier (MQCHAR12):

This field contains the first 12 bytes of the MCA user identifier. It is used to specify the user identifier for the message channel agent (MCA), and can be set by a security exit.

There are two fields that contain the MCA user identifier:

- *MCAUserIdentifier* contains the first 12 bytes of the MCA user identifier, and is padded with blanks if the identifier is shorter than 12 bytes. *MCAUserIdentifier* can be completely blank.
- *LongMCAUserIdPtr* points to the full MCA user identifier, which can be longer than 12 bytes. Its length is given by *LongMCAUserIdLength*. The full identifier contains no trailing blanks, and is not null-terminated. If the identifier is completely blank, *LongMCAUserIdLength* is zero, and the value of *LongMCAUserIdPtr* is undefined.

Note: *LongMCAUserIdPtr* is not present if *Version* is less than MQCD_VERSION_6.

If the MCA user identifier is nonblank, it specifies the user identifier to be used by the message channel agent for authorization to access WebSphere MQ resources. For channel types MQCHT_REQUESTER, MQCHT_RECEIVER, and MQCHT_CLUSRCVR, if PutAuthority is MQPA_DEFAULT this is the user identifier used for authorization checks for the put operation to destination queues.

If the MCA user identifier is blank, the message channel agent uses its default user identifier.

The MCA user identifier can be set by a security exit to indicate the user identifier that the message channel agent should use. The exit can change either *MCAUserIdentifier*, or the string pointed at by *LongMCAUserIdPtr*. If both are changed but differ from each other, the MCA uses *LongMCAUserIdPtr* in preference to *MCAUserIdentifier*. If the exit changes the length of the string addressed by *LongMCAUserIdPtr*, *LongMCAUserIdLength* must be set correspondingly. If the exit wishes to increase the length of the identifier, the exit must allocate storage of the required length, set that storage to the required identifier, and place the address of that storage in *LongMCAUserIdPtr*. The exit is responsible for freeing that storage when the exit is later invoked with the MQXR_TERM reason.

For channels with a *ChannelType* of MQCHT_SVRCONN, if *MCAUserIdentifier* in the channel definition is blank, any user identifier transferred from the client is copied into it. This user identifier (after any modification by the security exit at the server) is the one which the client application is assumed to be running under.

The MCA user identifier is not relevant for channels with a *ChannelType* of MQCHT_CLNTCONN.

This is an input/output field to the exit. The length of this field is given by MQ_USER_ID_LENGTH. This field is not present when *Version* is less than MQCD_VERSION_2.

MCAType (MQLONG):

Message channel agent type.

This is the type of the message channel agent program.

This field is relevant only for channels with a *ChannelType* of MQCHT_SENDER, MQCHT_SERVER, MQCHT_REQUESTER, MQCHT_CLUSSDR, or MQCHT_CLUSRCVR.

The value is one of the following:

MQMCAT_PROCESS

Process.

The message channel agent runs as a separate process.

MQMCAT_THREAD

Thread (i5/OS, UNIX, and Windows).

The message channel agent runs as a separate thread.

This field is not present when *Version* is less than MQCD_VERSION_2.

ConnectionName (MQCHAR264):

Connection name.

For cluster-receiver channels (when specified) CONNAME relates to the local queue manager, and for other channels it relates to the target queue manager. The value you specify depends on the transmission protocol (*TransportType*) to be used:

- For MQXPT_LU62, it is the fully-qualified name of the partner Logical Unit.
- For MQXPT_NETBIOS, it is the NetBIOS name defined on the remote machine.
- For MQXPT_TCP, it is either the host name, the network address of the remote machine specified in IPv4 dotted decimal or IPv6 hexadecimal format, or the local machine for cluster-receiver channels.
- For MQXPT_SPX, it is an SPX-style address comprising a 4-byte network address, a 6-byte node address, and a 2-byte socket number.

When defining a channel, this field is not relevant for channels with a *ChannelType* of MQCHT_SVRCONN or MQCHT_RECEIVER. However, when the channel definition is passed to an exit, this field contains the address of the partner, whatever the channel type.

The length of this field is given by MQ_CONN_NAME_LENGTH. This field is not present if *Version* is less than MQCD_VERSION_2.

RemoteUserIdentifier (MQCHAR12):

First 12 bytes of user identifier from partner.

There are two fields that contain the remote user identifier:

- *RemoteUserIdentifier* contains the first 12 bytes of the remote user identifier, and is padded with blanks if the identifier is shorter than 12 bytes. *RemoteUserIdentifier* can be completely blank.
- *LongRemoteUserIdPtr* points to the full remote user identifier, which can be longer than 12 bytes. Its length is given by *LongRemoteUserIdLength*. The full identifier contains no trailing blanks, and is not null-terminated. If the identifier is completely blank, *LongRemoteUserIdLength* is zero, and the value of *LongRemoteUserIdPtr* is undefined. *LongRemoteUserIdPtr* is not present if *Version* is less than MQCD_VERSION_6.

The remote user identifier is relevant only for channels with a *ChannelType* of MQCHT_CLNTCONN or MQCHT_SVRCONN.

- For a security exit on an MQCHT_CLNTCONN channel, this is a user identifier that has been obtained from the environment. The exit can choose to send it to the security exit at the server.

- For a security exit on an MQCHT_SVRCONN channel, this field may contain a user identifier which has been obtained from the environment at the client, if there is no client security exit. The exit may validate this user ID (possibly in conjunction with the password in *RemotePassword*) and update the value in *MCAUserIdentifier*.

If there is a security exit at the client, then this information can be obtained in a security flow from the client.

The length of this field is given by MQ_USER_ID_LENGTH. This field is not present if *Version* is less than MQCD_VERSION_2.

RemotePassword (MQCHAR12):

Password from partner.

This field contains valid information only if *ChannelType* is MQCHT_CLNTCONN or MQCHT_SVRCONN.

- For a security exit at an MQCHT_CLNTCONN channel, this is a password which has been obtained from the environment . The exit can choose to send it to the security exit at the server.
- For a security exit at an MQCHT_SVRCONN channel, this field may contain a password which has been obtained from the environment at the client, if there is no client security exit. The exit may use this to validate the user identifier in *RemoteUserIdentifier*.

If there is a security exit at the client, then this information can be obtained in a security flow from the client.

The length of this field is given by MQ_PASSWORD_LENGTH. This field is not present if *Version* is less than MQCD_VERSION_2.

The following fields in this structure are not present if *Version* is less than MQCD_VERSION_3.

MsgRetryExit (MQCHARn):

Channel message retry exit name.

The message retry exit is an exit that is invoked by the MCA when the MCA receives a completion code of MQCC_FAILED from an MQOPEN or MQPUT call. The purpose of the exit is to specify a time interval for which the MCA should wait before retrying the MQOPEN or MQPUT operation. Alternatively, the exit can decide that the operation should not be retried.

The exit is invoked for all reason codes that have a completion code of MQCC_FAILED — it is up to the exit to decide which reason codes it wants the MCA to retry, for how many attempts, and at what time intervals.

When the exit decides that the operation should not be retried any more, the MCA performs its normal failure processing; this includes generating an exception report message (if specified by the sender), and either placing the original message on the dead-letter queue or discarding the message (according to whether the sender specified MQRO_DEAD_LETTER_Q or MQRO_DISCARD_MSG, respectively). Note that failures involving the dead-letter queue (for example, dead-letter queue full) do not cause the message-retry exit to be invoked.

If the exit name is nonblank, the exit is called at the following times:

- Immediately before performing the wait prior to retrying a message
- At initialization and termination of the channel

See “Exit name fields” on page 414 for a description of the content of this field in various environments.

This field is relevant only for channels with a *ChannelType* of MQCHT_REQUESTER, MQCHT_RECEIVER, or MQCHT_CLUSRCVR.

The length of this field is given by MQ_EXIT_NAME_LENGTH.

Note: The value of this constant is environment specific.

This field is not present when *Version* is less than MQCD_VERSION_3.

MsgRetryUserData (MQCHAR32):

Channel message retry exit user data.

This is passed to the channel message-retry exit in the *ExitData* field of the *ChannelExitParms* parameter (see MQ_CHANNEL_EXIT).

This field initially contains the data that was set in the channel definition. However, during the lifetime of this MCA instance, any changes made to the contents of this field by an exit of any type are preserved by the MCA, and made visible to subsequent invocations of exits (regardless of type) for this MCA instance. Such changes have no effect on the channel definition used by other MCA instances. Any characters (including binary data) can be used.

This field is relevant only for channels with a *ChannelType* of MQCHT_REQUESTER, MQCHT_RECEIVER, or MQCHT_CLUSRCVR.

The length of this field is given by MQ_EXIT_DATA_LENGTH. This field is not present when *Version* is less than MQCD_VERSION_3.

This field is not relevant in WebSphere MQ for i5/OS.

MsgRetryCount (MQLONG):

Number of times MCA will try to put the message, after the first attempt has failed.

This indicates the number of times that the MCA will retry the open or put operation, if the first MQOPEN or MQPUT fails with completion code MQCC_FAILED. The effect of this attribute depends on whether *MsgRetryExit* is blank or nonblank:

- If *MsgRetryExit* is blank, the *MsgRetryCount* attribute controls whether the MCA attempts retries. If the attribute value is zero, no retries are attempted. If the attribute value is greater than zero, the retries are attempted at intervals given by the *MsgRetryInterval* attribute.

Retries are attempted only for the following reason codes:

- MQRC_PAGESET_FULL
- MQRC_PUT_INHIBITED

– MQRC_Q_FULL

For other reason codes, the MCA proceeds immediately to its normal failure processing, without retrying the failing message.

- If *MsgRetryExit* is nonblank, the *MsgRetryCount* attribute has no effect on the MCA; instead it is the message-retry exit which determines how many times the retry is attempted, and at what intervals; the exit is invoked even if the *MsgRetryCount* attribute is zero.

The *MsgRetryCount* attribute is made available to the exit in the MQCD structure, but the exit it not required to honor it — retries continue indefinitely until the exit returns MQXCC_SUPPRESS_FUNCTION in the *ExitResponse* field of MQCXP.

This field is relevant only for channels with a *ChannelType* of MQCHT_REQUESTER, MQCHT_RECEIVER, or MQCHT_CLUSRCVR.

This field is not present when *Version* is less than MQCD_VERSION_3.

MsgRetryInterval (MQLONG):

Minimum interval in milliseconds after which the open or put operation will be retried.

The effect of this attribute depends on whether *MsgRetryExit* is blank or nonblank:

- If *MsgRetryExit* is blank, the *MsgRetryInterval* attribute specifies the minimum period of time that the MCA will wait before retrying a message, if the first MQOPEN or MQPUT fails with completion code MQCC_FAILED. A value of zero means that the retry will be performed as soon as possible after the previous attempt. Retries are performed only if *MsgRetryCount* is greater than zero.

This attribute is also used as the wait time if the message-retry exit returns an invalid value in the *MsgRetryInterval* field in MQCXP.

- If *MsgRetryExit* is not blank, the *MsgRetryInterval* attribute has no effect on the MCA; instead it is the message-retry exit which determines how long the MCA should wait. The *MsgRetryInterval* attribute is made available to the exit in the MQCD structure, but the exit it not required to honor it.

The value is in the range 0 through 999 999 999.

This field is relevant only for channels with a *ChannelType* of MQCHT_REQUESTER, MQCHT_RECEIVER, or MQCHT_CLUSRCVR.

This field is not present when *Version* is less than MQCD_VERSION_3.

The following fields in this structure are not present if *Version* is less than MQCD_VERSION_4.

HeartbeatInterval (MQLONG):

Time in seconds between heartbeat flows.

The interpretation of this field depends on the channel type, as follows:

- For a channel type of MQCHT_SENDER, MQCHT_SERVER, MQCHT_RECEIVER, MQCHT_REQUESTER, MQCHT_CLUSSDR, or MQCHT_CLUSRCVR, this is the time in seconds between heartbeat flows

passed from the sending MCA when there are no messages on the transmission queue. This gives the receiving MCA the opportunity to quiesce the channel. To be useful, *HeartbeatInterval* should be significantly less than *DiscInterval*.

- For a channel type of MQCHT_CLNTCONN or MQCHT_SVRCONN with the MQCD Sharing Conversations field set to zero, this is the time in seconds between heartbeat flows passed from the server MCA when that MCA has issued an MQGET call with the MQGMO_WAIT option on behalf of a client application. This allows the server MCA to handle situations where the client connection fails during an MQGET with MQGMO_WAIT.
- For a channel type of MQCHT_CLNTCONN or MQCHT_SVRCONN with the MQCD Sharing Conversations field set to a non-zero value, this is the time in seconds between heartbeat flow when there are no data flows sent or received. This allows the channel to be quiesced efficiently.

The value is in the range 0 through 999 999. The value that is actually used is the larger of the values specified at the sending side and receiving side unless a value of 0 is specified at either side, in which case no heartbeat exchange occurs.

This is an input field to the exit. The field is not present if *Version* is less than MQCD_VERSION_4.

BatchInterval (MQLONG):

Batch duration.

This is the approximate time in milliseconds that a channel will keep a batch open, if fewer than *BatchSize* messages have been transmitted in the current batch.

If *BatchInterval* is greater than zero, the batch is terminated by whichever of the following occurs first:

- *BatchSize* messages have been sent, or
- *BatchInterval* milliseconds have elapsed since the start of the batch.

If *BatchInterval* is zero, the batch is terminated by whichever of the following occurs first:

- *BatchSize* messages have been sent, or
- the transmission queue becomes empty.

BatchInterval must be in the range zero through 999 999 999.

This field is relevant only for channels with a *ChannelType* of MQCHT_SENDER, MQCHT_SERVER, MQCHT_CLUSSDR, or MQCHT_CLUSRCVR.

This is an input field to the exit. The field is not present when *Version* is less than MQCD_VERSION_4.

NonPersistentMsgSpeed (MQLONG):

Speed at which nonpersistent messages are sent.

This specifies the speed at which nonpersistent messages travel through the channel.

This field is relevant only for channels with a *ChannelType* of MQCHT_SENDER, MQCHT_SERVER, MQCHT_RECEIVER, MQCHT_REQUESTER, MQCHT_CLUSSDR, or MQCHT_CLUSRCVR.

The value is one of the following:

MQNPMS_NORMAL

Normal speed.

If a channel is defined to be MQNPMS_NORMAL, nonpersistent messages travel through the channel at normal speed. This has the advantage that these messages will not be lost if there is a channel failure. Also, persistent and nonpersistent messages on the same transmission queue maintain their order relative to each other.

MQNPMS_FAST

Fast speed.

If a channel is defined to be MQNPMS_FAST, nonpersistent messages travel through the channel at fast speed. This improves the throughput of the channel, but means that nonpersistent messages will be lost if there is a channel failure. Also, it is possible for nonpersistent messages to jump ahead of persistent messages waiting on the same transmission queue, that is, the order of nonpersistent messages is not maintained relative to persistent messages. However the order of nonpersistent messages relative to each other is maintained. Similarly, the order of persistent messages relative to each other is maintained.

StrucLength (MQLONG):

Length of MQCD structure.

This is the length in bytes of the MQCD structure. The length does not include any of the strings addressed by pointer fields contained within the structure. The value is one of the following:

MQCD_LENGTH_4

Length of version-4 channel definition structure.

MQCD_LENGTH_5

Length of version-5 channel definition structure.

MQCD_LENGTH_6

Length of version-6 channel definition structure.

MQCD_LENGTH_7

Length of version-7 channel definition structure.

MQCD_LENGTH_8

Length of version-8 channel definition structure.

MQCD_LENGTH_9

Length of version-9 channel definition structure.

The following constant specifies the length of the current version:

MQCD_CURRENT_LENGTH

Length of current version of channel definition structure.

Note: These constants have values that are environment specific.

The field is not present if *Version* is less than MQCD_VERSION_4.

ExitNameLength (MQLONG):

Length of exit name.

This is the length in bytes of each of the names in the lists of exit names addressed by the *MsgExitPtr*, *SendExitPtr*, and *ReceiveExitPtr* fields. This length is not necessarily the same as MQ_EXIT_NAME_LENGTH.

This is an input field to the exit. The field is not present if *Version* is less than MQCD_VERSION_4.

ExitDataLength (MQLONG):

Length of exit user data.

This is the length in bytes of each of the user data items in the lists of exit user data items addressed by the *MsgUserDataPtr*, *SendUserDataPtr*, and *ReceiveUserDataPtr* fields. This length is not necessarily the same as MQ_EXIT_DATA_LENGTH.

This is an input field to the exit. The field is not present if *Version* is less than MQCD_VERSION_4.

MsgExitsDefined (MQLONG):

Number of message exits defined.

This is the number of channel message exits in the chain. It is greater than or equal to zero.

This is an input field to the exit. The field is not present if *Version* is less than MQCD_VERSION_4.

SendExitsDefined (MQLONG):

Number of send exits defined.

This is the number of channel send exits in the chain. It is greater than or equal to zero.

This is an input field to the exit. The field is not present if *Version* is less than MQCD_VERSION_4.

ReceiveExitsDefined (MQLONG):

Number of receive exits defined.

This is the number of channel receive exits in the chain. It is greater than or equal to zero.

This is an input field to the exit. The field is not present if *Version* is less than MQCD_VERSION_4.

MsgExitPtr (MQPTR):

Address of first *MsgExit* field.

If *MsgExitsDefined* is greater than zero, this is the address of the list of names of each channel message exit in the chain.

Each name is in a field of length *ExitNameLength*, padded to the right with blanks. There are *MsgExitsDefined* fields adjoining one another – one for each exit.

Any changes made to these names by an exit are preserved, although the message channel exit takes no explicit action – it does not change which exits are invoked.

If *MsgExitsDefined* is zero, this field is the null pointer.

On platforms where the programming language does not support the pointer data type, this field is declared as a byte string of the appropriate length.

This is an input field to the exit. The field is not present if *Version* is less than MQCD_VERSION_4.

MsgUserDataPtr (MQPTR):

Address of first *MsgUserData* field.

If *MsgExitsDefined* is greater than zero, this is the address of the list of user data items for each channel message exit in the chain.

Each user data item is in a field of length *ExitDataLength*, padded to the right with blanks. There are *MsgExitsDefined* fields adjoining one another – one for each exit. If the number of user data items defined is less than the number of exit names, undefined user data items are set to blanks. Conversely, if the number of user data items defined is greater than the number of exit names, the excess user data items are ignored and not presented to the exit.

Any changes made to these values by an exit are preserved. This allows one exit to pass information to another exit. No validation is carried out on any changes so, for example, binary data can be written to these fields if required.

If *MsgExitsDefined* is zero, this field is the null pointer.

On platforms where the programming language does not support the pointer data type, this field is declared as a byte string of the appropriate length.

This is an input field to the exit. The field is not present if *Version* is less than MQCD_VERSION_4.

SendExitPtr (MQPTR):

Address of first *SendExit* field.

If *SendExitsDefined* is greater than zero, this is the address of the list of names of each channel send exit in the chain.

Each name is in a field of length *ExitNameLength*, padded to the right with blanks. There are *SendExitsDefined* fields adjoining one another – one for each exit.

Any changes made to these names by an exit are preserved, although the message send exit takes no explicit action – it does not change which exits are invoked.

If *SendExitsDefined* is zero, this field is the null pointer.

On platforms where the programming language does not support the pointer data type, this field is declared as a byte string of the appropriate length.

This is an input field to the exit. The field is not present if *Version* is less than MQCD_VERSION_4.

SendUserDataPtr (MQPTR):

Address of first *SendUserData* field.

If *SendExitsDefined* is greater than zero, this is the address of the list of user data items for each channel message exit in the chain.

Each user data item is in a field of length *ExitDataLength*, padded to the right with blanks. There are *MsgExitsDefined* fields adjoining one another – one for each exit. If the number of user data items defined is less than the number of exit names, undefined user data items are set to blanks. Conversely, if the number of user data items defined is greater than the number of exit names, the excess user data items are ignored and not presented to the exit.

Any changes made to these values by an exit are preserved. This allows one exit to pass information to another exit. No validation is carried out on any changes so, for example, binary data can be written to these fields if required.

If *SendExitsDefined* is zero, this field is the null pointer.

On platforms where the programming language does not support the pointer data type, this field is declared as a byte string of the appropriate length.

This is an input field to the exit. The field is not present if *Version* is less than MQCD_VERSION_4.

ReceiveExitPtr (MQPTR):

Address of first *ReceiveExit* field.

If *ReceiveExitsDefined* is greater than zero, this is the address of the list of names of each channel receive exit in the chain.

Each name is in a field of length *ExitNameLength*, padded to the right with blanks. There are *ReceiveExitsDefined* fields adjoining one another – one for each exit.

Any changes made to these names by an exit are preserved, although the message channel exit takes no explicit action – it does not change which exits are invoked.

If *ReceiveExitsDefined* is zero, this field is the null pointer.

On platforms where the programming language does not support the pointer data type, this field is declared as a byte string of the appropriate length.

This is an input field to the exit. The field is not present if *Version* is less than MQCD_VERSION_4.

ReceiveUserDataPtr (MQPTR):

Address of first *ReceiveUserData* field.

If *ReceiveExitsDefined* is greater than zero, this is the address of the list of user data item for each channel receive exit in the chain.

Each user data item is in a field of length *ExitDataLength*, padded to the right with blanks. There are *ReceiveExitsDefined* fields adjoining one another – one for each exit. If the number of user data items defined is less than the number of exit names, undefined user data items are set to blanks. Conversely, if the number of user data items defined is greater than the number of exit names, the excess user data items are ignored and not presented to the exit.

Any changes made to these values by an exit are preserved. This allows one exit to pass information to another exit. No validation is carried out on any changes so, for example, binary data can be written to these fields if required.

If *ReceiveExitsDefined* is zero, this field is the null pointer.

On platforms where the programming language does not support the pointer data type, this field is declared as a byte string of the appropriate length.

This is an input field to the exit. The field is not present if *Version* is less than MQCD_VERSION_4.

The following fields in this structure are not present if *Version* is less than MQCD_VERSION_5.

ClusterPtr (MQPTR):

Address of a list of cluster names.

If *ClustersDefined* is greater than zero, this is the address of a list of cluster names. The channel belongs to each cluster listed.

This field is relevant only for channels with a *ChannelType* of MQCHT_CLUSSDR or MQCHT_CLUSRCVR.

This is an input field to the exit. The field is not present if *Version* is less than MQCD_VERSION_5.

ClustersDefined (MQLONG):

Number of clusters to which the channel belongs.

This is the number of cluster names pointed to by *ClusterPtr*. It is zero or greater.

This field is relevant only for channels with a *ChannelType* of MQCHT_CLUSSDR or MQCHT_CLUSRCVR.

This is an input field to the exit. The field is not present if *Version* is less than MQCD_VERSION_5.

NetworkPriority (MQLONG):

Network priority.

This is the priority of the network connection for this channel. When multiple paths to a particular destination are available, the path with the highest priority is chosen. The value is in the range 0 through 9; 0 is the lowest priority.

This field is relevant only for channels with a *ChannelType* of MQCHT_CLUSSDR or MQCHT_CLUSRCVR.

This is an input field to the exit. The field is not present if *Version* is less than MQCD_VERSION_5.

The following fields in this structure are not present if *Version* is less than MQCD_VERSION_6.

LongMCAUserIdLength (MQLONG):

Length of long MCA user identifier.

This is the length in bytes of the full MCA user identifier pointed to by *LongMCAUserIdPtr*.

This field is not relevant for channels with a *ChannelType* of MQCHT_CLNTCONN.

This is an input/output field to the exit. The field is not present if *Version* is less than MQCD_VERSION_6.

LongRemoteUserIdLength (MQLONG):

Length of long remote user identifier.

This is the length in bytes of the full remote user identifier pointed to by *LongRemoteUserIdPtr*.

This field is relevant only for channels with a *ChannelType* of MQCHT_CLNTCONN or MQCHT_SVRCONN.

This is an input field to the exit. The field is not present if *Version* is less than MQCD_VERSION_6.

LongMCAUserIdPtr (MQPTR):

Address of long MCA user identifier.

If *LongMCAUserIdLength* is greater than zero, this is the address of the full MCA user identifier. The length of the full identifier is given by *LongMCAUserIdLength*. The first 12 bytes of the MCA user identifier are also contained in the field *MCAUserIdentifier*.

See the description of the *MCAUserIdentifier* field for details of the MCA user identifier.

This field is not relevant for channels with a *ChannelType* of MQCHT_CLNTCONN.

This is an input/output field to the exit. The field is not present if *Version* is less than MQCD_VERSION_6.

LongRemoteUserIdPtr (MQPTR):

Address of long remote user identifier.

If *LongRemoteUserIdLength* is greater than zero, this is the address of the full remote user identifier. The length of the full identifier is given by *LongRemoteUserIdLength*. The first 12 bytes of the remote user identifier are also contained in the field *RemoteUserIdentifier*.

See the description of the *RemoteUserIdentifier* field for details of the remote user identifier.

This field is relevant only for channels with a *ChannelType* of MQCHT_CLNTCONN or MQCHT_SVRCONN.

This is an input field to the exit. The field is not present if *Version* is less than MQCD_VERSION_6.

MCASecurityId (MQBYTE40):

MCA security identifier.

This is the security identifier for the MCA.

This field is not relevant for channels with a *ChannelType* of MQCHT_CLNTCONN.

The following special value indicates that there is no security identifier:

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.

This is an input/output field to the exit. The length of this field is given by MQ_SECURITY_ID_LENGTH. This field is not present if *Version* is less than MQCD_VERSION_6.

RemoteSecurityId (MQBYTE40):

Remote security identifier.

This is the security identifier for the remote user.

This field is relevant only for channels with a *ChannelType* of MQCHT_CLNTCONN or MQCHT_SVRCONN.

The following special value indicates that there is no security identifier:

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.

This is an input field to the exit. The length of this field is given by MQ_SECURITY_ID_LENGTH. This field is not present if *Version* is less than MQCD_VERSION_6.

The following fields in this structure are not present if *Version* is less than MQCD_VERSION_7.

SSLCipherSpec (MQCHAR32):

If SSLCipherSpec is blank, the channel is not using SSL. If it is not blank, this field contains a string specifying the CypherSpec in use.

This parameter is valid for all channel types. It is supported on AIX, HP-UX, Linux, i5/OS, Solaris, Windows, and z/OS. It is valid only for channel types of a transport type (TRPTYPE) of TCP.

This is an input field to the exit. The length of this field is given by MQ_SSL_CIPHER_SPEC_LENGTH. The field is not present if *Version* is less than MQCD_VERSION_7.

SSLPeerNamePtr (MQPTR):

Address of the SSL peer name.

When a certificate is received during a successful SSL handshake, the Distinguished Name of the subject of the certificate is copied into the MQCD field accessed by SSLPeerNamePtr at the end of the channel which receives the certificate. It overwrites the SSLPeerName value for the channel if this is present in the local user's channel definition. If a security exit is specified at this end of the channel it will receive the Distinguished Name from the peer certificate in the MQCD.

This is an input field to the exit. The field is not present if *Version* is less than MQCD_VERSION_7.

SSLPeerNameLength (MQLONG):

Length of SSL peer name.

This is the length in bytes of SSL peer name pointed to by *SSLPeerNamePtr*.

This is an input field to the exit. The field is not present if *Version* is less than MQCD_VERSION_7.

SSLClientAuth (MQLONG):

Determines whether SSL client authentication is required.

This field is relevant only to SVRCONN channel definitions.

The value is one of the following:

MQSCA_REQUIRED

Client authentication required.

MQSCA_OPTIONAL

Client authentication optional.

This is an input field to the exit. The field is not present if *Version* is less than MQCD_VERSION_7.

KeepAliveInterval (MQLONG):

Keepalive interval.

This is the value passed to the communications stack for keepalive timing for the channel. The value is applicable for the TCP/IP and SPX communications protocols, though not all implementations support this parameter.

The value is in the range 0 through 99 999; the units are seconds. A value of zero indicates that channel keepalive is not enabled, although keepalive may still occur if TCP/IP keepalive (rather than channel keepalive) is enabled. The following special value is also valid:

MQKAI_AUTO

Automatic.

This indicates that the keepalive interval is calculated from the negotiated heartbeat interval, as follows:

- If the negotiated heartbeat interval is greater than zero, the keepalive interval that is used is the heartbeat interval plus 60 seconds.
- If the negotiated heartbeat interval is zero, the keepalive interval that is used is zero.
- On z/OS, TCP/IP keepalive occurs when TCPKEEP(YES) is specified on the queue manager object.
- In other environments, TCP/IP keepalive occurs when the KEEPALIVE=YES parameter is specified in the TCP stanza in the distributed queuing configuration file.

This field is relevant only for channels that have a *TransportType* of MQXPT_TCP or MQXPT_SPX.

This is an input field to the exit. The field is not present if *Version* is less than MQCD_VERSION_7.

LocalAddress (MQCHAR48):

Local communications address.

This is the local TCP/IP address defined for the channel for outbound communications, or blank if no specific address is defined for outbound communications. The address can optionally include a port number or range of port numbers. The format of this address is:

[ip-addr][(low-port[,high-port])]

where square brackets ([]) denote optional information, ip-addr is specified in IPv4 dotted decimal, IPv6 hexadecimal, or alphanumeric form, and low-port and high-port are port numbers enclosed in parentheses. All are optional.

A specific IP address, port, or port range for outbound communications is useful in recovery scenarios where a channel is restarted on a different TCP/IP stack.

LocalAddress is similar in form to *ConnectionName*, but should not be confused with it. *LocalAddress* specifies the characteristics of the local communications, whereas *ConnectionName* specifies how to reach a remote queue manager.

This field is relevant only for channels with a *TransportType* of MQXPT_TCP, and a *ChannelType* of MQCHT_SENDER, MQCHT_SERVER, MQCHT_REQUESTER, MQCHT_CLNTCONN, MQCHT_CLUSSDR, or MQCHT_CLUSRCVR.

The length of this field is given by MQ_LOCAL_ADDRESS_LENGTH. This field is not present if *Version* is less than MQCD_VERSION_7.

BatchHeartbeat (MQLONG):

Batch heartbeat interval.

This specifies the time interval that is used to trigger a batch heartbeat for the channel. Batch heartbeating allows sender channels to determine whether the remote channel instance is still active before going indoubt. A batch heartbeat occurs if a sender channel has not communicated with the remote channel instance within the specified time interval.

The value is in the range 0 through 999 999; the units are milliseconds. A value of zero indicates that batch heartbeating is not enabled.

This field is relevant only for channels that have a *ChannelType* of MQCHT_SENDER, MQCHT_SERVER, MQCHT_CLUSSDR, or MQCHT_CLUSRCVR.

This is an input field to the exit. The field is not present if *Version* is less than MQCD_VERSION_7.

HdrCompList [2] (MQLONG):

Header data compression list.

The list of header data compression techniques which are supported by the channel.

The list contains one or more of the following:

MQCOMPRESS_NONE

No header data compression is performed.

MQCOMPRESS_SYSTEM

Header data compression is performed.

Unused values in the array are set to MQCOMPRESS_NOT_AVAILABLE.

This is an input field to the exit. The field is not present if *Version* is less than MQCD_VERSION_8.

MsgCompList [16] (MQLONG):

Message data compression list.

The list of message data compression techniques which are supported by the channel.

The list contains one or more of the following:

MQCOMPRESS_NONE

No message data compression is performed.

MQCOMPRESS_RLE

Message data compression is performed using run-length encoding.

MQCOMPRESS_ZLIBFAST

Message data compression is performed using the zlib compression technique. A fast compression time is preferred.

MQCOMPRESS_ZLIBHIGH

Message data compression is performed using the zlib compression technique. A high level of compression is preferred.

Unused values in the array are set to MQCOMPRESS_NOT_AVAILABLE.

This is an input field to the exit. The field is not present if *Version* is less than MQCD_VERSION_8.

CLWLChannelRank (MQLONG):

Cluster workload channel rank.

The workload manager choose algorithm selects a destination with the highest rank. When the final destination is a queue manager on a different cluster, you can set the rank of intermediate gateway queue managers (at the intersection of neighbouring clusters) so the choose algorithm will correctly choose a destination queue manager nearer the final destination.

The value is in the range 0 through 9. The default is 0.

This is an input field to the exit. The field is not present if *Version* is less than MQCD_VERSION_8.

For further information, see WebSphere MQ Queue Manager Clusters.

CLWLChannelPriority (MQLONG):

Cluster workload channel priority.

The workload manager choose algorithm selects a destination with the highest priority from the set of destinations selected on the basis of rank. If there are two possible destination queue managers, this attribute can be used to make one queue manager failover onto the other queue manager. All the messages will go to the queue manager with the highest priority until that ends, then the messages will go to the queue manager with the next highest priority.

The value is in the range 0 through 9. The default is 0.

This is an input field to the exit. The field is not present if *Version* is less than MQCD_VERSION_8.

For further information, see WebSphere MQ Queue Manager Clusters.

CLWLChannelWeight (MQLONG):

Cluster workload channel weight.

The workload manager choose algorithm will use the channel's "weight" attribute to the skew the destination choice so that more messages can be sent to a particular machine. For example, you can give a channel on a large Unix server a larger "weight" than another channel on small desktop PC, and the choose algorithm will choose the Unix server more frequently than the PC.

The value is in the range 1 through 99. The default is 50.

This is an input field to the exit. The field is not present if *Version* is less than MQCD_VERSION_8.

For further information, see WebSphere MQ Queue Manager Clusters.

ChannelMonitoring (MQLONG):

The current level of Monitoring data collection for the channel.

This field is not relevant for channels with a ChannelType of MQCHT_CLNTCONN.

The value is one of the following:

- MQMON_OFF
- MQMON_LOW
- MQMON_MEDIUM
- MQMON_HIGH

This is an input field to the exit. It is not present if *Version* is less than MQCD_VERSION_8.

ChannelStatistics (MQLONG):

The current level of Statistics data collection for the channel.

This field is not relevant for channels with a ChannelType of MQCHT_CLNTCONN.

The value is one of the following:

- MQMON_OFF
- MQMON_LOW
- MQMON_MEDIUM
- MQMON_HIGH

This is an input field to the exit. It is not present if *Version* is less than MQCD_VERSION_8.

SharingConversations (MQLONG):

This field determines the maximum number of conversations that can share a channel instance associated with this channel.

This field is used on client connection and server-connection channels.

A value of 0 means that the channel operates as it did prior to WebSphere MQ Version 7.0 with respect to the following attributes::

- Conversation sharing
- Read ahead
- STOP CHANNEL(<channelname>) MODE(QUIESCE)
- Heartbeating
- Client asynchronous consume

A value of 1 is the minimum value for WebSphere MQ V7.0 behavior. Although only one conversation is allowed on the channel instance, read ahead, asynchronous consume, and the Version 7 behavior of CLNTCONN-SVRCONN heartbeating and quiescent channel stopping are available.

This is an input field to the exit. It is not present if *Version* is less than MQCD_VERSION_9.

The default value of this field is 10.

Note: *MaxInstances* and *MaxInstancesPerClient* limits applied to a channel restrict the number of channel instances, not the number of conversations that might be sharing those instances.

ClientChannelWeight (MQLONG):

Specifies a weighting to influence which client-connection channel definition is used.

The ClientChannelWeight attribute is used so that client channel definitions can be selected at random based on their weighting when more than one suitable definition is available. When a client issues an MQCONN requesting connection to a queue manager group, by specifying a queue manager name starting with an asterisk, and more than one suitable channel definition is available in the client channel definition table (CCDT), the definition to use is randomly selected based on the weighting, with any applicable ClientChannelWeight(0) definitions selected first in alphabetical order.

Specify a value in the range 0 – 99. The default is 0.

A value of 0 indicates that no load balancing is performed and applicable definitions are selected in alphabetical order. To enable load balancing choose a value in the range 1 - 99 where 1 is the lowest weighting and 99 is the highest. The distribution of messages between two or more channels with non-zero weightings is approximately proportional to the ratio of those weightings. For example, three channels with ClientChannelWeight values of 2, 4, and 14 are selected approximately 10%, 20%, and 70% of the time. This distribution is not guaranteed.

This attribute is valid for the client-connection channel type only.

This is an input field to the exit. The field is not present if *Version* is less than MQCD_VERSION_9.

ConnectionAffinity (MQLONG):

Specifies which client channel definition that the client applications use to connect to the queue manager if multiple connections are available.

Use this attribute when multiple applicable channel definitions are available.

The value is one of the following:

PREFERRED

The first connection in a process reading a client channel definition table (CCDT) creates a list of applicable definitions based on the weighting with any applicable CLNTWGHT(0) definitions first and in alphabetical order. Each connection in the process attempts to connect using the first definition in the list. If a connection is unsuccessful the next definition is used. Unsuccessful definitions with CLNTWGHT values other than 0 are moved to the end of the list. CLNTWGHT(0) definitions remain at the start of the list and are selected first for each connection.

Each client process with the same hostname always creates the same list.

For client applications written in C, C++, or the .NET programming framework (including fully managed .NET) the list is updated if the CCDT has been modified since the list was created.

This is the default value.

NONE

The first connection in a process reading a CCDT creates a list of applicable definitions. All connections in a process select an applicable definition based on the weighting with any applicable CLNTWGHT(0) definitions selected first in alphabetical order.

For client applications written in C, C++, or the .NET programming framework (including fully managed .NET) the list is updated if the CCDT has been modified since the list was created.

This attribute is valid for the client-connection channel type only.

This is an input field to the exit. The field is not present if *Version* is less than MQCD_VERSION_9.

PropertyControl (MQLONG):

This field determines how message properties are handled for messages that are retrieved from queues using the MQGET call with the MQGMO_PROPERTIES_AS_Q_DEF option.

The value can be:

MQPROP_COMPATIBILITY

If the message contains a property with a prefix of mcd., jms., usr., or mqext., all message properties are delivered to the application in an MQRFH2 header. Otherwise all properties of the message, except those contained in the message descriptor (or extension), are discarded and are no longer accessible to the application.

This is the default value; it allows applications, which expect JMS related properties to be in an MQRFH2 header in the message data, to continue to work unmodified.

MQPROP_NONE

All properties of the message, except those in the message descriptor (or extension), are removed from the message before the message is sent to the remote queue manager.

MQPROP_ALL

All properties of the message are included with the message when it is sent to the remote queue manager. The properties, except those in the message descriptor (or extension), are placed in one or more MQRFH2 headers in the message data.

This attribute is applicable to Sender, Server, Cluster Sender and Cluster Receiver channels.

MaxInstances (MQLONG):

This field specifies the maximum number of simultaneous instances of an individual server-connection channel that can be started.

This field is used only on server-connection channels.

The field can have a value in the range 0 to 999 999 999. A value of zero prevents all client access.

The default value of this field is 999 999 999.

If the value of this field is reduced to a number that is less than the number of instances of the server-connection channel that are currently running, then those running instances are not affected. However, new instances cannot start until sufficient existing instances have ceased to run so that the number of currently running instances is less than the value of the field.

MaxInstancesPerClient (MQLONG):

This field specifies the maximum number of simultaneous instances of an individual server-connection channel that can be started from a single client. In this context, connections that originate from the same remote network address are regarded as coming from the same client.

This field is used only on server-connection channels.

The field can have a value in the range 0 to 999 999 999. A value of zero prevents all client access.

The default value of this field is 999 999 999.

If the value of this field is reduced to a number that is less than the number of instances of the server-connection channel that are currently running from individual clients, then those running instances are not affected. However, new instances from any of those clients cannot start until sufficient existing instances have ceased to run such that the number of currently running instances, originating from the client attempting to start a new one, is less than the value of the field.

C declaration

This is the C declaration for the MQCD structure.

```

typedef struct tagMQCD MQCD;
struct tagMQCD {
    MQCHAR    ChannelName[20];        /* Channel definition name */
    MQLONG    Version;                /* Structure version number */
    MQLONG    ChannelType;            /* Channel type */
    MQLONG    TransportType;          /* Transport type */
    MQCHAR    Desc[64];                /* Channel description */
    MQCHAR    QMgrName[48];           /* Queue-manager name */
    MQCHAR    XmitQName[48];          /* Transmission queue name */
    MQCHAR    ShortConnectionName[20]; /* First 20 bytes of connection
                                        name */

    MQCHAR    MCAName[20];             /* Reserved */
    MQCHAR    ModeName[8];            /* LU 6.2 Mode name */
    MQCHAR    TpName[64];             /* LU 6.2 transaction program
                                        name */

    MQLONG    BatchSize;               /* Batch size */
    MQLONG    DiscInterval;           /* Disconnect interval */
    MQLONG    ShortRetryCount;        /* Short retry count */
    MQLONG    ShortRetryInterval;     /* Short retry wait interval */
    MQLONG    LongRetryCount;         /* Long retry count */
    MQLONG    LongRetryInterval;      /* Long retry wait interval */
    MQCHAR    SecurityExit[n];        /* Channel security exit name */
    MQCHAR    MsgExit[n];             /* Channel message exit name */
    MQCHAR    SendExit[n];            /* Channel send exit name */
    MQCHAR    ReceiveExit[n];         /* Channel receive exit name */
    MQLONG    SeqNumberWrap;          /* Highest allowable message
                                        sequence number */

    MQLONG    MaxMsgLength;           /* Maximum message length */
    MQLONG    PutAuthority;           /* Put authority */
    MQLONG    DataConversion;         /* Data conversion */
    MQCHAR    SecurityUserData[32];    /* Channel security exit user
                                        data */

    MQCHAR    MsgUserData[32];        /* Channel message exit user
                                        data */

    MQCHAR    SendUserData[32];       /* Channel send exit user data */
    MQCHAR    ReceiveUserData[32];    /* Channel receive exit user
                                        data */

    MQCHAR    UserIdentifier[12];     /* User identifier */
    MQCHAR    Password[12];           /* Password */
    MQCHAR    MCAUserIdentifier[12];  /* First 12 bytes of MCA user
                                        identifier */

    MQLONG    MCAType;                /* Message channel agent type */
    MQCHAR    ConnectionName[264];    /* Connection name */
    MQCHAR    RemoteUserIdentifier[12]; /* First 12 bytes of user
                                        identifier from partner */

    MQCHAR    RemotePassword[12];     /* Password from partner */
    MQCHAR    MsgRetryExit[n];        /* Channel message retry exit
                                        name */

    MQCHAR    MsgRetryUserData[32];   /* Channel message retry exit
                                        user data */

    MQLONG    MsgRetryCount;          /* Number of times MCA will try
                                        to put the message, after the
                                        first attempt has failed */

    MQLONG    MsgRetryInterval;       /* Minimum interval in
                                        milliseconds after which the
                                        open or put operation will be
                                        retried */

    MQLONG    HeartbeatInterval;      /* Time in seconds between
                                        heartbeat flows */
    MQLONG    BatchInterval;          /* Batch duration */
    MQLONG    NonPersistentMsgSpeed;  /* Speed at which nonpersistent
                                        messages are sent */

    MQLONG    StrucLength;             /* Length of MQCD structure */
    MQLONG    ExitNameLength;         /* Length of exit name */
    MQLONG    ExitDataLength;         /* Length of exit user data */
    MQLONG    MsgExitsDefined;        /* Number of message exits
                                        defined */
};

```

```

MQLONG    SendExitsDefined;          /* Number of send exits
                                        defined */
MQLONG    ReceiveExitsDefined;       /* Number of receive exits
                                        defined */
MQPTR     MsgExitPtr;                /* Address of first MsgExit
                                        field */
MQPTR     MsgUserDataPtr;            /* Address of first MsgUserData
                                        field */
MQPTR     SendExitPtr;               /* Address of first SendExit
                                        field */
MQPTR     SendUserDataPtr;           /* Address of first SendUserData
                                        field */
MQPTR     ReceiveExitPtr;            /* Address of first ReceiveExit
                                        field */
MQPTR     ReceiveUserDataPtr;        /* Address of first
                                        ReceiveUserData field */
MQPTR     ClusterPtr;                /* Address of a list of cluster
                                        names */
MQLONG    ClustersDefined;           /* Number of clusters to which
                                        the channel belongs */
MQLONG    NetworkPriority;            /* Network priority */
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 */
MQBYTE40  MCASecurityId;             /* MCA security identifier */
MQBYTE40  RemoteSecurityId;          /* Remote security identifier */
MQCHAR    SSLCipherSpec[32];         /* SSL CipherSpec */
MQPTR     SSLPeerNamePtr;            /* Address of SSL peer name */
MQLONG    SSLPeerNameLength;         /* Length of SSL peer name */
MQLONG    SSLClientAuth;             /* Whether SSL client
                                        authentication is required */
MQLONG    KeepAliveInterval;         /* Keepalive interval */
MQCHAR    LocalAddress[48];          /* Local communications
                                        address */

MQLONG    BatchHeartbeat;            /* Batch heartbeat interval */
MQLONG    HdrCompList[2];            /* Header data compression list */
MQLONG    MsgCompList[16];           /* Message data compression list */
MQLONG    CLWLChannelRank;           /* Channel rank */
MQLONG    CLWLChannelPriority;        /* Channel priority */
MQLONG    CLWLChannelWeight;         /* Channel weight */
MQLONG    ChannelMonitoring;         /* Channel Monitoring control */
MQLONG    ChannelStatistics;         /* Channel Statistics */
MQLONG    SharingConversations;       /* Limit on sharing conversations */
MQLONG    PropertyControl;           /* Message property control */
MQLONG    MaxInstances;              /* Limit on SVRCONN channel instances */
MQLONG    MaxInstancesPerClient;     /* Limit on SVRCONN channel instances
                                        per client */
MQLONG    ClientChannelWeight;        /* Client channel weight */
MQLONG    ConnectionAffinity;        /* Connection Affinity */
};

```

COBOL declaration

```

** MQCD structure
  10 MQCD.
** Channel definition name
  15 MQCD-CHANNELNAME PIC X(20).
** Structure version number
  15 MQCD-VERSION PIC S9(9) BINARY.
** Channel type
  15 MQCD-CHANNELTYPE PIC S9(9) BINARY.
** Transport type
  15 MQCD-TRANSPORTTYPE PIC S9(9) BINARY.

```

```

** Channel description
15 MQCD-DESC PIC X(64).
** Queue-manager name
15 MQCD-QMGRNAME PIC X(48).
** Transmission queue name
15 MQCD-XMITQNAME PIC X(48).
** First 20 bytes of connection name
15 MQCD-SHORTCONNECTIONNAME PIC X(20).
** Reserved
15 MQCD-MCANAME PIC X(20).
** LU 6.2 Mode name
15 MQCD-MODENAME PIC X(8).
** LU 6.2 transaction program name
15 MQCD-TPNAME PIC X(64).
** Batch size
15 MQCD-BATCHSIZE PIC S9(9) BINARY.
** Disconnect interval
15 MQCD-DISCINTERVAL PIC S9(9) BINARY.
** Short retry count
15 MQCD-SHORTRETRYCOUNT PIC S9(9) BINARY.
** Short retry wait interval
15 MQCD-SHORTRETRYINTERVAL PIC S9(9) BINARY.
** Long retry count
15 MQCD-LONGRETRYCOUNT PIC S9(9) BINARY.
** Long retry wait interval
15 MQCD-LONGRETRYINTERVAL PIC S9(9) BINARY.
** Channel security exit name
15 MQCD-SECURITYEXIT PIC X(n).
** Channel message exit name
15 MQCD-MSGEXIT PIC X(n).
** Channel send exit name
15 MQCD-SENDEXIT PIC X(n).
** Channel receive exit name
15 MQCD-RECEIVEEXIT PIC X(n).
** Highest allowable message sequence number
15 MQCD-SEQNUMBERWRAP PIC S9(9) BINARY.
** Maximum message length
15 MQCD-MAXMSGLLENGTH PIC S9(9) BINARY.
** Put authority
15 MQCD-PUTAUTHORITY PIC S9(9) BINARY.
** Data conversion
15 MQCD-DATACONVERSION PIC S9(9) BINARY.
** Channel security exit user data
15 MQCD-SECURITYUSERDATA PIC X(32).
** Channel message exit user data
15 MQCD-MSGUSERDATA PIC X(32).
** Channel send exit user data
15 MQCD-SENDUSERDATA PIC X(32).
** Channel receive exit user data
15 MQCD-RECEIVEUSERDATA PIC X(32).
** User identifier
15 MQCD-USERIDENTIFIER PIC X(12).
** Password
15 MQCD-PASSWORD PIC X(12).
** First 12 bytes of MCA user identifier
15 MQCD-MCAUSERIDENTIFIER PIC X(12).
** Message channel agent type
15 MQCD-MCATYPE PIC S9(9) BINARY.
** Connection name
15 MQCD-CONNECTIONNAME PIC X(264).
** First 12 bytes of user identifier from partner
15 MQCD-REMOTEUSERIDENTIFIER PIC X(12).
** Password from partner
15 MQCD-REMOTEPASSWORD PIC X(12).
** Channel message retry exit name
15 MQCD-MSGRETRYEXIT PIC X(n).
** Channel message retry exit user data

```

```

15 MQCD-MSGRETRYUSERDATA      PIC X(32).
**  Number of times MCA will try to put the message, after the first
**  attempt has failed
15 MQCD-MSGRETRYCOUNT        PIC S9(9) BINARY.
**  Minimum interval in milliseconds after which the open or put
**  operation will be retried
15 MQCD-MSGRETRYINTERVAL      PIC S9(9) BINARY.
**  Time in seconds between heartbeat flows
15 MQCD-HEARTBEATINTERVAL     PIC S9(9) BINARY.
**  Batch duration
15 MQCD-BATCHINTERVAL         PIC S9(9) BINARY.
**  Speed at which nonpersistent messages are sent
15 MQCD-NONPERSISTENTMSGSPD   PIC S9(9) BINARY.
**  Length of MQCD structure
15 MQCD-STRUCLength          PIC S9(9) BINARY.
**  Length of exit name
15 MQCD-EXITNAMELENGTH        PIC S9(9) BINARY.
**  Length of exit user data
15 MQCD-EXITDATALENGTH        PIC S9(9) BINARY.
**  Number of message exits defined
15 MQCD-MSGEXITSDEFINED        PIC S9(9) BINARY.
**  Number of send exits defined
15 MQCD-SENDEXITSDEFINED       PIC S9(9) BINARY.
**  Number of receive exits defined
15 MQCD-RECEIVEEXITSDEFINED    PIC S9(9) BINARY.
**  Address of first MSGEXIT field
15 MQCD-MSGEXITPTR            POINTER.
**  Address of first MSGUSERDATA field
15 MQCD-MSGUSERDATAPTR        POINTER.
**  Address of first SENDEXIT field
15 MQCD-SENDEXITPTR           POINTER.
**  Address of first SENDUSERDATA field
15 MQCD-SENDUSERDATAPTR       POINTER.
**  Address of first RECEIVEEXIT field
15 MQCD-RECEIVEEXITPTR        POINTER.
**  Address of first RECEIVEUSERDATA field
15 MQCD-RECEIVEUSERDATAPTR    POINTER.
**  Address of a list of cluster names
15 MQCD-CLUSTERPTR            POINTER.
**  Number of clusters to which the channel belongs
15 MQCD-CLUSTERSDEFINED        PIC S9(9) BINARY.
**  Network priority
15 MQCD-NETWORKPRIORITY        PIC S9(9) BINARY.
**  Length of long MCA user identifier
15 MQCD-LONGMCAUSERIDLENGTH    PIC S9(9) BINARY.
**  Length of long remote user identifier
15 MQCD-LONGREMOTEUSERIDLENGTH PIC S9(9) BINARY.
**  Address of long MCA user identifier
15 MQCD-LONGMCAUSERIDPTR       POINTER.
**  Address of long remote user identifier
15 MQCD-LONGREMOTEUSERIDPTR    POINTER.
**  MCA security identifier
15 MQCD-MCASECURITYID          PIC X(40).
**  Remote security identifier
15 MQCD-REMOTESECURITYID       PIC X(40).
**  SSL CipherSpec
15 MQCD-SSLCIPHERSPEC          PIC X(32).
**  Address of SSL peer name
15 MQCD-SSLPEERNAMEPTR         POINTER.
**  Length of SSL peer name
15 MQCD-SSLPEERNAMELENGTH      PIC S9(9) BINARY.
**  Whether SSL client authentication is required
15 MQCD-SSLCLIENTAUTH          PIC S9(9) BINARY.
**  Keepalive interval
15 MQCD-KEEPALIVEINTERVAL      PIC S9(9) BINARY.
**  Local communications address
15 MQCD-LOCALADDRESS           PIC X(48).

```

```

** Batch heartbeat interval
   15 MQCD-BATCHHEARTBEAT      PIC S9(9) BINARY.
** Header data compression list
   15 MQCD-HDRCOMPLIST         PIC S9(9) BINARY OCCURS 2.
** Message data compression list
   15 MQCD-MSGCOMPLIST         PIC S9(9) BINARY OCCURS 16.
** Channel rank
   15 MQCD-CLWLCHANNELRANK     PIC S9(9) BINARY.
** Channel priority
   15 MQCD-CLWLCHANNELPRIORITY PIC S9(9) BINARY.
** Channel weight
   15 MQCD-CLWLCHANNELWEIGHT   PIC S9(9) BINARY.
** Channel Monitoring control
   15 MQCD-CHANNELMONITORING   PIC S9(9) BINARY.
** Channel Statistics
   15 MQCD-CHANNELSTATISTICS   PIC S9(9) BINARY.
** Limit on sharing conversations
   15 MQCD-SHARINGCONVERSATIONS PIC S9(9) BINARY.
** Message property control
   15 MQCD-PROPERTYCONTROL     PIC S9(9) BINARY.
** Limit on SVRCONN channel instances
   15 MQCD-MAXINSTANCES        PIC S9(9) BINARY.
** Limit on SVRCONN channel instances per client
   15 MQCD-MAXINSTANCESPERCLIENT PIC S9(9) BINARY.
** Client channel weight
   15 MQCD-CLIENTCHANNELWEIGHT PIC S9(9) BINARY
** Channel affinity
   15 MQCD-CONNECTIONAFFINITY  PIC S9(9) BINARY

```

RPG declaration (ILE)

```

D*.1.....2.....3.....4.....5.....6.....7..
D* MQCD Structure
D*
D* Channel definition name
D CDCHN          1      20
D* Structure version number
D CDVER          21     24I 0
D* Channel type
D CDCHT          25     28I 0
D* Transport type
D CDTRT          29     32I 0
D* Channel description
D CDDDES         33     96
D* Queue-manager name
D CDQM           97     144
D* Transmission queue name
D CDXQ           145    192
D* First 20 bytes of connection name
D CDSCN          193    212
D* Reserved
D CDMCA          213    232
D* LU 6.2 Mode name
D CDMOD          233    240
D* LU 6.2 transaction program name
D CDTP           241    304
D* Batch size
D CDBS           305    308I 0
D* Disconnect interval
D CDDI           309    312I 0
D* Short retry count
D CDSRC          313    316I 0
D* Short retry wait interval
D CDSRI          317    320I 0
D* Long retry count
D CDLRC          321    324I 0
D* Long retry wait interval
D CDLRI          325    328I 0

```

```

D* Channel security exit name
D CDSCX          329   348
D* Channel message exit name
D CDMSX          349   368
D* Channel send exit name
D CDSNX          369   388
D* Channel receive exit name
D CDRCX          389   408
D* Highest allowable message sequence number
D CDSNW          409   412I 0
D* Maximum message length
D CDMML          413   416I 0
D* Put authority
D CDPA           417   420I 0
D* Data conversion
D CDDC           421   424I 0
D* Channel security exit user data
D CDSCD          425   456
D* Channel message exit user data
D CDMSD          457   488
D* Channel send exit user data
D CDSND          489   520
D* Channel receive exit user data
D CDRCd          521   552
D* User identifier
D CDUID          553   564
D* Password
D CDPW           565   576
D* First 12 bytes of MCA user identifier
D CDAUI          577   588
D* Message channel agent type
D CDCAT          589   592I 0
D* Connection name (characters 1 through 256)
D CDCON          593   848
D* Connection name (characters 257 through 264)
D CDCN2          849   856
D* First 12 bytes of user identifier from partner
D CDRUI          857   868
D* Password from partner
D CDRPW          869   880
D* Channel message retry exit name
D CDMRX          881   900
D* Channel message retry exit user data
D CDMRD          901   932
D* Number of times MCA will try to put the message, after the first
D* attempt has failed
D CDMRC          933   936I 0
D* Minimum interval in milliseconds after which the open or put
D* operation will be retried
D CDMRI          937   940I 0
D* Time in seconds between heartbeat flows
D CDHBI          941   944I 0
D* Batch duration
D CDBI           945   948I 0
D* Speed at which nonpersistent messages are sent
D CDNPM          949   952I 0
D* Length of MQCD structure
D CDLEN          953   956I 0
D* Length of exit name
D CDXNL          957   960I 0
D* Length of exit user data
D CDXDL          961   964I 0
D* Number of message exits defined
D CDMXD          965   968I 0
D* Number of send exits defined
D CDSXD          969   972I 0
D* Number of receive exits defined

```

D	CDRXD	973	976I	0
D*	Address of first CDMSX field			
D	CDMXP	977	992*	
D*	Address of first CDMSD field			
D	CDMUP	993	1008*	
D*	Address of first CDSNX field			
D	CDSXP	1009	1024*	
D*	Address of first CDSND field			
D	CDSUP	1025	1040*	
D*	Address of first CDRCX field			
D	CDRXP	1041	1056*	
D*	Address of first CDRCD field			
D	CDRUP	1057	1072*	
D*	Limit on sharing conversations			
D	CDSHC	1073	1076B	0
D*	Message property control			
D	CDPRO	1077	1080I	0
D*	Address of a list of cluster names			
D	CDCLP	1073	1088*	
D*	Number of clusters to which the channel belongs			
D	CDCLD	1089	1092I	0
D*	Network priority			
D	CDNP	1093	1096I	0
D*	Length of long MCA user identifier			
D	CDLML	1097	1100I	0
D*	Length of long remote user identifier			
D	CDLRL	1101	1104I	0
D*	Address of long MCA user identifier			
D	CDLMP	1105	1120*	
D*	Address of long remote user identifier			
D	CDLRP	1121	1136*	
D*	MCA security identifier			
D	CDMSI	1137	1176	
D*	Remote security identifier			
D	CDRSI	1177	1216	
D*	SSL CipherSpec			
D	CDSCS	1217	1248	
D*	Address of SSL peer name			
D	CDSPP	1249	1264*	
D*	Length of SSL peer name			
D	CDSPL	1265	1268I	0
D*	Whether SSL client authentication is required			
D	CDSCA	1269	1272I	0
D*	Keepalive interval			
D	CDKAI	1273	1276I	0
D*	Local communications address			
D	CDLAD	1277	1324	
D*	Batch heartbeat interval			
D	CDBHB	1325	1328I	0
D*	Header data compression list			
D	CDHCL	1329	1330I	0
D*	Message data compression list			
D	CDMCL	1331	1346I	0
D*	Channel Rank			
D	CDCWCR	1347	1350I	0
D*	Channel priority			
D	CDCWCP	1351	1354I	0
D*	Channel Weight			
D	CDCWCW	1355	1358I	0
D*	Channel Monitoring control			
D	CDCHLMON	1359	1362I	0
D*	Channel Statistics			
D	CDCHLST	1363	1366I	0
D*	Client channel weight			
D	CDCNW	1367	1380I	0
D*	Connection affinity			
D	CDCNA	1381	1384I	0

System/390 assembler declaration

MQCD	DSECT		
MQCD_CHANNELNAME	DS	CL20	Channel definition name
MQCD_VERSION	DS	F	Structure version number
MQCD_CHANNELTYPE	DS	F	Channel type
MQCD_TRANSPORTTYPE	DS	F	Transport type
MQCD_DESC	DS	CL64	Channel description
MQCD_QMGRNAME	DS	CL48	Queue-manager name
MQCD_XMITQNAME	DS	CL48	Transmission queue name
MQCD_SHORTCONNECTIONNAME	DS	CL20	First 20 bytes of connection name
*			
MQCD_MCANAME	DS	CL20	Reserved
MQCD_MODENAME	DS	CL8	LU 6.2 Mode name
MQCD_TPNAME	DS	CL64	LU 6.2 transaction program name
MQCD_BATCHSIZE	DS	F	Batch size
MQCD_DISCINTERVAL	DS	F	Disconnect interval
MQCD_SHORTRETRYCOUNT	DS	F	Short retry count
MQCD_SHORTRETRYINTERVAL	DS	F	Short retry wait interval
MQCD_LONGRETRYCOUNT	DS	F	Long retry count
MQCD_LONGRETRYINTERVAL	DS	F	Long retry wait interval
MQCD_SECURITYEXIT	DS	CLn	Channel security exit name
MQCD_MSGEXIT	DS	CLn	Channel message exit name
MQCD_SENDEXIT	DS	CLn	Channel send exit name
MQCD_RECEIVEEXIT	DS	CLn	Channel receive exit name
MQCD_SEQNUMBERWRAP	DS	F	Highest allowable message sequence number
*			
MQCD_MAXMSGLLENGTH	DS	F	Maximum message length
MQCD_PUTAUTHORITY	DS	F	Put authority
MQCD_DATACONVERSION	DS	F	Data conversion
MQCD_SECURITYUSERDATA	DS	CL32	Channel security exit user data
MQCD_MSGUSERDATA	DS	CL32	Channel message exit user data
MQCD_SENDUSERDATA	DS	CL32	Channel send exit user data
MQCD_RECEIVEUSERDATA	DS	CL32	Channel receive exit user data
MQCD_USERIDENTIFIER	DS	CL12	User identifier
MQCD_PASSWORD	DS	CL12	Password
MQCD_MCAUSERIDENTIFIER	DS	CL12	First 12 bytes of MCA user identifier
*			
MQCD_MCATYPE	DS	F	Message channel agent type
MQCD_CONNECTIONNAME	DS	CL264	Connection name
MQCD_REMOTEUSERIDENTIFIER	DS	CL12	First 12 bytes of user identifier from partner
*			
MQCD_REMOTEPASSWORD	DS	CL12	Password from partner
MQCD_MSGRETRYEXIT	DS	CLn	Channel message retry exit name
MQCD_MSGRETRYUSERDATA	DS	CL32	Channel message retry exit user data
*			
MQCD_MSGRETRYCOUNT	DS	F	Number of times MCA will try to put the message, after the first attempt has failed
*			
*			
MQCD_MSGRETRYINTERVAL	DS	F	Minimum interval in milliseconds after which the open or put operation will be retried
*			
*			
MQCD_HEARTBEATINTERVAL	DS	F	Time in seconds between heartbeat flows
*			
MQCD_BATCHINTERVAL	DS	F	Batch duration
MQCD_NONPERSISTENTMSGSPPEED	DS	F	Speed at which nonpersistent messages are sent
*			
MQCD_STRUCLLENGTH	DS	F	Length of MQCD structure
MQCD_EXITNAMELENGTH	DS	F	Length of exit name
MQCD_EXITDATALENGTH	DS	F	Length of exit user data
MQCD_MSGEXITSDEFINED	DS	F	Number of message exits defined
MQCD_SENDEXITSDEFINED	DS	F	Number of send exits defined
MQCD_RECEIVEEXITSDEFINED	DS	F	Number of receive exits defined
MQCD_MSGEXITPTR	DS	F	Address of first MSGEXIT field
MQCD_MSGUSERDATAPTR	DS	F	Address of first MSGUSERDATA field
*			
MQCD_SENDEXITPTR	DS	F	Address of first SENDEXIT field

MQCD_SENDUSERDATAPTR	DS	F	Address of first SENDUSERDATA field
* MQCD_RECEIVEEXITPTR	DS	F	Address of first RECEIVEEXIT field
* MQCD_RECEIVEUSERDATAPTR	DS	F	Address of first RECEIVEUSERDATA field
* MQCD_CLUSTERPTR	DS	F	Address of a list of cluster names
* MQCD_CLUSTERSDEFINED	DS	F	Number of clusters to which the channel belongs
MQCD_NETWORKPRIORITY	DS	F	Network priority
* MQCD_LONGMCAUSERIDLENGTH	DS	F	Length of long MCA user identifier
* MQCD_LONGREMOTEUSERIDLENGTH	DS	F	Length of long remote user identifier
* MQCD_LONGMCAUSERIDPTR	DS	F	Address of long MCA user identifier
* MQCD_LONGREMOTEUSERIDPTR	DS	F	Address of long remote user identifier
MQCD_MCASESECURITYID	DS	XL40	MCA security identifier
MQCD_REMOTESECURITYID	DS	XL40	Remote security identifier
MQCD_SSLCIPHERSPEC	DS	CL32	SSL CipherSpec
MQCD_SSLPEERNAMEPTR	DS	F	Address of SSL peer name
MQCD_SSLPEERNAMELENGTH	DS	F	Length of SSL peer name
MQCD_SSLCLIENTAUTH	DS	F	Whether SSL client authentication is required
* MQCD_KEEPALIVEINTERVAL	DS	F	Keepalive interval
MQCD_LOCALADDRESS	DS	CL48	Local communications address
MQCD_BATCHHEARTBEAT	DS	F	Batch heartbeat interval
MQCD_HDRCOMPLIST	DS	CL2	Header data compression list
MQCD_MSGCOMPLIST	DS	CL16	Message data compression list
MQCD_CLWLCHANNELRANK	DS	F	Channel rank
MQCD_CLWLCHANNELPRIORITY	DS	F	Channel priority
MQCD_CLWLCHANNELWEIGHT	DS	F	Channel weight
MQCD_CHANNELMONITORING	DS	F	Channel monitoring
MQCD_CHANNELSTATISTICS	DS	F	Channel statistics
MQCD_SHARINGCONVERSATIONS	DS	F	Limit on sharing conversations
* MQCD_PROPERTYCONTROL	DS	F	Message property control
* MQCD_SHARINGCONVERSATIONS	DS	F	Limit on sharing conversations
MQCD_PROPERTYCONTROL	DS	F	Message property control
MQCD_MAXINSTANCES	DS	F	Limit on SVRCONN chl instances
MQCD_MAXINSTANCESPERCLIENT	DS	F	Limit on SVRCONN chl instances per client
MQCD_CLIENTCHANNELWEIGHT	DS	F	Channel weight
MQCD_CONNECTIONAFFINITY	DS	F	Connection Affinity
MQCD_LENGTH	EQU	*-MQCD	
	ORG	MQCD	
MQCD_AREA	DS	CL(MQCD_LENGTH)	

Visual Basic declaration

In Visual Basic, the MQCD structure can be used with the MQCNO structure on the MQCONN call.

```

Type MQCD
  ChannelName As String*20 'Channel definition name'
  Version As Long 'Structure version number'
  ChannelType As Long 'Channel type'
  TransportType As Long 'Transport type'
  Desc As String*64 'Channel description'
  QMgrName As String*48 'Queue-manager name'
  XmitQName As String*48 'Transmission queue name'
  ShortConnectionName As String*20 'First 20 bytes of connection'
  'name'
  MCAName As String*20 'Reserved'

```

ModeName	As String*8	'LU 6.2 Mode name'
TpName	As String*64	'LU 6.2 transaction program name'
BatchSize	As Long	'Batch size'
DiscInterval	As Long	'Disconnect interval'
ShortRetryCount	As Long	'Short retry count'
ShortRetryInterval	As Long	'Short retry wait interval'
LongRetryCount	As Long	'Long retry count'
LongRetryInterval	As Long	'Long retry wait interval'
SecurityExit	As String*n	'Channel security exit name'
MsgExit	As String*n	'Channel message exit name'
SendExit	As String*n	'Channel send exit name'
ReceiveExit	As String*n	'Channel receive exit name'
SeqNumberWrap	As Long	'Highest allowable message' 'sequence number'
MaxMsgLength	As Long	'Maximum message length'
PutAuthority	As Long	'Put authority'
DataConversion	As Long	'Data conversion'
SecurityUserData	As String*32	'Channel security exit user data'
MsgUserData	As String*32	'Channel message exit user data'
SendUserData	As String*32	'Channel send exit user data'
ReceiveUserData	As String*32	'Channel receive exit user data'
UserIdentifier	As String*12	'User identifier'
Password	As String*12	'Password'
MCAUserIdentifier	As String*12	'First 12 bytes of MCA user' 'identifier'
MCAType	As Long	'Message channel agent type'
ConnectionName	As String*264	'Connection name'
RemoteUserIdentifier	As String*12	'First 12 bytes of user' 'identifier from partner'
RemotePassword	As String*12	'Password from partner'
MsgRetryExit	As String*n	'Channel message retry exit name'
MsgRetryUserData	As String*32	'Channel message retry exit user' 'data'
MsgRetryCount	As Long	'Number of times MCA will try to' 'put the message, after the' 'first attempt has failed'
MsgRetryInterval	As Long	'Minimum interval in' 'milliseconds after which the' 'open or put operation will be' 'retried'
HeartbeatInterval	As Long	'Time in seconds between' 'heartbeat flows'
BatchInterval	As Long	'Batch duration'
NonPersistentMsgSpeed	As Long	'Speed at which nonpersistent' 'messages are sent'
StrucLength	As Long	'Length of MQCD structure'
ExitNameLength	As Long	'Length of exit name'
ExitDataLength	As Long	'Length of exit user data'
MsgExitsDefined	As Long	'Number of message exits defined'
SendExitsDefined	As Long	'Number of send exits defined'
ReceiveExitsDefined	As Long	'Number of receive exits defined'
MsgExitPtr	As String*4	'Address of first MsgExit field'
MsgUserDataPtr	As String*4	'Address of first MsgUserData' 'field'
SendExitPtr	As String*4	'Address of first SendExit field'
SendUserDataPtr	As String*4	'Address of first SendUserData' 'field'
ReceiveExitPtr	As String*4	'Address of first ReceiveExit' 'field'
ReceiveUserDataPtr	As String*4	'Address of first' 'ReceiveUserData field'
ClusterPtr	As String*4	'Address of a list of cluster' 'names'
ClustersDefined	As Long	'Number of clusters to which the' 'channel belongs'
NetworkPriority	As Long	'Network priority'
LongMCAUserIdLength	As Long	'Length of long MCA user'

```

LongRemoteUserIdLength As Long      'identifier'
                                'Length of long remote user'
                                'identifier'
LongMCAUserIdPtr        As String*4  'Address of long MCA user'
                                'identifier'
LongRemoteUserIdPtr    As String*4  'Address of long remote user'
                                'identifier'
MCASecurityId          As String*40  'MCA security identifier'
RemoteSecurityId       As String*40  'Remote security identifier'
SSLCipherSpec          As String*32  'SSL CipherSpec'
SSLPeerNamePtr        As String*4   'Address of SSL peer name'
SSLPeerNameLength     As Long        'Length of SSL peer name'
SSLClientAuth         As Long        'Whether SSL client'
                                'authentication is required'
KeepAliveInterval     As Long        'Keepalive interval'
LocalAddress           As String*48  'Local communications address'
BatchHeartbeat        As Long        'Batch heartbeat interval'
HdrCompList           As Long2       'Header data compression list'
MsgCompList           As Long16     'Message data compression list'
CLWLChannelRank       As Long        'Channel Rank'
CLWLChannelPriority    As Long        'Channel priority'
CLWLChannelWeight     As Long        'Channel Weight'
ChannelMonitoring     As Long        'Channel Monitoring control'
ChannelStatistics     As Long        'Channel Statistics'
End Type

```

Changing MQCD fields in a channel exit

A channel exit can change fields in the MQCD. However, these changes are not generally acted on, except in the circumstances listed.

If a channel exit program changes a field in the MQCD data structure, the new value is generally ignored by the WebSphere MQ channel process. However, the new value remains in the MQCD and is passed to any remaining exits in an exit chain and to any conversation sharing the channel instance.

If SharingConversations is set to FALSE in the MQCXP structure, changes to certain fields can be acted on, depending on the type of exit program, the type of channel, and the exit reason code. The following table shows the fields that can be changed and have an effect on the behavior of the channel, and in what circumstances. If an exit program changes one of these fields in any other circumstances, or any field not listed, the new value is ignored by the channel process but remains in the MQCD and is passed to any remaining exits in an exit chain and to any conversation sharing the channel instance

For example, changes to ChannelName will be acted on when MQCXP SharingConversations is set to FALSE, it is changed by any type of exit program and for any type of channel but only when the exit is initializing: Changes to MCAUserIdentifier will be acted on when MQCXP SharingConversations is FALSE, for the various channel types and exit reason codes listed, but only if it is changed by a security exit.

Field	Exit Reason Code	Exit Type	Channel Type
ChannelName	MQXR_INIT	All	All
TransportType	MQXR_INIT	All	All
XmitQName	MQXR_INIT	All	SDR,RCVR
ModeName	MQXR_INIT	All	All
TpName	MQXR_INIT	All	All

BatchSize	MQXR_INIT	All	SDR, SVR, RCVR, RQSTR, CLUSSDR, CLUSRCVR
DiscInterval	MQXR_INIT	All	SDR, SVR, RCVR, RQSTR, CLUSSDR, CLUSRCVR
ShortRetryCount	MQXR_INIT	All	SDR, SVR, RCVR, RQSTR, CLUSSDR, CLUSRCVR
ShortRetryInterval	MQXR_INIT	All	SDR, SVR, RCVR, RQSTR, CLUSSDR, CLUSRCVR
LongRetryCount	MQXR_INIT	All	SDR, SVR, RCVR, RQSTR, CLUSSDR, CLUSRCVR
LongRetryInterval	MQXR_INIT	All	SDR, SVR, RCVR, RQSTR, CLUSSDR, CLUSRCVR
SeqNumberWrap	MQXR_INIT	All	SDR, SVR, RCVR, RQSTR, CLUSSDR, CLUSRCVR
MaxMsgLength	MQXR_INIT	All	All
PutAuthority	MQXR_INIT	All	SDR, SVR, RCVR, RQSTR, CLUSSDR, CLUSRCVR
DataConversion	MQXR_INIT	All	All
MCAUserIdentifier	MQXR_INIT, MQXR_INIT_SEC, MQXR_SEC_MSG, MQXR_SEC_PARMS	Security	SDR, SVR, RCVR, RQSTR, CLUSSDR, CLUSRCVR
ConnectionName	MQXR_INIT	All	SDR, SVR, RQSTR, CLNTCONN, CLUSSDR, CLUSRCVR
MsgRetryUserData	MQXR_INIT	All	RCVR, RQSTR, CLUSRCVR
MsgRetryCount	MQXR_INIT	All	RCVR, RQSTR, CLUSRCVR
MsgRetryInterval	MQXR_INIT	All	RCVR, RQSTR, CLUSRCVR
HeartbeatInterval	MQXR_INIT	All	All
BatchInterval	MQXR_INIT	All	SDR, SVR, CLUSSDR, CLUSRCVR

NonPersistentMsgSpeed	MQXR_INIT	All	SDR, SVR, RCVR, RQSTR, CLUSSDR, CLUSRCVR
MCASecurityId	MQXR_INIT, MQXR_INIT_SEC, MQXR_SEC_MSG, MQXR_SEC_PARMS	Security	SDR, SVR, RCVR, RQSTR, SVRCONN, CLUSSDR, CLUSRCVR
SSLCipherSpec	MQXR_INIT	All	All
SSLPeerNamePtr	MQXR_INIT	All	All
SSLPeerNameLength	MQXR_INIT	All	All
SSLClientAuth	MQXR_INIT	All	SVR, RCVR, RQSTR, SVRCONN, CLUSRCVR
KeepAliveInterval	MQXR_INIT	All	All
LocalAddress	MQXR_INIT	All	SDR, SVR, RQSTR, CLNTCONN, CLUSSDR, CLUSRCVR
BatchHeartbeat	MQXR_INIT	All	SDR, SVR, CLUSSDR, CLUSRCVR
HdrCompList	MQXR_INIT	All	All
MsgCompList	MQXR_INIT	All	All
ChannelMonitoring	MQXR_INIT	All	SDR, SVR, RCVR, RQSTR, SVRCONN, CLUSSDR, CLUSRCVR
ChannelStatistics	MQXR_INIT	All	SDR, SVR, RCVR, RQSTR, CLUSSDR, CLUSRCVR
SharingConversations	MQXR_INIT	All	SVRCONN, CLNTCONN
PropertyControl	MQXR_INIT	All	SDR, SVR, CLUSSDR, CLUSRCVR

MQCXP – Channel exit parameter

The MQCXP structure is passed to each type of exit called by a Message Channel Agent (MCA), client-connection channel, or server-connection channel. See MQ_CHANNEL_EXIT.

The fields described as “input to the exit” in the descriptions that follow are ignored by the channel when the exit returns control to the channel. The exit should not expect that any input fields that it changes in the channel exit parameter block will be preserved for its next invocation. Changes made to

input/output fields (for example, the *ExitUserArea* field), are preserved for invocations of that instance of the exit only. Such changes cannot be used to pass data between different exits defined on the same channel, or between the same exit defined on different channels.

Fields

StrucId (MQCHAR4):

Structure identifier.

The value must be:

MQCXP_STRUC_ID

Identifier for channel exit parameter structure.

For the C programming language, the constant `MQCXP_STRUC_ID_ARRAY` is also defined; this has the same value as `MQCXP_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 depends on the environment:

MQCXP_VERSION_1

Version-1 channel exit parameter structure.

MQCXP_VERSION_2

Version-2 channel exit parameter structure.

The field has this value in the following environments: HP OpenVMS, Compaq NonStop Kernel.

MQCXP_VERSION_3

Version-3 channel exit parameter structure.

The field has this value in the following environments: UNIX systems not listed elsewhere.

MQCXP_VERSION_4

Version-4 channel exit parameter structure.

MQCXP_VERSION_5

Version-5 channel exit parameter structure.

MQCXP_VERSION_6

Version-6 channel exit parameter structure.

MQCXP_VERSION_8

Version-8 channel exit parameter structure.

The field has this value in the following environments: z/OS, AIX, HP-UX, Linux, i5/OS, Solaris, Windows.

Fields that exist only in the more-recent versions of the structure are identified as such in the descriptions of the fields. The following constant specifies the version number of the current version:

MQCXP_CURRENT_VERSION

Current version of channel exit parameter structure.

The value of this constant depends on the environment (see above).

Note: When a new version of the MQCXP structure is introduced, the layout of the existing part is not changed. The exit should therefore check that the version number is equal to or greater than the lowest version which contains the fields that the exit needs to use.

This is an input field to the exit.

ExitId (MQLONG):

Type of exit.

This indicates the type of exit being called, and is set on entry to the exit routine. Possible values are:

MQXT_CHANNEL_SEC_EXIT

Channel security exit.

MQXT_CHANNEL_MSG_EXIT

Channel message exit.

MQXT_CHANNEL_SEND_EXIT

Channel send exit.

MQXT_CHANNEL_RCV_EXIT

Channel receive exit.

MQXT_CHANNEL_MSG_RETRY_EXIT

Channel message-retry exit.

MQXT_CHANNEL_AUTO_DEF_EXIT

Channel auto-definition exit.

On z/OS, this type of exit is supported only for channels of type MQCHT_CLUSSDR and MQCHT_CLUSRCVR.

This is an input field to the exit.

ExitReason (MQLONG):

Reason for invoking exit.

This indicates the reason why the exit is being called, and is set on entry to the exit routine. It is not used by the auto-definition exit. Possible values are:

MQXR_INIT

Exit initialization.

This indicates that the exit is being invoked for the first time. It allows the exit to acquire and initialize any resources that it may need (for example: memory).

MQXR_TERM

Exit termination.

This indicates that the exit is about to be terminated. The exit should free any resources that it may have acquired since it was initialized (for example: memory).

MQXR_MSG

Process a message.

This indicates that the exit is being invoked to process a message. This occurs for channel message exits only.

MQXR_XMIT

Process a transmission.

This occurs for channel send and receive exits only.

MQXR_SEC_MSG

Security message received.

This occurs for channel security exits only.

MQXR_INIT_SEC

Initiate security exchange.

This occurs for channel security exits only.

The receiver's security exit is always invoked with this reason immediately after being invoked with MQXR_INIT, to give it the opportunity to initiate a security exchange. If it declines the opportunity (by returning MQXCC_OK instead of MQXCC_SEND_SEC_MSG or MQXCC_SEND_AND_REQUEST_SEC_MSG), the sender's security exit is invoked with MQXR_INIT_SEC.

If the receiver's security exit does initiate a security exchange (by returning MQXCC_SEND_SEC_MSG or MQXCC_SEND_AND_REQUEST_SEC_MSG), the sender's security exit is never invoked with MQXR_INIT_SEC; instead it is invoked with MQXR_SEC_MSG to process the receiver's message. (In either case it is first invoked with MQXR_INIT.)

Unless one of the security exits requests termination of the channel (by setting *ExitResponse* to MQXCC_SUPPRESS_FUNCTION or MQXCC_CLOSE_CHANNEL), the security exchange must complete at the side that initiated the exchange. Therefore, if a security exit is invoked with MQXR_INIT_SEC and it does initiate an exchange, the next time the exit is invoked it will be with MQXR_SEC_MSG. This happens whether or not there is a security message for the exit to process. There will be a security message if the partner returns MQXCC_SEND_SEC_MSG or MQXCC_SEND_AND_REQUEST_SEC_MSG, but not if the partner returns MQXCC_OK or there is no security exit at the partner. If there is no security message to process, the security exit at the initiating end is re-invoked with a *DataLength* of zero.

MQXR_RETRY

Retry a message.

This occurs for message-retry exits only.

MQXR_AUTO_CLUSSDR

Automatic definition of a cluster-sender channel.

This occurs for channel auto-definition exits only.

MQXR_AUTO_RECEIVER

Automatic definition of a receiver channel.

This occurs for channel auto-definition exits only.

MQXR_AUTO_SVRCONN

Automatic definition of a server-connection channel.

This occurs for channel auto-definition exits only.

MQXR_AUTO_CLUSRCVR

Automatic definition of a cluster-receiver channel.

This occurs for channel auto-definition exits only.

MQXR_SEC_PARMS

Security parameters

This applies to security exits only and indicates that an MQCSP structure is being passed to the exit. For more information, see MQCSP – Security parameters

Note:

1. If you have more than one exit defined for a channel, they will each be invoked with MQXR_INIT when the MCA is initialized, and will each be invoked with MQXR_TERM when the MCA is terminated.
2. For the channel auto-definition exit, *ExitReason* is not set if *Version* is less than MQCXP_VERSION_4. The value MQXR_AUTO_SVRCONN is implied in this case.

This is an input field to the exit.

ExitResponse (MQLONG):

Response from exit.

This is set by the exit to communicate with the MCA. It must be one of the following:

MQXCC_OK

Exit completed successfully.

- For the channel security exit, this indicates that message transfer should now proceed normally.
- For the channel message retry exit, this indicates that the MCA should wait for the time interval returned by the exit in the *MsgRetryInterval* field in MQCXP, and then retry the message.

The *ExitResponse2* field may contain additional information.

MQXCC_SUPPRESS_FUNCTION

Suppress function.

- For the channel security exit, this indicates that the channel should be terminated.
- For the channel message exit, this indicates that the message is not to proceed any further towards its destination. Instead the MCA generates an exception report message (if one was requested by the sender of the original message), and places the message contained in the original buffer on the dead-letter queue (if the sender specified MQRO_DEAD_LETTER_Q), or discards it (if the sender specified MQRO_DISCARD_MSG).

For persistent messages, if the sender specified MQRO_DEAD_LETTER_Q, but the put to the dead-letter queue fails, or there is no dead-letter queue, the original message is left on the

transmission queue and the report message is not generated. The original message is also left on the transmission queue if the report message cannot be generated successfully.

The *Feedback* field in the MQDLH structure at the start of the message on the dead-letter queue indicates why the message was put on the dead-letter queue; this feedback code is also used in the message descriptor of the exception report message (if one was requested by the sender).

- For the channel message retry exit, this indicates that the MCA should not wait and retry the message; instead, the MCA continues immediately with its normal failure processing (the message is placed on the dead-letter queue or discarded, as specified by the sender of the message).
- For the channel auto-definition exit, either MQXCC_OK or MQXCC_SUPPRESS_FUNCTION must be specified. If neither of these is specified, MQXCC_SUPPRESS_FUNCTION is assumed by default and the auto-definition is abandoned.

This response is not supported for the channel send and receive exits.

MQXCC_SEND_SEC_MSG

Send security message.

This value can be set only by a channel security exit. It indicates that the exit has provided a security message which should be transmitted to the partner.

MQXCC_SEND_AND_REQUEST_SEC_MSG

Send security message that requires a reply.

This value can be set only by a channel security exit. It indicates

- that the exit has provided a security message which should be transmitted to the partner, and
- that the exit requires a response from the partner. If no response is received, the channel must be terminated, because the exit has not yet decided whether communications can proceed.

MQXCC_SUPPRESS_EXIT

Suppress exit.

- This value can be set by all types of channel exit other than a security exit or an auto-definition exit. It suppresses any further invocation of that exit (as if its name had been blank in the channel definition), until termination of the channel, when the exit is again invoked with an *ExitReason* of MQXR_TERM.
- If a message retry exit returns this value, message retries for subsequent messages are controlled by the *MsgRetryCount* and *MsgRetryInterval* channel attributes as normal. For the current message, the MCA performs the number of outstanding retries, at intervals given by the *MsgRetryInterval* channel attribute, but only if the reason code is one that the MCA would normally retry (see the *MsgRetryCount* field described in “MQCD – Channel definition” on page 413). The number of outstanding retries is the value of the *MsgRetryCount* attribute, less the number of times the exit returned MQXCC_OK for the current message; if this number is negative, no further retries are performed by the MCA for the current message.

MQXCC_CLOSE_CHANNEL

Close channel.

This value can be set by any type of channel exit except an auto-definition exit. It closes the channel.

This is an input/output field from the exit.

ExitResponse2 (MQLONG):

Secondary response from exit.

This is set to zero on entry to the exit routine. It can be set by the exit to provide further information to the WebSphere MQ channel functions. It is not used by the auto-definition exit.

The exit can set one or more of the following. If more than one is required, the values are added together. Combinations that are not valid are noted; other combinations are allowed.

MQXR2_PUT_WITH_DEF_ACTION

Put with default action.

This is set by the receiver's channel message exit. It indicates that the message is to be put with the MCA's default action, that is either the MCA's default user ID, or the context *UserIdentifier* in the MQMD (message descriptor) of the message.

The value of this constant is zero, which corresponds to the initial value set when the exit is invoked. The constant is provided for documentation purposes.

MQXR2_PUT_WITH_DEF_USERID

Put with default user identifier.

This can only be set by the receiver's channel message exit. It indicates that the message is to be put with the MCA's default user identifier.

MQXR2_PUT_WITH_MSG_USERID

Put with message's user identifier.

This can only be set by the receiver's channel message exit. It indicates that the message is to be put with the context *UserIdentifier* in the MQMD (message descriptor) of the message (this may have been modified by the exit).

Only one of MQXR2_PUT_WITH_DEF_ACTION, MQXR2_PUT_WITH_DEF_USERID, and MQXR2_PUT_WITH_MSG_USERID should be set.

MQXR2_USE_AGENT_BUFFER

Use agent buffer.

This indicates that any data to be passed on is in *AgentBuffer*, not *ExitBufferAddr*.

The value of this constant is zero, which corresponds to the initial value set when the exit is invoked. The constant is provided for documentation purposes.

MQXR2_USE_EXIT_BUFFER

Use exit buffer.

This indicates that any data to be passed on is in *ExitBufferAddr*, not *AgentBuffer*.

Only one of MQXR2_USE_AGENT_BUFFER and MQXR2_USE_EXIT_BUFFER should be set.

MQXR2_DEFAULT_CONTINUATION

Default continuation.

Continuation with the next exit in the chain depends on the response from the last exit invoked:

- If MQXCC_SUPPRESS_FUNCTION or MQXCC_CLOSE_CHANNEL are returned, no further exits in the chain are called.
- Otherwise, the next exit in the chain is invoked.

MQXR2_CONTINUE_CHAIN

Continue with the next exit.

MQXR2_SUPPRESS_CHAIN

Skip remaining exits in chain.

This is an input/output field to the exit.

Feedback (MQLONG):

Feedback code.

This is set to MQFB_NONE on entry to the exit routine.

If a channel message exit sets the *ExitResponse* field to MQXCC_SUPPRESS_FUNCTION, the *Feedback* field specifies the feedback code that identifies why the message was put on the dead-letter (undelivered-message) queue, and is also used to send an exception report if one has been requested. In this case, if the *Feedback* field is MQFB_NONE, the following feedback code is used:

MQFB_STOPPED_BY_MSG_EXIT

Message stopped by channel message exit.

The value returned in this field by channel security, send, receive, and message-retry exits is not used by the MCA.

The value returned in this field by auto-definition exits is not used if *ExitResponse* is MQXCC_OK, but otherwise is used for the *AuxErrorDataInt1* parameter in the event message.

This is an input/output field from the exit.

MaxSegmentLength (MQLONG):

Maximum segment length.

This is the maximum length in bytes that can be sent in a single transmission. It is not used by the auto-definition exit. It is of interest to a channel send exit, because this exit must ensure that it does not increase the size of a transmission segment to a value greater than *MaxSegmentLength*. The length includes the initial 8 bytes that the exit must not change. The value is negotiated between the WebSphere MQ channel functions when the channel is initiated. See “Writing and compiling channel-exit programs” on page 394 for more information about segment lengths.

The value in this field is not meaningful if *ExitReason* is MQXR_INIT.

This is an input field to the exit.

ExitUserArea (MQBYTE16):

Exit user area.

This is a field that is available for the exit to use.

It is initialized to binary zero before the first invocation of the exit (which has an *ExitReason* set to MQXR_INIT), and thereafter any changes made to this field by the exit are preserved across invocations of the exit.

The following value is defined:

MQXUA_NONE

No user information.

The value is binary zero for the length of the field.

For the C programming language, the constant MQXUA_NONE_ARRAY is also defined; this has the same value as MQXUA_NONE, but is an array of characters instead of a string.

The length of this field is given by MQ_EXIT_USER_AREA_LENGTH. This is an input/output field to the exit.

ExitData (MQCHAR32):

Exit data.

This is set on entry to the exit routine to information that WebSphere MQ channel functions took from the channel definition. If no such information is available, this field is all blanks.

The length of this field is given by MQ_EXIT_DATA_LENGTH.

This is an input field to the exit.

The following fields in this structure are not present if *Version* is less than MQCXP_VERSION_2.

MsgRetryCount (MQLONG):

Number of times the message has been retried.

The first time the exit is invoked for a particular message, this field has the value zero (no retries yet attempted). On each subsequent invocation of the exit for that message, the value is incremented by one by the MCA.

This is an input field to the exit. The value in this field is not meaningful if *ExitReason* is MQXR_INIT. The field is not present if *Version* is less than MQCXP_VERSION_2.

MsgRetryInterval (MQLONG):

Minimum interval in milliseconds after which the put operation should be retried.

The first time the exit is invoked for a particular message, this field contains the value of the *MsgRetryInterval* channel attribute. The exit can leave the value unchanged, or modify it to specify a different time interval in milliseconds. If the exit returns MQXCC_OK in *ExitResponse*, the MCA will wait for at least this time interval before retrying the MQOPEN or MQPUT operation. The time interval specified must be zero or greater.

The second and subsequent times the exit is invoked for that message, this field contains the value returned by the previous invocation of the exit.

If the value returned in the *MsgRetryInterval* field is less than zero or greater than 999 999 999, and *ExitResponse* is MQXCC_OK, the MCA ignores the *MsgRetryInterval* field in MQCXP and waits instead for the interval specified by the *MsgRetryInterval* channel attribute.

This is an input/output field to the exit. The value in this field is not meaningful if *ExitReason* is MQXR_INIT. The field is not present if *Version* is less than MQCXP_VERSION_2.

MsgRetryReason (MQLONG):

Reason code from previous attempt to put the message.

This is the reason code from the previous attempt to put the message; it is one of the MQRC_* values.

This is an input field to the exit. The value in this field is not meaningful if *ExitReason* is MQXR_INIT. The field is not present if *Version* is less than MQCXP_VERSION_2.

The following fields in this structure are not present if *Version* is less than MQCXP_VERSION_3.

HeaderLength (MQLONG):

Length of header information.

This field is relevant only for a message exit and a message-retry exit. The value is the length of the routing header structures at the start of the message data; these are the MQXQH structure, the MQMDE (message description extension header), and (for a distribution-list message) the MQDH structure and arrays of MQOR and MQPMR records that follow the MQXQH structure.

The message exit can examine this header information, and modify it if necessary, but the data that the exit returns must still be in the correct format. The exit must not, for example, encrypt or compress the header data at the sending end, even if the message exit at the receiving end makes compensating changes.

If the message exit modifies the header information in such a way as to change its length (for example, by adding another destination to a distribution-list message), it must change the value of *HeaderLength* correspondingly before returning.

This is an input/output field to the exit. The value in this field is not meaningful if *ExitReason* is MQXR_INIT. The field is not present if *Version* is less than MQCXP_VERSION_3.

PartnerName (MQCHAR48):

Partner Name.

The name of the partner, as follows:

- For SVRCONN channels, it is the logged-on user ID at the client.
- For all other types of channel, it is the queue-manager name of the partner.

When the exit is initialized this field is blank because the queue manager does not know the name of the partner until after initial negotiation has taken place.

This is an input field to the exit. The field is not present if *Version* is less than MQCXP_VERSION_3.

FAPLevel (MQLONG):

Negotiated Formats and Protocols level.

This is an input field to the exit. The field is not present if *Version* is less than MQCXP_VERSION_3.

CapabilityFlags (MQLONG):

Capability flags.

The following are defined:

MQCF_NONE

No flags.

MQCF_DIST_LISTS

Distribution lists supported.

This is an input field to the exit. The field is not present if *Version* is less than MQCXP_VERSION_3.

ExitNumber (MQLONG):

Exit number.

The ordinal number of the exit, within the type defined in *ExitId*. For example, if the exit being invoked is the third message exit defined, this field contains the value 3. If the exit type is one for which a list of exits cannot be defined (for example, a security exit), this field has the value 1.

This is an input field to the exit. The field is not present if *Version* is less than MQCXP_VERSION_3.

The following fields in this structure are not present if *Version* is less than MQCXP_VERSION_5.

ExitSpace (MQLONG):

Number of bytes in transmission buffer reserved for exit to use.

This field is relevant only for a send exit. It specifies the amount of space in bytes that the WebSphere MQ channel functions will reserve in the transmission buffer

for the exit to use. This allows the exit to add to the transmission buffer a small amount of data (typically not exceeding a few hundred bytes) for use by a complementary receive exit at the other end. The data added by the send exit must be removed by the receive exit.

The value is always zero on z/OS.

Note: This facility should not be used to send large amounts of data, as this may degrade performance, or even inhibit operation of the channel.

By setting *ExitSpace* the exit is guaranteed that there will always be at least that number of bytes available in the transmission buffer for the exit to use. However, the exit can use less than the amount reserved, or more than the amount reserved if there is space available in the transmission buffer. The exit space in the buffer is provided following the existing data.

ExitSpace can be set by the exit only when *ExitReason* has the value MQXR_INIT; in all other cases the value returned by the exit is ignored. On input to the exit, *ExitSpace* is zero for the MQXR_INIT call, and is the value returned by the MQXR_INIT call in other cases.

If the value returned by the MQXR_INIT call is negative, or there are fewer than 1024 bytes available in the transmission buffer for message data after reserving the requested exit space for all of the send exits in the chain, the MCA outputs an error message and closes the channel. Similarly, if during data transfer the exits in the send exit chain allocate more user space than they reserved such that fewer than 1024 bytes remain in the transmission buffer for message data, the MCA outputs an error message and closes the channel. The limit of 1024 allows the channel's control and administrative flows to be processed by the chain of send exits, without the need for the flows to be segmented.

This is an input/output field to the exit if *ExitReason* is MQXR_INIT, and an input field in all other cases. The field is not present if *Version* is less than MQCXP_VERSION_5.

SSLCertUserId (MQCHAR12):

This field specifies the UserId associated with the remote certificate. It is blank on all platforms except z/OS

This is an input field to the exit. The field is not present if *Version* is less than MQCXP_VERSION_6.

SSLRemCertIssNameLength (MQLONG):

This field contains the length in bytes of the full Distinguished Name of the issuer of the remote certificate pointed to by SSLCertRemoteIssuerNamePtr.

This is an input field to the exit. The field is not present if *Version* is less than MQCXP_VERSION_6. The value is zero if it is not an SSL channel.

SSLRemCertIssNamePtr (PMQVOID):

This field contains the address of the full Distinguished Name of the issuer of the remote certificate. Its value is the null pointer if it is not an SSL channel.

This is an input field to the exit. The field is not present if *Version* is less than MQCXP_VERSION_6.

SecurityParms (PMQCSP):

Address of the MQCSP structure used to specify a user ID and password. The initial value of this field is the null pointer.

This is an input/output field to the exit. The field is not present if *Version* is less than MQCXP_VERSION_6.

CurHdrCompression (MQLONG):

Current header data compression.

This field specifies which technique is currently being used to compress the header data.

It is set to one of the following:

MQCOMPRESS_NONE

No header data compression is performed.

MQCOMPRESS_SYSTEM

Header data compression is performed.

The value can be altered by a sending channel's message exit to one of the negotiated supported values accessed from the HdrCompList field of the MQCD. This enables the technique used to compress the header data to be decided for each message based on the content of the message. The altered value is used for the current message only. The channel ends if the attribute is altered to an unsupported value. The value is ignored if altered outside a sending channel's message exit.

This is an input/output field to the exit. The field is not present if *Version* is less than MQCXP_VERSION_6.

CurMsgCompression (MQLONG):

Current message compression.

This field specifies which technique is currently being used to compress the message data.

It is set to one of the following:

MQCOMPRESS_NONE

No header data compression is performed.

MQCOMPRESS_RLE

Message data compression is performed using run-length encoding.

MQCOMPRESS_ZLIBFAST

Message data compression is performed using the zlib compression technique. A fast compression time is preferred.

MQCOMPRESS_ZLIBHIGH

Message data compression is performed using the zlib compression technique. A high level of compression is preferred.

The value can be altered by a sending channel's message exit to one of the negotiated supported values accessed from the `MsgCompList` field of the MQCD. This enables the technique used to compress the message data to be decided for each message based on the content of the message. The altered value is used for the current message only. The channel ends if the attribute is altered to an unsupported value. The value is ignored if altered outside a sending channel's message exit.

This is an input/output field to the exit. The field is not present if *Version* is less than `MQCXP_VERSION_6`.

Hconn (MQHCONN):

Connection handle

This field contains the connection handle that the exit should use if it needs to make any MQI calls within the exit. This field is not relevant to exits running on client-connection channels, where it contains the value `MQHC_UNUSABLE_HCONN` (-1).

This is an input field to the exit. The field is not present if *Version* is less than `MQCXP_VERSION_7`.

SharingConversations (MQBOOL):

This field indicates whether the conversation is the only one that could currently be running on this channel instance, or whether more than one conversation could currently be running on this channel instance.

It also indicates whether the exit program is subject to the risk of the MQCD being altered by another exit program running at the same time.

This field is only relevant for exit programs running on client-connection or server-connection channels.

It is set to one of the following:

FALSE

The exit instance is the only exit instance that could currently be running on this channel instance. This allows the exit to safely update the MQCD fields without contention from other exits running on other channel instances. Whether changes to the MQCD fields are acted upon by the channel is defined by the table of MQCD fields in "Changing MQCD fields in a channel exit" on page 456.

TRUE The exit instance is not the only exit instance that could currently be running on this channel instance. Any changes made to the MQCD will not be acted upon by the channel, with the exception of those listed in the table of MQCD fields in "Changing MQCD fields in a channel exit" on page 456 for Exit Reasons other than `MQXR_INIT`. If this exit updates the MQCD fields, ensure there is no contention from other exits running on other conversations at the same time by providing serialization among the exits that run on this channel instance.

This is an input field to the exit. The field is not present if *Version* is less than `MQCXP_VERSION_7`.

C declaration

```
typedef struct tagMQCXP MQCXP;
struct tagMQCXP {
    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 code */
    MQLONG    MaxSegmentLength; /* Maximum segment length */
    MQBYTE16  ExitUserArea;     /* Exit user area */
    MQCHAR32  ExitData;         /* Exit data */
    MQLONG    MsgRetryCount;    /* Number of times the message has been
    retried */
    MQLONG    MsgRetryInterval; /* Minimum interval in milliseconds after
    which the put operation should be
    retried */
    MQLONG    MsgRetryReason;   /* Reason code from previous attempt to
    put the message */
    MQLONG    HeaderLength;     /* Length of header information */
    MQCHAR48  PartnerName;     /* Partner Name */
    MQLONG    FAPLevel;        /* Negotiated Formats and Protocols
    level */
    MQLONG    CapabilityFlags;  /* Capability flags */
    MQLONG    ExitNumber;       /* Exit number */
    MQLONG    ExitSpace;        /* Number of bytes in transmission buffer
    reserved for exit to use */
    MQCHAR12  SSLCertUserid;    /* User identifier associated
    with remote SSL certificate */
    MQLONG    SSLRemCertIssNameLength; /* Length of
    distinguished name of issuer
    of remote SSL certificate */
    MQPTR     SSLRemCertIssNamePtr; /* Address of
    distinguished name of issuer
    of remote SSL certificate */
    PMQVOID   SecurityParms;    /* Security parameters */
    MQLONG    CurHdrCompression; /* Header data compression
    used for current message */
    MQLONG    CurMsgCompression; /* Message data compression
    used for current message */
    MQHCONN   Hconn;           /* Connection handle */
    MQBOOL    SharingConversations; /* Multiple conversations
    possible on channel inst? */
};
```

COBOL declaration

```
** MQCXP structure
  10 MQCXP.
**   Structure identifier
  15 MQCXP-STRUCID          PIC X(4).
**   Structure version number
  15 MQCXP-VERSION        PIC S9(9) BINARY.
**   Type of exit
  15 MQCXP-EXITID         PIC S9(9) BINARY.
**   Reason for invoking exit
  15 MQCXP-EXITREASON     PIC S9(9) BINARY.
**   Response from exit
  15 MQCXP-EXITRESPONSE   PIC S9(9) BINARY.
**   Secondary response from exit
  15 MQCXP-EXITRESPONSE2  PIC S9(9) BINARY.
**   Feedback code
  15 MQCXP-FEEDBACK       PIC S9(9) BINARY.
**   Maximum segment length
  15 MQCXP-MAXSEGMENTLENGTH PIC S9(9) BINARY.
**   Exit user area
```

```

15 MQCXP-EXITUSERAREA    PIC X(16).
** Exit data
15 MQCXP-EXITDATA       PIC X(32).
** Number of times the message has been retried
15 MQCXP-MSGRETRYCOUNT PIC S9(9) BINARY.
** Minimum interval in milliseconds after which the put operation
** should be retried
15 MQCXP-MSGRETRYINTERVAL PIC S9(9) BINARY.
** Reason code from previous attempt to put the message
15 MQCXP-MSGRETRYREASON  PIC S9(9) BINARY.
** Length of header information
15 MQCXP-HEADERLENGTH   PIC S9(9) BINARY.
** Partner Name
15 MQCXP-PARTNERNAME     PIC X(48).
** Negotiated Formats and Protocols level
15 MQCXP-FAPLEVEL       PIC S9(9) BINARY.
** Capability flags
15 MQCXP-CAPABILITYFLAGS PIC S9(9) BINARY.
** Exit number
15 MQCXP-EXITNUMBER     PIC S9(9) BINARY.
** Number of bytes in transmission buffer reserved for exit to use
15 MQCXP-EXITSPACE      PIC S9(9) BINARY.
** User Id associated with remote certificate
15 MQCXP-SSLCERTUSERID  PIC X(12).
** Length of distinguished name of issuer of remote SSL
** certificate
15 MQCXP-SSLREMCERTISSNAMELENGTH PIC S9(9) BINARY.
** Address of distinguished name of issuer of remote SSL
** certificate
15 MQCXP-SSLREMCERTISSNAMEPTR  POINTER.
** Security parameters
15 MQCXP-SECURITYPARMS      PIC S9(18) BINARY.
** Header data compression used for current message
15 MQCXP-CURHDRCOMPRESSION  PIC S9(9) BINARY.
** Message data compression used for current message
15 MQCXP-CURMSGCOMPRESSION  PIC S9(9) BINARY.
** Connection handle
15 MQCXP-HCONN              PIC S9(9) BINARY.
** Multiple conversations possible on channel instance?
15 MQCXP-SHARINGCONVERSATIONS PIC S9(9) BINARY.

```

RPG declaration (ILE)

```

D*..1.....2.....3.....4.....5.....6.....7..
D* MQCXP Structure
D*
D* Structure identifier
D CXSID          1      4
D* Structure version number
D CXVER          5      8I 0
D* Type of exit
D CXXID          9      12I 0
D* Reason for invoking exit
D CXREA         13      16I 0
D* Response from exit
D CXRES         17      20I 0
D* Secondary response from exit
D CXRE2         21      24I 0
D* Feedback code
D CXFB          25      28I 0
D* Maximum segment length
D CXMSL         29      32I 0
D* Exit user area
D CXUA          33      48
D* Exit data
D CXDAT         49      80
D* Number of times the message has been retried
D CXMRC         81      84I 0

```

D* Minimum interval in milliseconds after which the put operation
D* should be retried
D CXMRI 85 88I 0
D* Reason code from previous attempt to put the message
D CXMRR 89 92I 0
D* Length of header information
D CXHDL 93 96I 0
D* Partner Name
D CXPNM 97 144
D* Negotiated Formats and Protocols level
D CXFAP 145 148I 0
D* Capability flags
D CXCAP 149 152I 0
D* Exit number
D CXEXN 153 156I 0
D* Number of bytes in transmission buffer reserved for exit to use
D CXHDL 157 160I 0
D* User identifier associated with remote SSL certificate
D CXSSLCU 161 172
D* Length of distinguished name of issuer of remote SSL certificate
D CXSRCINL 173 176I 0
D* Address of distinguished name of issuer of remote SSL certificate
D CXSRCINP 177 192*
D* Security parameters
D CXSECP 193 208*
D* Header data compression used for current message
D CXCHC 209 212I 0
D* Message data compression used for current message
D CXCMC 213 216I 0
D* Connection handle
D CXHCONN 217 220I 0
D* Multiple conversations possible on channel instance?
D CXSHARECONV 221 224I 0

System/390 assembler declaration

```

MQCXP          DSECT
MQCXP_STRUCID  DS   CL4  Structure identifier
MQCXP_VERSION  DS   F    Structure version number
MQCXP_EXITID   DS   F    Type of exit
MQCXP_EXITREASON DS   F    Reason for invoking exit
MQCXP_EXITRESPONSE DS   F    Response from exit
MQCXP_EXITRESPONSE2 DS   F    Secondary response from exit
MQCXP_FEEDBACK DS   F    Feedback code
MQCXP_MAXSEGMENTLENGTH DS   F    Maximum segment length
MQCXP_EXITUSERAREA DS   XL16 Exit user area
MQCXP_EXITDATA DS   CL32 Exit data
MQCXP_MSGRETRYCOUNT DS   F    Number of times the message has been
*                               retried
MQCXP_MSGRETRYINTERVAL DS   F    Minimum interval in milliseconds
*                               after which the put operation should
*                               be retried
MQCXP_MSGRETRYREASON DS   F    Reason code from previous attempt to
*                               put the message
MQCXP_HEADERLENGTH DS   F    Length of header information
MQCXP_PARTNERNAME DS   CL48 Partner Name
MQCXP_FAPLEVEL  DS   F    Negotiated Formats and Protocols
*                               level
MQCXP_CAPABILITYFLAGS DS   F    Capability flags
MQCXP_EXITNUMBER DS   F    Exit number
MQCXP_EXITSPACE DS   F    Number of bytes in transmission
*                               buffer reserved for exit to use
MQCXP_SSLCERTUSERID DS   CL12 User identifier associated with
*                               remote SSL certificate
MQCXP_SSLREMCERTISSNAMELENGTH DS   F    Length of distinguished name
*                               of issuer of remote SSL certificate
MQCXP_SSLREMCERTISSNAMEPTR DS   F    Address of distinguished name
*                               of issuer of remote SSL certificate

```

MQCXP_SECURITYPARMS	DS	F	Address of security parameters
MQCXP_CURHDRCOMPRESSION	DS	F	Header data compression used for
*			current message
MQCXP_CURMSGCOMPRESSION	DS	F	Message data compression used for
*			current message
MQCXP_HCONN	DS	F	Connection handle
MQCXP_SHARINGCONVERSATIONS	DS	F	Multiple conversations possible on
*			channel inst?
MQCXP_LENGTH	EQU	*-MQCXP	
	ORG	MQCXP	
MQCXP_AREA	DS	CL(MQCXP_LENGTH)	

MQXWD – Exit wait descriptor

The MQXWD structure is an input/output parameter on the MQXWAIT call.

This structure is supported only on z/OS.

Fields

StrucId (MQCHAR4):

Structure identifier.

The value must be:

MQXWD_STRUC_ID

Identifier for exit wait descriptor structure.

For the C programming language, the constant MQXWD_STRUC_ID_ARRAY is also defined; this has the same value as MQXWD_STRUC_ID, but is an array of characters instead of a string.

The initial value of this field is MQXWD_STRUC_ID.

Version (MQLONG):

Structure version number.

The value must be:

MQXWD_VERSION_1

Version number for exit wait descriptor structure.

The initial value of this field is MQXWD_VERSION_1.

Reserved1 (MQLONG):

Reserved.

This is a reserved field; its value must be zero.

This is an input field.

Reserved2 (MQLONG):

Reserved.

This is a reserved field; its value must be zero.

This is an input field.

Reserved3 (MQLONG):

Reserved.

This is a reserved field; its value must be zero.

This is an input field.

ECB (MQLONG):

Event control block to wait on.

This is the event control block (ECB) to wait on. It should be set to zero before the MQXWAIT call is issued; on successful completion it will contain the post code.

This is an input/output field.

C declaration

```
typedef struct tagMQXWD MQXWD;
struct tagMQXWD {
    MQCHAR4  StrucId;    /* Structure identifier */
    MQLONG   Version;   /* Structure version number */
    MQLONG   Reserved1; /* Reserved */
    MQLONG   Reserved2; /* Reserved */
    MQLONG   Reserved3; /* Reserved */
    MQLONG   ECB;       /* Event control block to wait on */
};
```

System/390 assembler declaration

```
MQXWD          DSECT
MQXWD_STRUCID  DS    CL4  Structure identifier
MQXWD_VERSION  DS    F    Structure version number
MQXWD_RESERVED1 DS    F    Reserved
MQXWD_RESERVED2 DS    F    Reserved
MQXWD_RESERVED3 DS    F    Reserved
MQXWD_ECB      DS    F    Event control block to wait on
*
MQXWD_LENGTH   EQU   *-MQXWD
               ORG   MQXWD
MQXWD_AREA     DS    CL(MQXWD_LENGTH)
```

Problem determination in DQM

This chapter explains the various aspects of problem determination and suggests methods of resolving problems. Some of the problems mentioned in this chapter are platform and installation specific. Where this is the case, it is made clear in the text.

Problem determination for the following scenarios is discussed:

- “Error message from channel control” on page 477
- “Ping” on page 477
- “Dead-letter queue considerations” on page 477
- “Validation checks” on page 478
- “In-doubt relationship” on page 478
- “Channel startup negotiation errors” on page 478

- “When a channel refuses to run” on page 479
- “Retrying the link” on page 481
- “Data structures” on page 482
- “User exit problems” on page 482
- “Disaster recovery” on page 482
- “Channel switching” on page 483
- “Connection switching” on page 483
- “Client problems” on page 483
- “Error logs” on page 484
- “Message monitoring” on page 485

Error message from channel control

Problems found during normal operation of the channels are reported to the system console and to the system log. In WebSphere MQ for Windows they are reported to the channel log. Problem diagnosis starts with the collection of all relevant information from the log, and analysis of this information to identify the problem.

However, this could be difficult in a network where the problem may arise at an intermediate system that is staging some of your messages. An error situation, such as transmission queue full, followed by the dead-letter queue filling up, would result in your channel to that site closing down.

In this example, the error message you receive in your error log will indicate a problem originating from the remote site, but may not be able to tell you any details about the error at that site.

You need to contact your counterpart at the remote site to obtain details of the problem, and to receive notification of that channel becoming available again.

Ping

Ping is useful in determining whether the communication link and the two message channel agents that make up a message channel are functioning across all interfaces.

Ping makes no use of transmission queues, but it does invoke some user exit programs. If any error conditions are encountered, error messages are issued.

To use ping, you can issue the MQSC command PING CHANNEL. On z/OS and i5/OS, you can also use the panel interface to select this option.

On UNIX platforms, Windows, and i5/OS, you can also use the MQSC command PING QMGR to test whether the queue manager is responsive to commands. See the WebSphere MQ Script (MQSC) Command Reference for more information about this.

Dead-letter queue considerations

In some WebSphere MQ implementations the dead-letter queue is referred to as an *undelivered-message queue*.

If a channel ceases to run for any reason, applications will probably continue to place messages on transmission queues, creating a potential overflow situation. Applications can monitor transmission queues to find the number of messages waiting to be sent, but this would not be a normal function for them to carry out.

When this occurs in a message-originating node, and the local transmission queue is full, the application's PUT fails.

When this occurs in a staging or destination node, there are three ways that the MCA copes with the situation:

1. By calling the message-retry exit, if one is defined.
2. By directing all overflow messages to a *dead-letter queue* (DLQ), returning an exception report to applications that requested these reports.

Note: In distributed-queuing management, if the message is too big for the DLQ, the DLQ is full, or the DLQ is not available, the channel stops and the message remains on the transmission queue. Ensure your DLQ is defined, available, and sized for the largest messages you handle.

3. By closing down the channel, if neither of the previous options succeeded.
4. By returning the undelivered messages back to the sending end and returning a full report to the reply-to queue (MQRC_EXCEPTION_WITH_FULL_DATA and MQRO_DISCARD_MSG).

If an MCA is unable to put a message on the DLQ:

- The channel stops
- Appropriate error messages are issued at the system consoles at both ends of the message channel
- The unit of work is backed out, and the messages reappear on the transmission queue at the sending channel end of the channel
- Triggering is disabled for the transmission queue

Validation checks

A number of validation checks are made when creating, altering, and deleting channels, and where appropriate, an error message returned.

Errors may occur when:

- A duplicate channel name is chosen when creating a channel
- Unacceptable data is entered in the channel parameter fields
- The channel to be altered is in doubt, or does not exist

In-doubt relationship

If a channel is in doubt, it is usually resolved automatically on restart, so the system operator does not need to resolve a channel manually in normal circumstances. See "In-doubt channels" on page 65 for information about this.

Channel startup negotiation errors

During channel startup, the starting end has to state its position and agree channel running parameters with the corresponding channel. It may happen that the two

ends cannot agree on the parameters, in which case the channel closes down with error messages being issued to the appropriate error logs.

When a channel refuses to run

If a channel refuses to run:

- Check that DQM and the channels have been set up correctly. This is a likely problem source if the channel has never run. Reasons could be:
 - A mismatch of names between sending and receiving channels (remember that uppercase and lowercase letters are significant)
 - Incorrect channel types specified
 - The sequence number queue (if applicable) is not available, or is damaged
 - The dead-letter queue is not available
 - The sequence number wrap value is different on the two channel definitions
 - A queue manager or communication link is not available
 - A receiver channel might be in STOPPED state
 - The connection might not be defined correctly
 - There might be a problem with the communications software (for example, is TCP running?)
- It is possible that an in-doubt situation exists, if the automatic synchronization on startup has failed for some reason. This is indicated by messages on the system console, and the status panel may be used to show channels that are in doubt.

The possible responses to this situation are:

- Issue a Resolve channel request with Backout or Commit.

You need to check with your remote link supervisor to establish the number of the last message or unit of work committed. Check this against the last number at your end of the link. If the remote end has committed a number, and that number is not yet committed at your end of the link, then issue a RESOLVE COMMIT command.

In all other cases, issue a RESOLVE BACKOUT command.

The effect of these commands is that backed out messages reappear on the transmission queue and are sent again, while committed messages are discarded.

If in doubt yourself, perhaps backing out with the probability of duplicating a sent message would be the safer decision.

- Issue a RESET command.

This command is for use when sequential numbering is in effect, and should be used with care. Its purpose is to reset the sequence number of messages and you should use it only after using the RESOLVE command to resolve any in-doubt situations.

- On WebSphere MQ for i5/OS, Windows, UNIX systems, and z/OS, there is no need for the administrator to choose a particular sequence number to ensure that the sequence numbers are put back in step. When a sender channel starts up after being reset, it informs the receiver that it has been reset and supplies the new sequence number that is to be used by both the sender and receiver.
- If the status of a receiver end of the channel is STOPPED, it can be reset by starting the receiver end.

Note: This does not start the channel, it merely resets the status. The channel must still be started from the sender end.

Triggered channels

If a triggered channel refuses to run, investigate the possibility of in-doubt messages here: “When a channel refuses to run” on page 479

Another possibility is that the trigger control parameter on the transmission queue has been set to NOTRIGGER by the channel. This happens when:

- There is a channel error.
- The channel was stopped because of a request from the receiver.
- The channel was stopped because of a problem on the sender that requires manual intervention.

After diagnosing and fixing the problem, start the channel manually.

An example of a situation where a triggered channel fails to start is as follows:

1. A transmission queue is defined with a trigger type of FIRST.
2. A message arrives on the transmission queue, and a trigger message is produced.
3. The channel is started, but stops immediately because the communications to the remote system are not available.
4. The remote system is made available.
5. Another message arrives on the transmission queue.
6. The second message does not increase the queue depth from zero to one, so no trigger message is produced (unless the channel is in RETRY state). If this happens, restart the channel manually.

On WebSphere MQ for z/OS, if the queue manager is stopped using MODE(FORCE) during channel initiator shutdown, it might be necessary to manually restart some channels after channel initiator restart.

Conversion failure

Another reason for the channel refusing to run could be that neither end is able to carry out necessary conversion of message descriptor data between ASCII and EBCDIC, and integer formats. In this instance, communication is not possible.

Network problems

When using LU 6.2, make sure that your definitions are consistent throughout the network. For example, if you have increased the RU sizes in your CICS Transaction Server for OS/390 or Communications Manager definitions, but you have a controller with a small MAXDATA value in its definition, the session may fail if you attempt to send large messages across the network. A symptom of this may be that channel negotiation takes place successfully, but the link fails when message transfer occurs.

When using TCP, if your channels are unreliable and your connections break, you can set a KEEPALIVE value for your system or channels. You do this using the SO_KEEPALIVE option to set a system-wide value, and on WebSphere MQ for z/OS, you can also use the KeepAlive Interval channel attribute (KAINT) to set channel-specific keepalive values. On WebSphere MQ for z/OS you can alternatively use the RCVTIME and RCVTMIN channel initiator parameters. These

options are discussed in “Checking that the other end of the channel is still available” on page 61, and “KeepAlive Interval (KAINT)” on page 84.

Adopting an MCA:

The Adopt MCA function enables WebSphere MQ to cancel a receiver channel and to start a new one in its place.

For more information about this function, see “Adopting an MCA” on page 62. For details of its parameters, see WebSphere MQ for z/OS System Setup Guide.

Registration time for DDNS:

When a group TCP/IP listener is started, it registers with DDNS. But there may be a delay until the address is available to the network. A channel that is started in this period, and which targets the newly registered generic name, fails with an ‘error in communications configuration’ message. The channel then goes into retry until the name becomes available to the network. The length of the delay will be dependent on the name server configuration used.

Dial-up problems

WebSphere MQ supports connection over dial-up lines but you should be aware that with TCP, some protocol providers assign a new IP address each time you dial in. This can cause channel synchronization problems because the channel cannot recognize the new IP addresses and so cannot ensure the authenticity of the partner. If you encounter this problem, you need to use a security exit program to override the connection name for the session.

This problem does not occur when a WebSphere MQ for i5/OS, UNIX systems, or Windows systems product is communicating with another product at the same level, because the queue manager name is used for synchronization instead of the IP address.

Retrying the link

An error scenario may occur that is difficult to recognize. For example, the link and channel may be functioning perfectly, but some occurrence at the receiving end causes the receiver to stop. Another unforeseen situation could be that the receiver system has run out of memory and is unable to complete a transaction.

You need to be aware that such situations can arise, often characterized by a system that appears to be busy but is not actually moving messages. You need to work with your counterpart at the far end of the link to help detect the problem and correct it.

Retry considerations

If a link failure occurs during normal operation, a sender or server channel program will itself start another instance, provided that:

1. Initial data negotiation and security exchanges are complete
2. The retry count in the channel definition is greater than zero

Note: For i5/OS, UNIX systems, and Windows, to attempt a retry a channel initiator must be running. In platforms other than WebSphere MQ for i5/OS, UNIX

systems, and Windows systems, this channel initiator must be monitoring the initiation queue specified in the transmission queue that the channel is using.

Shared channel recovery on z/OS:

See “Shared channel recovery” on page 283, which includes a table that shows the types of shared-channel failure and how each type is handled.

Data structures

Data structures are needed for reference when checking logs and trace entries during problem diagnosis. Details can be found in “Channel-exit calls and data structures” on page 405 and in the WebSphere MQ Application Programming Reference book.

User exit problems

The interaction between the channel programs and the user-exit programs has some error-checking routines, but this facility can only work successfully when the user exits obey the rules described in Chapter 6, “Further intercommunication considerations,” on page 375. When errors occur, the most likely outcome will be that the channel stops and the channel program issues an error message, together with any return codes from the user exit. Any errors detected on the user exit side of the interface can be determined by scanning the messages created by the user exit itself.

You might need to use a trace facility of your host system to identify the problem.

Disaster recovery

Disaster recovery planning is the responsibility of individual installations, and the functions performed may include the provision of regular system ‘snapshot’ dumps that are stored safely off-site. These dumps would be available for regenerating the system, should some disaster overtake it. If this occurs, you need to know what to expect of the messages, and the following description is intended to start you thinking about it.

First a recap on system restart. If a system fails for any reason, it may have a system log that allows the applications running at the time of failure to be regenerated by replaying the system software from a syncpoint forward to the instant of failure. If this occurs without error, the worst that can happen is that message channel syncpoints to the adjacent system may fail on startup, and that the last batches of messages for the various channels will be sent again. Persistent messages will be recovered and sent again, nonpersistent messages may be lost.

If the system has no system log for recovery, or if the system recovery fails, or where the disaster recovery procedure is invoked, the channels and transmission queues may be recovered to an earlier state, and the messages held on local queues at the sending and receiving end of channels may be inconsistent.

Messages may have been lost that were put on local queues. The consequence of this happening depends on the particular WebSphere MQ implementation, and the channel attributes. For example, where strict message sequencing is in force, the receiving channel detects a sequence number gap, and the channel closes down for

manual intervention. Recovery then depends upon application design, as in the worst case the sending application may need to restart from an earlier message sequence number.

Channel switching

A possible solution to the problem of a channel ceasing to run would be to have two message channels defined for the same transmission queue, but with different communication links. One message channel would be preferred, the other would be a replacement for use when the preferred channel is unavailable.

If triggering is required for these message channels, the associated process definitions must exist for each sender channel end.

To switch message channels:

- If the channel is triggered, set the transmission queue attribute NOTRIGGER.
- Ensure the current channel is inactive.
- Resolve any in-doubt messages on the current channel.
- If the channel is triggered, change the process attribute in the transmission queue to name the process associated with the replacement channel.

In this context, some implementations allow a channel to have a blank process object definition, in which case you may omit this step as the queue manager will find and start the appropriate process object.

- Restart the channel, or if the channel was triggered, set the transmission queue attribute TRIGGER.

Connection switching

Another solution would be to switch communication connections from the transmission queues.

To do this:

- If the sender channel is triggered, set the transmission queue attribute NOTRIGGER.
- Ensure the channel is inactive.
- Change the connection and profile fields to connect to the replacement communication link.
- Ensure that the corresponding channel at the remote end has been defined.
- Restart the channel, or if the sender channel was triggered, set the transmission queue attribute TRIGGER.

Client problems

A client application may receive an unexpected error return code, for example:

- Queue manager not available
- Queue manager name error
- Connection broken

Look in the client error log for a message explaining the cause of the failure. There may also be errors logged at the server, depending on the nature of the failure.

Terminating clients

Even though a client has terminated, it is still possible for its surrogate process to be holding its queues open. Normally this will only be for a short time until the communications layer notifies that the partner has gone.

Error logs

WebSphere MQ error messages are placed in different error logs depending on the platform. There are error logs for:

- Windows
- UNIX systems
- z/OS

Error logs for Windows

WebSphere MQ for Windows uses a number of error logs to capture messages concerning the operation of WebSphere MQ itself, any queue managers that you start, and error data coming from the channels that are in use.

The location the error logs are stored in depends on whether the queue manager name is known and whether the error is associated with a client.

- If the queue manager name is known and the queue manager is available:
<install directory>\QMGRS\QMGrName\ERRORS\AMQERR01.LOG
- If the queue manager is not available:
<install directory>\QMGRS\@SYSTEM\ERRORS\AMQERR01.LOG
- If an error has occurred with a client application:
<install directory>\ERRORS\AMQERR01.LOG

On Windows, you should also examine the Windows application event log for relevant messages.

Error logs on UNIX systems

WebSphere MQ on UNIX systems uses a number of error logs to capture messages concerning the operation of WebSphere MQ itself, any queue managers that you start, and error data coming from the channels that are in use. The location the error logs are stored in depends on whether the queue manager name is known and whether the error is associated with a client.

- If the queue manager name is known and the queue manager is available:
/var/mqm/qmgrs/QMGrName/errors/AMQERR01.LOG
- If the queue manager is not available:
/var/mqm/qmgrs/@SYSTEM/errors/AMQERR01.LOG
- If an error has occurred with a client application:
/var/mqm/errors/AMQERR01.LOG

Error logs on z/OS

Error messages are written to:

- The z/OS system console
- The channel-initiator job log

If you are using the z/OS message processing facility to suppress messages, the console messages may be suppressed. See the WebSphere MQ for z/OS Concepts and Planning Guide for more information.

Message monitoring

If a message does not reach its intended destination, you can use the WebSphere MQ display route application, available through the control command **dspmqrte**, to determine the route a message takes through the queue manager network and its final location.

The WebSphere MQ display route application is described in Monitoring WebSphere MQ.

Chapter 7. Queue name resolution

This appendix describes queue name resolution as performed by queue managers at both sending and receiving ends of a channel.

In larger networks, the use of queue managers has a number of advantages over other forms of communication. These advantages derive from the name resolution function in DQM and the main benefits are:

- Applications do not need to make routing decisions
- Applications do not need to know the network structure
- Network links are created by systems administrators
- Network structure is controlled by network planners
- Multiple channels can be used between nodes to partition traffic

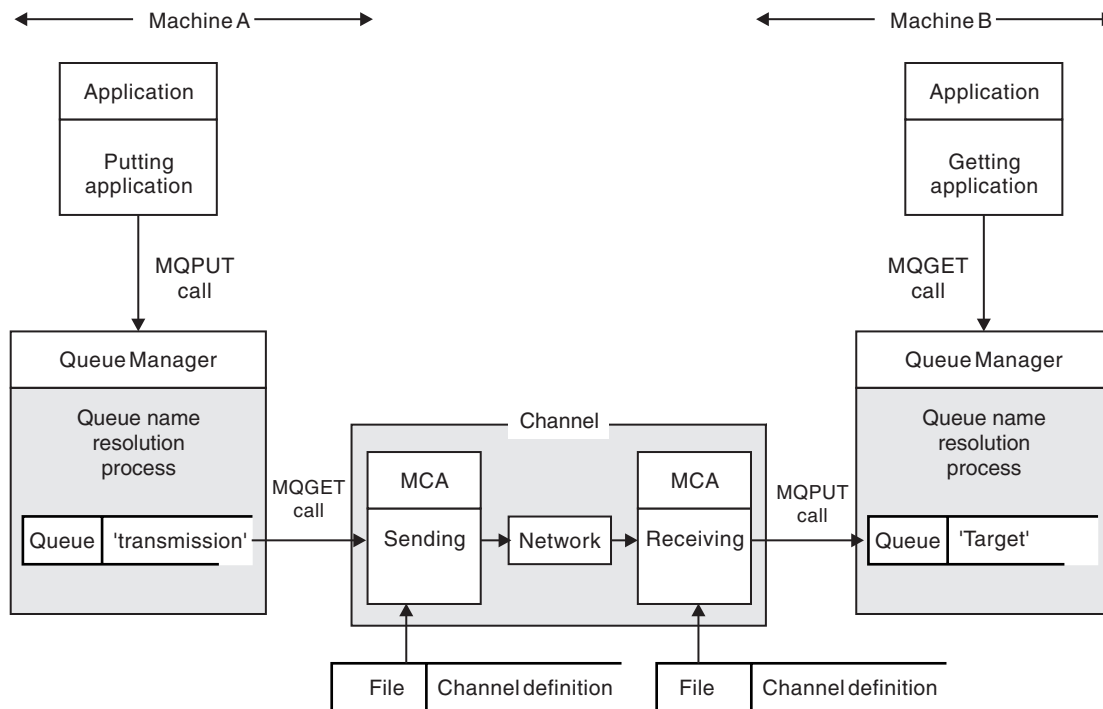


Figure 95. Name resolution

Referring to Figure 95, the basic mechanism for putting messages on a remote queue, as far as the application is concerned, is the same as for putting messages on a local queue:

- The application putting the message issues MQOPEN and MQPUT calls to put messages on the target queue.
- The application getting the messages issues MQOPEN and MQGET calls to get the messages from the target queue.

If both applications are connected to the same queue manager then no inter-queue manager communication is required, and the target queue is described as *local* to both applications.

However, if the applications are connected to different queue managers, two MCAs and their associated network connection are involved in the transfer, as shown in the figure. In this case, the target queue is considered to be a *remote queue* to the putting application.

The sequence of events is as follows:

1. The putting application issues MQOPEN and MQPUT calls to put messages to the target queue.
2. During the MQOPEN call, the *name resolution* function detects that the target queue is not local, and decides which transmission queue is appropriate. Thereafter, on the MQPUT calls associated with the MQOPEN call, all messages are placed on this transmission queue.
3. The sending MCA gets the messages from the transmission queue and passes them to the receiving MCA at the remote computer.
4. The receiving MCA puts the messages on the target queue, or queues.
5. The getting application issues MQOPEN and MQGET calls to get the messages from the target queue.

Note: Only step 1 and step 5 involve application code; steps 2 through 4 are performed by the local queue managers and the MCA programs. The putting application is unaware of the location of the target queue, which could be in the same processor, or in another processor on another continent.

The combination of sending MCA, the network connection, and the receiving MCA, is called a *message channel*, and is inherently a unidirectional device. Normally, it is necessary to move messages in both directions, and two channels are set up for this, one in each direction.

What is queue name resolution?

Queue name resolution is vital to DQM. It removes the need for applications to be concerned with the physical location of queues, and insulates them against the details of networks. A systems administrator can move queues from one queue manager to another, and change the routing between queue managers without applications needing to know anything about it.

In order to uncouple from the application design the exact path over which the data travels, it is necessary to introduce a level of indirection between the name used by the application when it refers to the target queue, and the naming of the channel over which the flow occurs. This indirection is achieved using the queue name resolution mechanism.

In essence, when an application refers to a queue name, the name is mapped by the resolution mechanism either to a transmission queue or to a local queue that is not a transmission queue. In the case of mapping to a transmission queue, a second name resolution is needed at the destination, and the received message is placed on the target queue as intended by the application designer. The application remains unaware of the transmission queue and channel used for moving the message.

Note: The definition of the queue and channel is a system management responsibility and can be changed by an operator or a system management utility, without the need to change applications.

An important requirement for the system management of message flows is that alternative paths should be provided between queue managers. For example, business requirements might dictate that different *classes of service* should be sent over different channels to the same destination. This is a system management decision and the queue name resolution mechanism provides a very flexible way to achieve it. The Application Programming Guide describes in detail how this is done, but the basic idea is to use queue name resolution at the sending queue manager to map the queue name supplied by the application to the appropriate transmission queue for the type of traffic involved. Similarly at the receiving end, queue name resolution maps the name in the message descriptor to a local (not a transmission) queue or again to an appropriate transmission queue.

Not only is it possible for the forward path from one queue manager to another to be partitioned into different types of traffic, but the return message that is sent to the reply-to queue definition in the outbound message can also use the same traffic partitioning. Queue name resolution satisfies this requirement and the application designer need not be involved in these traffic partitioning decisions.

The point that the mapping is carried out at both the sending and receiving queue managers is an important aspect of the way name resolution works. This allows the queue name supplied by the putting application to be mapped to a local queue or a transmission queue at the sending queue manager, and again remapped to a local queue or a transmission queue at the receiving queue manager.

Reply messages from receiving applications or MCAs have the name resolution carried out in exactly the same way, allowing return routing over specific paths by means of queue definitions at all the queue managers on route.

How queue name resolution works

The WebSphere MQ Application Programming Guide provides the rules for queue name resolution.

Chapter 8. Configuration file stanzas for distributed queuing

A description of the stanzas of the queue manager configuration file, `qm.ini`, related to distributed queuing.

This topic shows the stanzas in the queue manager configuration file that relate to distributed queuing. It applies to the queue manager configuration file for WebSphere MQ on UNIX systems, and for WebSphere MQ for i5/OS. The file is called `qm.ini` on both platforms.

Note: WebSphere MQ for Windows uses the registry. Use the WebSphere MQ Explorer to make equivalent changes to the configuration information.

The stanzas that relate to distributed queuing are:

- CHANNELS
- TCP
- LU62
- NETBIOS
- SPX (Windows XP and Windows 2003 Server only)
- EXITPATH

Figure 96 shows the values that you can set using these stanzas. When you are defining one of these stanzas, you do not need to start each item on a new line. You can use either a semicolon (;) or a hash character (#) to indicate a comment.

```
CHANNELS:
  MAXCHANNELS=n          ; Maximum number of channels allowed, the
                        ; default value is 100
  MAXACTIVECHANNELS=n   ; Maximum number of channels allowed to be active at
                        ; any time, the default is the value of MaxChannels
  MAXINITIATORS=n       ; Maximum number of initiators allowed, the
                        ; default and maximum value is 3
  MQIBINDTYPE=type1    ; Whether the binding for applications is to be
                        ; "fastpath" or "standard".
                        ;The default is "standard".
  ADOPTNEWMCA=ch1type   ; Stops previous process if channel fails to start.
                        ; The default is "NO".
  ADOPTNEWMCATIMEOUT=n ; Specifies the amount of time that the new
                        ; process should wait for the old process to end.
                        ; The default is 60.
  ADOPTNEWMCCHECK=     ; Specifies the type checking required.
  typecheck            ; The default is "NAME", "ADDRESS", and "QM".
TCP:
  PORT=n                ; TCP entries
                        ; Port number, the default is 1414
  KEEPALIVE=Yes         ; Switch TCP/IP KeepAlive on
  IPADDR=Addr/Name     ; TCP/IP address or name for Listener
  LIBRARY2=DLLName2    ; Same as above if code is in two libraries )
  EXITPATH:2          ; Location of user exits (MQSeries for AIX,
                        ; HP-UX, and Solaris only)
  EXITPATHS=           ; String of directory paths
```

Figure 96. `qm.ini` stanzas for distributed queuing

Note:

1. MQIBINDTYPE applies only to WebSphere MQ for AIX, WebSphere MQ for i5/OS, WebSphere MQ for HP-UX, and WebSphere MQ for Solaris.

2. EXITPATH applies only to WebSphere MQ for AIX, WebSphere MQ for HP-UX, and WebSphere MQ for Solaris.

For more information about the qm.ini file and the other stanzas in it, see WebSphere MQ System Administration Guide for WebSphere MQ for UNIX systems, and Windows systems, and WebSphere MQ for i5/OS System Administration Guide for WebSphere MQ for i5/OS.

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-k,u
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.

The following are trademarks of International Business Machines Corporation in the United States, or other countries, or both:

Advanced Peer-to-Peer Networking	AIX	CICS
DB2	DB2 Universal Database™	i5/OS
IBM	IBMLink™	iSeries™
MQSeries	MVS	MVS/ESA
OS/2	OS/390	OS/400
POWER	RACF	Redbooks
SupportPac™	System/390	VSE/ESA
VTAM	WebSphere	z/OS

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

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

A

active channels, maximum number 58
add routing entry 348
addressing information 21
addrtrge 348
administration, channel 53
Adopting an MCA 62
Adopting an MCA, information on 481
AFFINITY 79
agent
 intra-group queuing 300
AgentBuffer parameter 408
AgentBufferLength parameter 407
aliases 21
ALTDATA attribute 74
alter channel
 z/OS 247
alternate channels 14
ALTIME attribute 74
AMQCRCTA channel program 340
AMQCRS6A channel program 117
AMQCRSTA channel program 117, 340
AMQRMCLA channel program 340
APPC/MVS, defining a connection 266, 285
applications, trusted 11, 119
ARM (Automatic Restart Management) 262
assured delivery 20
attributes
 ALTDATA 74
 alter date 74
 alter time 74
 ALTIME 74
 batch heartbeat 74
 batch interval 74
 batch size 75
 BATCHHB 74
 BATCHINT 74
 BATCHSZ 75
 CHANNEL 76
 channel description 82
 channel name 76
 channel statistics 76
 channel type 77
 CHLTYPE 77
 CLUSNL 78
 CLUSTER 78
 cluster name 78
 cluster namelist 78
 CLWLPRTY 78
 CLWLRANK 79
 CLWLWGHT 79
 communication connection
 identifier 80
 COMPHDR 84
 COMPMSG 81
 CONNNAME 80
 connection name 80
 CONVERT 81
 convert message 81

attributes (*continued*)
 data compression 81
 DESCR 82
 DISCINT 82
 disconnect interval 82
 disposition 83
 HBINT 84
 header compression 84
 heartbeat interval 84
 KAINT 84
 Keepalive interval 84
 local address 85
 LOCALADDR 85
 long retry count 86
 long retry interval 87
 LONGRTY 86
 LONGTMR 87
 LU 6.2 mode name 87
 LU 6.2 TP name 88
 maximum instances 88
 maximum instances per client 89
 maximum message length 89
 MAXINST 88
 MAXINSTC 89
 MAXMSGL 89
 MCA name 89
 MCA type 90
 MCA user 90
 MCANAME 89
 MCATYPE 90
 MCAUSER 90
 message exit name 91
 message exit user data 91
 message retry count 91
 message retry interval 92
 message-retry exit name 91
 message-retry exit user data 91
 mode name 87
 MODENAME 87
 MONCHL 92
 monitoring 92
 MRDATA 91
 MREXIT 91
 MRRTY 91
 MRTMR 92
 MSGDATA 91
 MSGEXIT 91
 NETPRTY 93
 network-connection priority 93
 nonpersistent message speed 93
 NPMSPEED 93
 password 93
 priority 78
 PUT authority 94
 PUTAUT 94
 QMNAME 95
 QSGDISP 83
 queue manager name 95
 rank 79
 RCVDATA 96
 RCVEXIT 95

attributes (*continued*)
 receive exit name 95
 receive exit user data 96
 SCYDATA 96
 SCYEXIT 96
 security exit name 96
 security exit user data 96
 send exit name 97
 send exit user data 97
 SENDDATA 97
 SENDEXIT 97
 sequence number wrap 97
 SEQWRAP 97
 short retry count 97
 short retry interval 98
 SHORTRTY 97
 SHORTTMR 98
 SSL Cipher Specification 98
 SSL Client Authentication 99
 SSL Peer 99
 SSLCAUTH 99
 SSLCIPH 98
 SSLPEER 99
 STATCHL 76
 TPNAME 88
 transmission protocol 100
 transmission queue name 100
 transport type 100
 TRPTYPE 100
 user ID 100
 USERID 100
 weighting 79
 XMITQ 100
Attributes
 channel 71
 authority, PUT 94
 auto-definition exit program 393
 auto-definition of channels 54
 automatic channel reconnect for
 TCP/IP 262
 Automatic Restart Management
 (ARM) 262

B

back out in-doubt messages
 i5/OS 336
 UNIX systems 114
 Windows systems 114
balanced
 Workload 283
batch heartbeat 74
batch interval 74
batch size 75
BATCHHB attribute 74
BatchHeartbeat field 440
BATCHINT attribute 74
BatchInterval field 430
BatchSize field 419
BATCHSZ attribute 75

- benefits
 - intra-group queuing 300
 - shared queuing 283
- Benefits of shared queuing 283
- bind type 119
- BINDING channel state 56
- binding, fastpath 119
- browsing a channel 327

C

- caller
 - MCA 8
- caller MCA 8
- caller, responder 8
- calls
 - detailed description
 - MQ_CHANNEL_EXIT 406
 - MQXWAIT 412
- CapabilityFlags field 468
- change definition, channel 111, 333
- Change option 333
- channel
 - administration 53
 - alter
 - z/OS 247
 - attributes 111
 - auto-definition 54
 - auto-definition exit program 393
 - browsing 327
 - change definition 111, 333
 - channel control function
 - i5/OS 321
 - UNIX systems 105
 - Windows systems 105
 - characteristics
 - i5/OS 339
 - UNIX systems 116
 - Windows systems 116
 - client-connection 6
 - cluster-receiver 8
 - cluster-sender 8
 - command queue
 - z/OS 260
 - control commands 53
 - copy definition 333
 - create definition
 - i5/OS 332
 - UNIX Systems 111
 - Windows systems 111
 - creating 109, 324
 - creating your own defaults 332
 - default values supplied by WebSphere
 - MQ for i5/OS 332
 - define
 - z/OS 247
 - definition file
 - i5/OS 321
 - UNIX systems 106
 - Windows systems 106
 - definition, what is it? 51
 - delete 111, 333
 - z/OS 248
 - description 82
 - display
 - i5/OS 334
 - UNIX systems 111

- channel (*continued*)
 - display (*continued*)
 - Windows systems 111
 - display status
 - i5/OS 334
 - UNIX systems 111
 - Windows systems 111
 - display, z/OS 248
 - displaying 109, 334
 - displaying settings
 - i5/OS 334
 - UNIX Systems 112
 - Windows systems 112
 - displaying status 334
 - i5/OS 334
 - UNIX Systems 112
 - Windows systems 112
 - enabling 55
 - end 335
 - error 59
 - restarting after 64
 - fastpath binding 119
 - i5/OS
 - resolve 336
 - in doubt 65
 - in-doubt channels 65
 - initial data negotiation 54
 - initiator
 - AIX, HP-UX, Solaris, and Windows
 - systems 116
 - overview 9
 - starting 116
 - stopping 116
 - UNIX systems, and Windows
 - systems 116
 - z/OS 249
 - listener
 - overview 9
 - start, i5/OS 334
 - start, z/OS 251
 - stop, z/OS 252
 - STRMQMLSR command 334
 - trusted 11
 - monitoring 53
 - MQI 6
 - ping
 - i5/OS 334
 - UNIX systems 112
 - Windows systems 112
 - z/OS 254
 - preparing 54
 - program types
 - UNIX systems 116
 - WebSphere MQ for i5/OS 339
 - Windows systems 116
 - programs 339
 - AMQCCLA 339
 - AMQCRCTA 339
 - AMQCRS6A 116, 339
 - AMQCRSTA 116
 - AMQRMCLA 339
 - quiescing 62
 - receiver 6
 - receiving parameters 52
 - refuses to run 479
 - renaming
 - i5/OS 329

- channel (*continued*)
 - renaming (*continued*)
 - UNIX Systems 110
 - Windows systems 110
 - requester 6
 - requester-sender 7
 - requester-server 7
 - reset
 - z/OS 254
 - Reset
 - i5/OS 336
 - UNIX systems 114
 - Windows systems 114
 - resolving
 - i5/OS 336
 - UNIX Systems 114
 - Windows systems 114
 - z/OS 255
 - restart 55
 - restarting when stopped 64
 - run 110, 326
 - segregating messages 14
 - selecting 327
 - sender-receiver 6
 - sequence numbers 53
 - server-connection 6
 - server-receiver 8
 - sharing 13
 - start 55
 - i5/OS 326, 334
 - UNIX systems 112
 - UNIX Systems 110
 - Windows systems 110, 112
 - z/OS 252
 - startup negotiation errors 478
 - startup, data negotiation 54, 376, 378
 - state 55, 56
 - status 53
 - stopping 62, 335
 - i5/OS 335
 - UNIX systems 113
 - Windows systems 113
 - z/OS 255
 - switching 483
 - synchronizing 376
 - test, z/OS 254
 - triggering 18
 - UNIX systems 115
 - Windows systems 115
 - z/OS 260
 - trusted 119
 - types 77, 111, 116
 - using alternate channels 14
- CHANNEL attribute 76
- Channel attributes 71
 - by channel type 71
- channel auto-definition exit,
 - introduction 11
- channel configuration
 - WebSphere MQ for AIX 168
 - WebSphere MQ for HP-UX 190
 - WebSphere MQ for i5/OS 363
 - WebSphere MQ for Linux 234
 - WebSphere MQ for Solaris 211
 - WebSphere MQ for Windows 146
 - WebSphere MQ for z/OS 272, 292
- channel control error messages 477

- channel control function 53
 - i5/OS 321
 - UNIX systems 105
 - Windows systems 105
 - z/OS 243
- channel definition file
 - i5/OS 321
 - UNIX systems 106
 - Windows systems 106
- channel description 82
- channel disposition 83
- channel exit
 - compatible receive exit 389
 - considerations for pipelining 149
 - reserving space in send exit 389
- channel exits
 - auto-definition 393
 - message 390
 - message-retry 392
 - receive 387
 - security 378
 - CLNTCONN and SVRCONN channels 386
 - send 387
- channel functions
 - UNIX systems 111
 - Windows systems 111
- channel initiator
 - display, z/OS 249
 - overview 9
 - retries 59, 87
 - runmqchi command, WebSphere MQ for Windows 112
 - runmqchi command, WebSphere MQ on UNIX systems 112
 - running the MCA as a thread 90
 - start, i5/OS 335
 - start, Windows systems and UNIX systems 116
 - start, z/OS 249
 - stop, Windows systems and UNIX systems 116
 - stop, z/OS 250
 - STRMQMCHLI command 335
- channel listener
 - overview 9
 - start, i5/OS 335
 - start, z/OS 251
 - stop, z/OS 252
 - STRMQMLSR command 335
 - trusted 11
- channel name attribute 76
- channel planning example
 - i5/OS 368
 - UNIX systems 238
 - Windows 238
- channel planning examples
 - z/OS 275, 294
- channel programs
 - UNIX systems 116
 - WebSphere MQ for i5/OS 339
 - Windows systems 116
- channel refuses to run 479
- channel startup negotiation errors 478
- channel states
 - BINDING 55
 - i5/OS 340
- channel states (*continued*)
 - INACTIVE 59
- channel statistics attribute 76
- channel status
 - display, i5/OS 334
 - display, UNIX systems 111
 - display, Windows systems 111
 - display, z/OS 257
- channel type attribute 77
- channel-exit program 403
- channel-exit programs 375
 - channel definition structure, MQCD 394
 - data buffer 394
 - introduction 11
 - parameter structure, MQCXP 394
 - WebSphere MQ for AIX 399
 - WebSphere MQ for HP-UX 400
 - WebSphere MQ for i5/OS 397
 - WebSphere MQ for Solaris 401
 - WebSphere MQ for Windows 398
 - WebSphere MQ for z/OS 396
 - Windows 95 and Windows 98 client 398
 - Windows client 398
 - writing and compiling 394
- channel, performance improvement 149
- ChannelDefinition parameter
 - MQ_CHANNEL_AUTO_DEF_EXIT call 411
 - MQ_CHANNEL_EXIT call 407
- ChannelExitParms parameter
 - MQ_CHANNEL_AUTO_DEF_EXIT call 411
 - MQ_CHANNEL_EXIT call 407
- ChannelName field 414
- CHANNELS stanza 491
- channels, alternate to 14
- ChannelType field 416
- CHLTYPE attribute 77
- class
 - of service 280
- class of routing entry 350
- class of service 41
- Class of service 280
- client channel weight 78
- client channels
 - with queue sharing 283
- Client channels 283
- client-connection channel 6
 - security exit 386
- ClientChannelWeight 443
- clients, problem determination 483
- CLNTWGHT 78
- CLUSNL attribute 78
- CLUSTER attribute 78
- cluster channels, z/OS 258
- cluster components 4
- cluster name attribute 78
- cluster namelist attribute 78
- cluster-receiver 8
- cluster-receiver channel 6
- cluster-sender 8
- cluster-sender channel 6
- clustering
 - with intra-group queuing 306
- ClusterPtr field 435
- clusters
 - choosing transmission queue 33
 - components 4
 - concentrating messages 38
 - distribution lists 40
 - message flow 29
 - networking considerations 46
 - passing messages 35
 - putting messages 32
 - queue sharing with 284
 - reply-to queue 41
 - return routing 47
 - separating message flows 36
 - using 15
- Clusters and queue-sharing groups 284
- ClustersDefined field 435
- CLWLChannelPriority field 441
- CLWLChannelRank field 441
- CLWLChannelWeight field 442
- CLWLPRTY attribute 78
- CLWLRANK attribute 79
- CLWLWGHT attribute 79
- command queue channel, z/OS 260
- command validation 66
- commit in-doubt messages
 - i5/OS 336
 - UNIX systems 114
 - Windows systems 114
- committed messages
 - i5/OS 336
 - UNIX systems 114
 - Windows systems 114
- Communications Server for AIX V5 160
- Communications Server for Linux
 - establishing a session 220
 - explanation of terms 218
 - operation 232
- Communications Server for Windows NT 134
- communications side object
 - i5/OS 344, 345
 - z/OS 266, 285
- CompCode parameter 413
- COMPHDR attribute 84
- COMPMSG attribute 81
- components of
 - shared queuing 280
- components of distributed-queuing environment 12
 - channel initiator 9
 - channel listener 9
 - message channel 6
 - message channel agent 8
 - transmission queue 9
- Components of shared queuing 280
- components, cluster 4
- compression
 - data 81
 - header 84
- compression of data 387
- concentrating messages 38
- concentrators 27
- concepts
 - shared queuing 280
- concepts of
 - intra-group queuing 298

- concepts of intercommunication 1, 15, 20
- Concepts of queue-sharing groups 280
- configuration
 - WebSphere MQ for AIX 167
 - WebSphere MQ for HP-UX 190
 - WebSphere MQ for i5/OS 362
 - WebSphere MQ for Linux 234
 - WebSphere MQ for Solaris 211
 - WebSphere MQ for Windows 145
 - WebSphere MQ for z/OS 272, 291
- configuration file 68
 - UNIX systems 68
- configurations
 - intra-group queuing 303
- CONNNAME attribute 80
- connection
 - APPC/MVS, z/OS 263, 284
 - deciding upon
 - i5/OS 342
 - z/OS 263, 284
 - defining APPC/MVS (LU 6.2) 266, 285
 - defining LU 6.2
 - i5/OS 344
 - UNIX systems 154
 - Windows systems 122
 - LU 6.2
 - i5/OS 342
 - UNIX systems 150
 - Windows 120
 - z/OS 263, 284
 - NetBIOS
 - Windows 120
 - SPX
 - Windows 120
 - switching 483
 - TCP
 - i5/OS 342
 - UNIX systems 150
 - Windows 120
 - z/OS 263, 284
 - UDP
 - UNIX systems 150
- connection affinity 79
- connection name 80
- ConnectionAffinity 444
- ConnectionName field 426
- context security 94
- control commands, channel 53
- Conversations, sharing 404
- conversion failure, problem determination 480
- conversion of data 53
- CONVERT attribute 81
- convert message 81
- coordination with adjacent systems 38
- Copy option 333
- Create option 332
- creating
 - channel
 - i5/OS 324
 - UNIX systems 109
 - Windows systems 109
 - defaults 332
 - objects
 - i5/OS 324

- creating (*continued*)
 - objects (*continued*)
 - UNIX systems 108
 - Windows systems 108
 - queues 115, 337
 - transmission queue 115, 337
- CRTCSI command 345
- CRTMQM command 108
- CurHdrCompression field 470
- CurMsgCompression field 470
- current channels
 - specifying maximum number 58

D

- data
 - compression 387
 - conversion 390
 - decompression 387
 - encryption 390
 - negotiation 18, 54
- data compression 81
- data conversion 70
- data types, detailed description
 - MQCD 413
 - MQCXP 458
 - MQXWD 475
- DataConversion field 422
- DataLength parameter 407
- DDNS
 - registration time for 481
- dead-letter queue 46
 - overview 12
 - problem determination 477
 - processing 477
 - UNIX systems 117
 - WebSphere MQ for i5/OS 340
 - Windows systems 117
 - z/OS 117
- decompression of data 387
- default channel values
 - i5/OS 332
- default object creation 108
- define channel
 - z/OS 247
- defining
 - an LU 6.2 connection
 - i5/OS 344
 - UNIX systems 154
 - Windows systems 122
 - APPC/MVS (LU 6.2) connection
 - z/OS 266, 285
 - objects
 - z/OS 260
 - queues
 - z/OS 260
 - z/OS 247
- definition file
 - i5/OS 321
 - UNIX systems 106
 - Windows systems 106
- delete channel
 - distributed platforms 111
 - i5/OS 333
 - z/OS 248
- delivery, messages 20
- Desc field 417

- DESCR attribute 82
- description, channel 82
- destination queue 36
- dial-up support 481
- disabled receiver channels 112, 334
- disaster recovery 482
- DISCINT attribute 82
- DiscInterval field 419
- disconnect interval 82
- display
 - option 334
 - z/OS, DQM 249
- display channel
 - i5/OS 334
 - UNIX systems 109
 - Windows systems 109
 - z/OS 248
- display channel initiator
 - z/OS 249
- display channel status
 - z/OS 257
- Display channel status
 - UNIX systems 110
 - Windows systems 110
- display DQM 249
- disposition 83
- distributed queuing
 - components 5, 12
 - functions 50
 - with intra-group queuing 304
- distributed queuing with queue-sharing groups 280
- distributed-queuing management in WebSphere MQ for i5/OS 336
- distribution lists 40, 53
- diverting message flows 39
- DQM
 - display, z/OS 249

E

- ECB field 476
- edit
 - change
 - i5/OS 333
 - UNIX systems 111
 - Windows systems 111
 - copy
 - i5/OS 333
 - create
 - i5/OS 332
 - UNIX systems 111
 - Windows systems 111
 - delete
 - i5/OS 333
 - UNIX systems 111
 - Windows systems 111
- enabling a channel to transmit messages 55
- encryption
 - in send exit 389
- encryption of messages 375
- end 113
- End option 335
- ending a channel 113, 335
- ENDMQLSR command 117

- error
 - at remote sites 477
 - channel 59
 - logs 113, 484
 - message from channel control 477
 - recovery 476
 - example
 - alias walk-through 45
 - channel initiators 9
 - channel listeners 9
 - channel names 26
 - channel planning
 - for distributed platforms 238
 - for i5/OS 368
 - for z/OS 275, 294
 - i5/OS 368
 - UNIX systems 238
 - Windows 238
 - z/OS 275, 294
 - choosing the transmission queue 33
 - cluster of queue managers 4
 - communication in WebSphere MQ for i5/OS, TCP connection 342
 - concentrating messages 38
 - configurations 101, 102
 - create channel 109
 - creating reply-to aliases 30
 - defining channels 16
 - defining queues 17
 - defining remote queue definitions 30
 - display channel 109
 - display channel status 110
 - diverting message flows 39
 - establishing a session using Communications Server for Linux 220
 - establishing a session using SNAP-IX 199
 - flow control 29
 - intra-group queuing 313
 - local queue definition
 - i5/OS 371
 - UNIX systems 241
 - Windows 241
 - z/OS 278
 - message channels
 - cluster-receiver 8
 - cluster-sender 8
 - requester-sender 7
 - requester-server 7
 - sender-receiver 6
 - multi-hopping 13
 - passing messages through system 35
 - passing through intermediate queue managers 13
 - putting messages on remote queues 32
 - QM-concentrators 27
 - queue name resolution 48
 - receiver channel definition
 - i5/OS 371, 372
 - UNIX systems 240, 241
 - Windows 240, 241
 - z/OS 278, 279
 - receiving messages 34
 - remote queue definition
 - i5/OS 370
 - example (continued)
 - remote queue definition (continued)
 - UNIX systems 240
 - Windows 240
 - z/OS 277
 - renaming a channel 110
 - reply-to queue 41, 42
 - reply-to queue definition
 - i5/OS 371
 - UNIX systems 240
 - Windows 240
 - z/OS 278
 - reply-to-queue alias 24
 - running
 - i5/OS 372
 - UNIX systems 241
 - Windows 241
 - z/OS 279
 - send exits reserving space 390
 - sender channel definition
 - i5/OS 370, 372
 - UNIX systems 240, 241
 - Windows 240, 241
 - z/OS 278, 279
 - sending messages 3, 15
 - sending messages in both directions 3
 - separating message flows 36
 - setting up communication for Windows systems
 - defining a NetBIOS connection 124
 - defining a TCP connection 121
 - defining an LU 6.2 connection 122
 - defining an SPX connection 126
 - setting up communication in UNIX systems
 - defining a TCP connection 150
 - defining an LU 6.2 connection 154
 - setting up communication in WebSphere MQ for z/OS
 - LU 6.2 connection 266, 285
 - TCP connection 263
 - sharing a transmission queue 13
 - starting a channel 110, 326
 - transmission queue definition
 - i5/OS 370, 372
 - UNIX systems 240, 241
 - Windows 240, 241
 - z/OS 277, 279
 - triggering 19, 116
 - using multiple channels 14
 - using the remote queue definition object 31
 - WebSphere MQ for AIX
 - configuration 155
 - WebSphere MQ for HP-UX
 - configuration 172
 - WebSphere MQ for i5/OS
 - configuration 350
 - WebSphere MQ for Linux
 - configuration 215
 - WebSphere MQ for Solaris
 - configuration 194
 - example (continued)
 - WebSphere MQ for Windows
 - configuration 129
 - WebSphere MQ for z/OS
 - configuration 266, 286
 - example configurations
 - WebSphere MQ for AIX 155
 - WebSphere MQ for HP-UX 172
 - WebSphere MQ for i5/OS 350, 368
 - WebSphere MQ for Linux 215
 - WebSphere MQ for Solaris 194
 - WebSphere MQ for Windows 129
 - WebSphere MQ for z/OS 266, 286
 - exit wait descriptor structure 475
 - ExitBufferAddr parameter 408
 - ExitBufferLength parameter 408
 - ExitData field 466
 - ExitDataLength field 432
 - ExitId field 460
 - ExitNameLength field 432
 - ExitNumber field 468
 - EXITPATH
 - stanza of qm.ini file 491
 - ExitReason field
 - MQCXP structure 460
 - ExitResponse field
 - MQCXP structure 462
 - ExitResponse2 field 464
 - ExitSpace field 468
 - ExitUserArea field
 - MQCXP structure 466
- F**
 - FAPLevel field 468
 - fast, nonpersistent messages 20
 - sequence of retrieval 49
 - specifying 93
 - Feedback field
 - MQCXP structure 465
 - fields
 - BatchHeartbeat 440
 - BatchInterval 430
 - BatchSize 419
 - CapabilityFlags 468
 - ChannelName 414
 - ChannelType 416
 - ClusterPtr 435
 - ClustersDefined 435
 - CLWLChannelPriority 441
 - CLWLChannelRank 441
 - CLWLChannelWeight 442
 - ConnectionName 426
 - CurHdrCompression 470
 - CurMsgCompression 470
 - DataConversion 422
 - Desc 417
 - DiscInterval 419
 - ECB 476
 - ExitData 466
 - ExitDataLength 432
 - ExitId 460
 - ExitNameLength 432
 - ExitNumber 468
 - ExitReason
 - MQCXP structure 460

fields (*continued*)

- ExitResponse
 - MQCXP structure 462
- ExitResponse2 464
- ExitSpace 468
- ExitUserArea
 - MQCXP structure 466
- FAPLevel 468
- Feedback
 - MQCXP structure 465
- Hconn 471
- HdrCompList 440
- HeaderLength 467
- HeartbeatInterval 429
- KeepAliveInterval 439
- LocalAddress 439
- LongMCAUserIdLength 436
- LongMCAUserIdPtr 436
- LongRemoteUserIdLength 436
- LongRemoteUserIdPtr 437
- LongRetryCount 420
- LongRetryInterval 420
- MaxInstances 445
- MaxInstancesPerClient 445
- MaxMsgLength 422
- MaxSegmentLength 465
- MCAName 418
- MCASecurityId 437
- MCAType 425
- MCAUserIdentifier 424
- ModeName 418
- MsgCompList 441
- MsgExit 420
- MsgExitPtr 433
- MsgExitsDefined 432
- MsgRetryCount
 - MQCD structure 428
 - MQCXP structure 466
- MsgRetryExit 427
- MsgRetryInterval
 - MQCD structure 429
 - MQCXP structure 467
- MsgRetryReason 467
- MsgRetryUserData 428
- MsgUserData 423
- MsgUserDataPtr 433
- NetworkPriority 436
- NonPersistentMsgSpeed 430
- PartnerName 468
- Password 424
- PropertyControl 444
- PutAuthority 422
- QMgrName 417
- ReceiveExit 421
- ReceiveExitPtr 434
- ReceiveExitsDefined 432
- ReceiveUserData 423
- ReceiveUserDataPtr 435
- RemotePassword 427
- RemoteSecurityId 437
- RemoteUserIdentifier 426
- Reserved1 475
- Reserved2 475
- Reserved3 476
- SecurityExit 420
- SecurityParms 470
- SecurityUserData 422

fields (*continued*)

- SendExit 421
- SendExitPtr 433
- SendExitsDefined 432
- SendUserData 423
- SendUserDataPtr 434
- SeqNumberWrap 421
- SharingConversations 443, 471
- ShortConnectionName 418
- ShortRetryCount 419
- ShortRetryInterval 419
- SSLCertUserId 469
- SSLCipherSpec 438
- SSLClientAuth 438
- SSLPeerNameLength 438
- SSLPeerNamePtr 438
- SSLRemCertIssNameLength 469
- SSLRemCertIssNamePtr 469
- StrucId
 - MQCXP structure 459
 - MQXWD structure 475
- StrucLength 431
- TpName 418
- TransportType
 - MQCD structure 416
- UserIdentifier 424
- Version
 - MQCD structure 415
 - MQCXP structure 459
 - MQXWD structure 475
- XmitQName 417

flow control 29

for shared queuing

- listeners 280

functions available

- UNIX systems 106
- Windows systems 106

G

- generic
 - interface 280
- Generic interface 280
- getting started
 - intra-group queuing 302
- Getting started with intra-group queuing 302

H

- HBINT attribute 84
- Hconn field 471
- Hconn parameter 412
- HdrCompList field 440
- header compression 84
- HeaderLength field 467
- heartbeat interval 84
- HeartbeatInterval field 429

I

- i5/OS
 - KEEPALIVE 343
- IBM Communications Server for Windows NT 134
- in-doubt 75

- in-doubt channels, manual resynchronization 65
- in-doubt message on channel, resolve on z/OS 255
- in-doubt messages, commit or back out
 - i5/OS 336
 - UNIX systems 114
 - Windows systems 114
- INACTIVE channel state 56, 59
- inbound channels
 - with shared queuing 282
- Inbound channels with shared queuing 282
- INETD 232
- ini file 68
- initial data negotiation 18, 54
- initialization data set, z/OS 68
- initialization file 68
- initiator for channel
 - AIX, HP-UX, Solaris, and Windows systems 116
 - UNIX systems and Windows systems 116
 - z/OS 249
- integrity of delivery 20
- intercommunication
 - concepts 1, 15, 20
 - example configuration 101
- intercommunication examples
 - WebSphere MQ for AIX 155
 - WebSphere MQ for HP-UX 172
 - WebSphere MQ for i5/OS 350
 - WebSphere MQ for Linux 215
 - WebSphere MQ for Solaris 194
 - WebSphere MQ for Windows 129
 - WebSphere MQ for z/OS 266, 286
- interface
 - generic 280
- intra-group queuing
 - agent 300
 - benefits 300
 - concepts of 298
 - configurations 303
 - example 313
 - getting started 302
 - intra-group queuing agent 299
 - limitations 302
 - messages 308
 - security 310
 - shared transmission queue 300
 - specific properties 311
 - terminology 300
 - with clustering 306
 - with distributed queuing 304
- intra-group queuing agent
 - intra-group queuing 299
- Intra-group queuing agent 300
- Intra-group queuing and the intra-group queuing agent 299
- Intra-group queuing benefits 300
- Intra-group queuing concepts 298
- Intra-group queuing configurations 303
- intra-group queuing example 313
- Intra-group queuing limitations 302
- Intra-group queuing security 310
- Intra-group queuing specific properties 311

Intra-group queuing terminology 300
Intra-group queuing with clustering 306
Intra-group queuing with distributed queuing 304

J

journaling 390

K

KAINT attribute 84
KEEPALIVE 61
 i5/OS 343
 UNIX systems 154
KeepAlive interval 84
KeepAliveInterval field 439

L

limitations
 intra-group queuing 302
links, wide-band 27
list cluster channels, z/OS 258
listener, trusted 9, 11, 119
listeners
 for shared queuing 280
Listeners 280
listening on LU 6.2
 UNIX systems 155
 Windows systems 123
 z/OS 266, 285
listening on NetBIOS
 Windows systems 126
listening on SPX
 Windows 144
 Windows systems 127
listening on TCP
 i5/OS 343
 UNIX systems 151
 Windows systems 121
 z/OS 264, 285
Local Address 85
local queue definition
 example
 i5/OS 371
 UNIX systems 241
 Windows 241
 z/OS 278
local queue manager 1
LOCALADDR attribute 85
LocalAddress field 439
location name 35
log
 error 113, 484
 file, @SYSTEM 484
logs for errors 113
long retry count attribute 86
long retry interval attribute 87
LongMCAUserIdLength field 436
LongMCAUserIdPtr field 436
LongRemoteUserIdLength field 436
LongRemoteUserIdPtr field 437
LongRetryCount field 420
LongRetryInterval field 420
LONGRTY attribute 86

LONGTMR attribute 87
loopback testing 49
LU 6.2
 mode name 87
 settings
 i5/OS 344
 UNIX systems 154
 Windows systems 122
 TP name 88
LU 6.2 connection
 setting up
 i5/OS 342
 UNIX systems 150
 Windows 120
 z/OS 266, 285
 WebSphere MQ for AIX 156
 WebSphere MQ for HP-UX 172
 WebSphere MQ for i5/OS 351
 WebSphere MQ for Linux (x86 platform) 215
 WebSphere MQ for Solaris 195
 WebSphere MQ for Windows 129
 WebSphere MQ for z/OS 267, 286
 worksheet
 WebSphere MQ for Linux configuration 215
LU62
 stanza of qm.ini file 491

M

maximum
 active channels 58
 current channels 58
 message length 89
 server-connection channels 60
maximum instances 88
maximum instances per client 89
MAXINST attribute 88
MaxInstances field 445
MaxInstancesPerClient field 445
MAXINSTC attribute 89
MAXMSGSL attribute 89
MaxMsgLength field 422
MaxSegmentLength field 465
MCA
 adopting 62, 481
 caller 8
 name 89
 responder 8
 type 90
 user 90
 user-written 70
MCANAME attribute 89
MCANAME field 418
MCASecurityId field 437
MCATYPE attribute 90
MCAType field 425
MCAUSER attribute 90
MCAUserIdentifier field 424
message
 committed
 i5/OS 336
 UNIX systems 114
 Windows systems 114
 concentrating 38
 converting 81
message (*continued*)
 diverting flows 39
 encryption 375
 error 477
 for distribution list 40
 passing through system 35
 putting on remote queue 31
 queue name translations 47
 receiving 34
 return routing 47
 return to sender 67
 routing 33
 sending and receiving 51
 separating flows 36
 sequence numbering 49
 sequential retrieval 49
 splitting 53
 undeliverable 66
message channel
 cluster-receiver 6, 8
 cluster-sender 6, 8
 receiver 6
 requester 6
 requester-sender 7
 requester-server 7
 sender 6
 sender-receiver 6
 server 6
 server-receiver 8
message channel agent
 caller 8
 initiation 378, 387
 responder 8
 security 94
 termination 378, 387
 user-written 70
message channel agent (MCA) 8, 51
message channel agents
 with shared queuing 281
Message channel agents with shared queuing 281
message exit 11
message exit name 91
message exit program 390
 overview 376
message exit user data 91
message flow control 29
 networking considerations 46
message monitoring 485
message retry 67
message-retry exit
 introduction 11
 name 91
 retry count 91
 retry interval 92
 user data 91
message-retry exit program 392
messages
 assured delivery 20
 back out in-doubt messages
 i5/OS 336
 commit in-doubt messages
 i5/OS 336
 intra-group queuing 308
 resolve in-doubt messages
 i5/OS 336
 sending 15

- Messages
 - back out in-doubt messages 114
 - commit in-doubt messages 114
 - resolve in-doubt messages 114
- messages and codes 66
- Messages put to
 - SYSTEM.QSG.TRANSMIT
 - .QUEUE 308
- mode name 87
- MODENAME attribute 87
- ModeName field 418
- MONCHL 92
- monitoring 92
 - message 485
- monitoring and controlling channels
 - i5/OS 321
 - UNIX systems 105
 - Windows 105
 - z/OS 243
- monitoring channels 53
- moving service component 2
- MQ_CHANNEL_AUTO_DEF_EXIT
 - call 410
- MQ_CHANNEL_EXIT call 406
- MQCD structure 413
- MQCD, channel definition structure 394
- MQCXP structure 458
 - ExitReason field 389
 - ExitSpace field 389
- MQCXP_* values 459
- MQCXP, channel exit parameter
 - structure 394
- MQFB_* values 465
- MQI channels 6
- MQIBindType 119
- MQRMH, reference-message header 390
- mqs.ini 69
- MQSINI 69
- MQXCC_* values
 - MQCXP structure 462
- MQXCP_VERSION_5, of MQCXP
 - structure 389
- MQXQH, transmission header 390, 392
- MQXR_* values
 - MQCXP structure 460
- MQXR_INIT, ExitReason value 389
- MQXR_XMIT, ExitReason value 389
- MQXR2_* values 464
- MQXT_* values 460
- MQXUA_* values
 - MQTXP structure 466
- MQXWAIT call 412
- MQXWD structure 475
- MQXWD_* values 475
- MRDATA attribute 91
- MREXIT attribute 91
- MRRTY attribute 91
- MRTMR attribute 92
- MsgCompList field 441
- MSGDATA attribute 91
- MSGEXIT attribute 91
- MsgExit field 420
- MsgExitPtr field 433
- MsgExitsDefined field 432
- MsgRetryCount field
 - MQCD structure 428
 - MQCXP structure 466

- MsgRetryExit field 427
- MsgRetryInterval field
 - MQCD structure 429
 - MQCXP structure 467
- MsgRetryReason field 467
- MsgRetryUserData field 428
- MsgUserData field 423
- MsgUserDataPtr field 433
- multi-hopping 13
- multiple message channels per
 - transmission queue
 - UNIX systems 55
 - Windows systems 55
- multiple queue managers 123

N

- name resolution
 - conflicts 47
 - convention 46
 - description 487
 - introduction 22
 - location 35
 - queue name translations 47
 - restriction 42
- negotiations on startup 54, 478
- NetBIOS 2, 124
- NETBIOS
 - stanza of qm.ini file 491
- NetBIOS connection
 - WebSphere MQ for Windows 142
 - Windows 120
- NetBIOS products, in example
 - configurations 102
- NetBIOS, example configurations 102
- network infrastructure, example
 - configurations 102
- network planner 27
- network-connection priority 93
- networking 35
- networking considerations 46
- NetworkPriority field 436
- networks 25
- node centric 30
- nonpersistent message speed 93
- NonPersistentMsgSpeed field 430

O

- objects
 - creating
 - default 108
 - i5/OS 324
 - UNIX systems 108
 - Windows systems 108
 - defining
 - z/OS 260
 - security 117, 341
- operator commands
 - i5/OS 321
- options
 - change 333
 - copy 333
 - create 332
 - display 334
 - display status 334

- options (*continued*)
 - end 335
 - ping 334
 - reset 336
 - resolve 114
 - i5/OS 336
 - start 334
- outbound channels
 - with shared queuing 282
- Outbound channels with shared
 - queuing 282

P

- panel data, validation 66
- panels
 - browsing a channel
 - i5/OS 327
 - channel start
 - i5/OS 334
 - creating a channel
 - i5/OS 324
 - display
 - i5/OS 334
 - display channel status 334
 - ending channel
 - i5/OS 335
 - i5/OS
 - resolve 336
 - work with status 334
 - ping
 - i5/OS 334
 - reset
 - i5/OS 336
 - selecting a channel
 - i5/OS 327
 - using, z/OS 244
 - Work with channel status
 - i5/OS 329
 - work-with-channel choices
 - i5/OS 331
- parameters
 - AgentBuffer 408
 - AgentBufferLength 407
 - ChannelDefinition
 - MQ_CHANNEL_EXIT call 407
 - ChannelExitParms
 - MQ_CHANNEL_EXIT call 407
 - CompCode 413
 - DataLength 407
 - ExitBufferAddr 408
 - ExitBufferLength 408
 - Hconn 412
 - Reason 413
 - security exit 384
 - WaitDesc 412
- parameters, receiving 52
- PartnerName field 468
- Password attribute
 - encrypted value 100
 - introduction 93
- Password field 424
- PAUSED channel state 56, 59
- peer recovery
 - with queue-sharing 283
- Peer recovery 283

- peer-shared-channel retry on z/OS
 - about 482
- performance
 - channel 149
- ping 334
- problem determination 477
 - UNIX systems 112
 - Windows systems 112
- ping channel
 - z/OS 254
- ping with LU 6.2 112, 334
- pipelining
 - in MCA message transfer 149
 - parameter in qm.ini file 149
- port 200
 - in qm.ini file 491
 - WebSphere MQ for AIX 161
 - WebSphere MQ for HP-UX 178
 - WebSphere MQ for i5/OS 342
 - WebSphere MQ for Linux (x86 platform) 221
 - WebSphere MQ for Windows systems 121
 - WebSphere MQ for z/OS 249, 272, 291
- preparation
 - getting started
 - i5/OS 323
 - UNIX systems 107
 - Windows systems 107
- preparing channels 54
- preparing WebSphere MQ for i5/OS 336
- priority 78
- problem determination 476
 - channel refuses to run 479
 - channel startup negotiation errors 478
 - channel switching 483
 - clients 483
 - connection switching 483
 - conversion failure 480
 - data structures 482
 - dead-letter queue 477
 - error messages 477
 - retrying the link 481
 - scenarios 476
 - transmission queue overflow 477
 - triggered channels 480
 - undelivered-message queue 477
 - user-exit programs 482
 - using the PING command 477
 - validation checks 478
- process definition for triggering
 - i5/OS 339
 - UNIX systems 115
 - Windows systems 115
 - z/OS 260
- process security 94
- processing problems 66
- programs
 - AMQCRCTA 340
 - AMQCRS6A 117
 - AMQCRSTA 117, 340
 - AMQRMCLA 340
 - RUNMQCHI 117
 - RUNMQCHL 117
 - RUNMQLSR 117

- PropertyControl field 444
- PUT authority 94
- PUTAUT attribute 94
- PutAuthority field 422
- putting messages 31
 - on remote queues 31
 - to distribution lists 40

Q

- qm.ini 69
 - Channels stanza 149
 - stanzas used for distributed queuing 491
- QMGrName field 417
- QMIMI file
 - stanzas used for distributed queuing 491
- QMNAME attribute 95
- QSGDISP attribute 83
- queue
 - destination 36
 - reply-to 41
- queue manager
 - alias 21, 30
 - receiving 34
 - name 95
 - alias 36
 - types 1
- queue manager alias 21, 30
 - introduction 22
 - receiving 34
- queue name
 - resolution 487
 - how it works 489
 - what is it? 488
 - translations 47
- queue sharing
 - peer recovery with 283
- queue-sharing
 - clusters and 284
- queues
 - create a transmission queue 115, 337
 - defining
 - z/OS 260
- queuing
 - shared 280
- quiescing channels 62

R

- rank 79
- RCVDATA attribute 96
- RCVEXIT attribute 95
- Reason parameter 413
- receive exit 11
 - name 95
 - program 387
 - user data 96
- ReceiveExit field 421
- ReceiveExitPtr field 434
- ReceiveExitsDefined field 432
- receiver
 - channel 6
 - channel definition example
 - i5/OS 371

- receiver (*continued*)
 - channel definition example (*continued*)
 - UNIX systems 240
 - Windows 240
 - z/OS 278
- receiver channel definition example
 - i5/OS 372
 - UNIX systems 241
 - Windows 241
 - z/OS 279
 - overview 3
- ReceiveUserData field 423
- ReceiveUserDataPtr field 435
- receiving
 - messages 34, 51
 - on LU 6.2
 - UNIX systems 155
 - Windows systems 123
 - z/OS 266, 285
 - on SPX
 - Windows 144
 - Windows systems 127
 - on TCP
 - i5/OS 343
 - UNIX systems 151
 - Windows systems 121
 - z/OS 264, 285
- receiving messages 34, 51
- receiving on LU 6.2
 - UNIX systems 155
 - Windows systems 123
 - z/OS 266, 285
- receiving on SPX
 - Windows 144
 - Windows systems 127
- receiving on TCP
 - i5/OS 343
 - UNIX systems 151
 - Windows systems 121
 - z/OS 264, 285
- recovery with queue-sharing
 - peer 283
- reference-message header
 - message exit program 390
- Registration time for DDNS 481
- registry 68, 69, 122, 124, 129, 144, 398
- remote queue definition 30
 - example
 - i5/OS 370
 - UNIX systems 240
 - Windows 240
 - z/OS 277
 - introduction 14, 21
 - what it is 12
- remote queue manager 1
- RemotePassword field 427
- RemoteSecurityId field 437
- RemoteUserIdentifier field 426
- renaming a channel
 - i5/OS 329
 - UNIX systems 110
 - Windows systems 110
- reply-to alias 30
- reply-to queue 41
 - alias definition 24
 - alias example 42

- reply-to queue (*continued*)
 - aliases 21
 - preparing for 25
 - specifying 24
- reply-to queue definition
 - example
 - i5/OS 371
 - UNIX systems 240
 - Windows 240
 - z/OS 278
- requester channel 6
- REQUESTING channel state 56
- Reserved1 field 475
- Reserved2 field 475
- Reserved3 field 476
- reset 114, 336
- RESET CHANNEL command 479
- reset channel sequence numbers,
 - z/OS 254
- RESOLVE CHANNEL command 479
- resolve in-doubt message on channel,
 - z/OS 255
- resolve in-doubt messages 114
 - i5/OS 336
- resolve option 114
 - i5/OS 336
- responder
 - LU6.2 112
 - MCA 8
- responder MCA 8
- responder process 112
- restarting
 - channels 55
 - restarting stopped channels 64
- RETRY channel state 56, 59
- retry considerations 481
- retrying the link, problem
 - determination 481
- return routing 47
- return to sender 67
- route tracing 485
- routing entry
 - add 349
 - class 350
- routing entry class 350
- routing messages 33
- run channel 110, 326
- run channel initiator 116
- runmqchi command
 - AIX, HP-UX, Solaris, and Windows
 - systems 116
 - UNIX systems and Windows
 - systems 116
- RUNMQCHI command 117
- RUNMQCHL command 117
- RUNMQLSR command 117

S

- sample security exit 384, 403
- scenarios, problem determination 476
- SCYDATA attribute 96
- SCYEXIT attribute 96
- security
 - context 94
 - exit 11
 - exit name 96

- security (*continued*)
 - exit program 378
 - CLNTCONN and SVRCONN
 - channels 386
 - exit program, overview 376
 - exit user data 96
 - intra-group queuing 310
 - levels for exit programs 119
 - message channel agent 94
 - objects
 - UNIX systems 117
 - WebSphere MQ for i5/OS 341
 - Windows systems 117
 - process 94
 - security exit
 - parameters 384
 - sample 384, 403
 - Security Services Programming Interface (SSPI) 403
 - SecurityExit field 420
 - SecurityParms field 470
 - SecurityUserData field 422
 - segregating messages 14
 - selecting a channel 327
 - send
 - exit 11
 - exit name 97
 - exit program 387
 - message 51
 - send exit
 - multiple send exits 390
 - send exit user data 97
 - SENDDATA attribute 97
 - sender channel 6
 - sender channel definition
 - example
 - i5/OS 370, 372
 - UNIX systems 240, 241
 - Windows 240, 241
 - z/OS 278, 279
 - overview 3
 - SENDEXIT attribute 97
 - SendExit field 421
 - SendExitPtr field 433
 - SendExitsDefined field 432
 - sending
 - messages 15, 51
 - on SPX
 - Windows 127
 - on TCP
 - Windows 121
 - sending on TCP 151
 - SendUserData field 423
 - SendUserDataPtr field 434
 - SeqNumberWrap field 421
 - sequence number wrap 97
 - sequence numbering 49
 - sequence numbers 53
 - reset, z/OS 254
 - sequential retrieval of messages 49
 - SEQWRAP attribute 97
 - server
 - channel 6
 - server-connection channel 6
 - security exit 386
 - server-connection channels, maximum
 - number 60

- service
 - class of 280
- service, class of 41
- setting up
 - communication
 - i5/OS 342
 - UNIX systems 150
 - Windows 120
- shared
 - queuing 280
- shared queuing
 - benefits of 283
 - components of 280
 - concepts of 280
 - inbound channels 282
 - message channel agents with 281
 - outbound channels 282
 - synchronization queue 282
 - transmission 281
 - triggering with 281
- shared transmission queue
 - intra-group queuing 300
- Shared transmission queue for
 - intra-group queuing 300
- SHAREPORT 291
- sharing channels 13
- Sharing conversations 404
- SharingConversations
 - MQCD 404
 - MQCXP 404
- SharingConversations field 443, 471
- short retry
 - count 97
 - interval 98
- ShortConnectionName field 418
- ShortRetryCount field 419
- ShortRetryInterval field 419
- SHORTRTY attribute 97
- SHORTTMR attribute 98
- side object
 - i5/OS 345
- SNA 2
 - products, in example
 - configurations 102
- SNAP-IX
 - configuration parameters 195
 - establishing a session 199
 - explanation of terms 197
 - operation 210
 - sender-channel definitions 214
- SNAplus2 176
- SO_KEEPALIVE
 - i5/OS 343
 - UNIX systems 154
 - Windows systems 122
- source queue manager 1
- specific properties
 - intra-group queuing 311
- splitting messages 53
- SPX 2
 - connection
 - WebSphere MQ for Windows 143
 - example configurations 102
 - products, in example
 - configurations 102
 - stanza of qm.ini file 491
 - Windows 120

- SSLCAUTH attribute 99
- SSLCertUserId field 469
- SSLCIPH attribute 98
- SSLCipherSpec field 438
- SSLClientAuth field 438
- SSLPEER attribute 99
- SSLPeerNameLength field 438
- SSLPeerNamePtr field 438
- SSLRemCertIssNameLength field 469
- SSLRemCertIssNamePtr field 469
- SSPI (Security Services Programming Interface) 403
- stanza file 68
- stanza, in qm.ini file
 - Channels 149
- start
 - channel 55
 - UNIX systems 110
 - Windows systems 110
 - z/OS 252
 - channel initiator, z/OS 249
 - channel listener, z/OS 251
 - option 334
- STARTING channel state 56
- startup dialog 376
- STATCHL attribute 76
- state, channel 55
- status
 - display channel 110
 - work with channel 329
- status panels 334
- status, channel 53
- stop
 - channel 62, 113
 - channel initiator, z/OS 250
 - channel listener, OS/390 252
 - channel, z/OS 255
 - controlled 336
 - immediate 336
 - quiesce 113, 114
- stop channel initiator 116
- stop force 113, 114
- STOPPED channel state 56, 59
- stopped channels, restarting 64
- STOPPING channel state 56
- STRMQM command 108
- StrucId field
 - MQCXP structure 459
 - MQXWD structure 475
- StrucLength field 431
- structure
 - MQCXP 389
- structures
 - MQCD 413
 - MQCXP 458
 - MQXWD 475
- switching channels 483
- synchronization
 - with shared queuing 282
- Synchronization queue with shared queuing 282
- synchronization queue, z/OS 260
- synchronizing channels 376
- syncpoint introduction 75
- Sysplex Distributor 291
- system extension 119, 341

- system extensions
 - user-exit programs
 - UNIX systems 119
 - WebSphere MQ for i5/OS 341
 - Windows systems 119
- SYSTEM.CHANNEL.INITQ queue
 - i5/OS 368
 - UNIX systems 238
 - Windows 238
 - z/OS 243, 260
- SYSTEM.CHANNEL.REPLY.INFO queue 243, 260
- SYSTEM.CHANNEL.SYNCQ 260

T

- target queue manager 1
- TCP
 - connection
 - listener backlog 127, 152, 265, 344
 - WebSphere MQ for AIX 166
 - WebSphere MQ for HP-UX 189
 - WebSphere MQ for i5/OS 360
 - WebSphere MQ for Linux 232
 - WebSphere MQ for Solaris 210
 - WebSphere MQ for z/OS 272, 290
 - Windows 142
 - example configurations 102
 - listener backlog option 127, 152, 265, 344
 - OpenEdition MVS sockets 262
 - products, in example configurations 102
 - stanza of qm.ini file 491
 - stanza of QMINI file 491
- TCP connection
 - setting up
 - UNIX systems 150
 - Windows 120
 - z/OS 263
- TCP KEEPALIVE
 - i5/OS 343
 - UNIX systems 154
 - Windows systems 122
- TCP/IP 2
- TCP/IP KEEPALIVE 61
- terminology
 - intra-group queuing 300
- test channel, z/OS 254
- testing connections, lookback testing 49
- threads
 - multiple 149
- time-out 82
- TPNAME and TPPATH
 - i5/OS 344
 - UNIX systems 154
 - Windows systems 122
- TPNAME attribute 88
- TpName field 418
- transaction
 - program name 88
- transmission
 - shared queuing 281
- transmission header
 - alias
 - definition 22
 - message exit program 390

- transmission header (*continued*)
 - message-retry exit program 392
 - queue name 30
- transmission of messages
 - maximum transmission size 389
 - transmission buffer 389
- transmission protocol 100
- transmission queue
 - definition of 9
 - example definition
 - i5/OS 370
 - UNIX systems 240
 - Windows 240
 - z/OS 277
 - multiple message channels
 - UNIX systems 55
 - Windows systems 55
 - overflow 477
 - selecting 36
 - sharing 13
- Transmission queue 281
- transmission queue definition
 - example
 - i5/OS 372
 - UNIX systems 241
 - Windows 241
 - z/OS 279
- transmission queue name 100
- transport type 100
 - supported 2
- transport-retry exit
 - introduction 11
- TransportType field
 - MQCD structure 416
- triggered channels, problem determination 480
- triggering
 - channels 18
 - UNIX systems 115
 - WebSphere MQ for i5/OS 339
 - Windows systems 115
 - z/OS 260
 - with shared queuing 281
- Triggering with shared queuing 281
- TRPTYPE attribute 100
- trusted applications 11, 119
- type, bind 119
- types of channel 77

U

- UCD products, in example configurations 102
- UDP
 - example configurations 102
- undeliverable message 66
- undelivered-message queue
 - UNIX systems 117
 - Windows systems 117
 - z/OS 117
- UNIX
 - KEEPALIVE 154
 - user ID 100, 119
- user-exit
 - programs 482
- user-exit programs 375
 - problem determination 482

- user-exit programs *(continued)*
 - security levels 119
 - system extension
 - i5/OS 341
 - UNIX systems 119
 - Windows systems 119
 - writing and compiling 394
- user-written MCAs 70
- USERDATA parameter
 - z/OS 260
- USERID attribute 100
- UserIdentifier field 424

V

- validation
 - checks 478
 - command 66
 - of user IDs 390
 - panel data 66
- values supplied by WebSphere MQ for i5/OS 332
- Version field
 - MQQCD structure 415
 - MQCXP structure 459
 - MQXWD structure 475

W

- WaitDesc parameter 412
- WAITING channel state 56
- WebSphere MQ for AIX
 - channel configuration 168
 - channel-exit programs 399
 - configuration 167
 - intercommunication example 155
 - LU 6.2 connection 156
 - TCP connection 166
- WebSphere MQ for HP-UX
 - channel configuration 190
 - channel-exit programs 400
 - configuration 190
 - intercommunication example 172
 - LU 6.2 connection 172
 - TCP connection 189
- WebSphere MQ for i5/OS
 - channel configuration 363
 - channel-exit programs 397
 - configuration 362
 - intercommunication example 350, 368
 - LU 6.2 connection 351
 - TCP connection 360
- WebSphere MQ for Linux
 - channel configuration 234
 - configuration 234
 - intercommunication example 215
 - TCP connection 232
 - using INETD 232
 - using XINETD 233
- WebSphere MQ for Solaris
 - channel configuration 211
 - channel-exit programs 401
 - configuration 211
 - intercommunication example 194
 - TCP connection 210

- WebSphere MQ for Windows
 - channel configuration 146
 - channel-exit programs 398
 - configuration 145
 - intercommunication example 129
 - LU 6.2 connection 129
 - NetBIOS connection 142
 - SPX connection 143
 - TCP connection 142
- WebSphere MQ for z/OS
 - channel configuration 272, 292
 - channel-exit programs 396
 - configuration 272, 291
 - intercommunication example 266
 - LU 6.2 connection 267, 286
 - reset channel sequence numbers 254
 - resolving in-doubt message on
 - channel 255
 - TCP connection 272, 290
 - weighting 79
 - wide-band links 27
 - work with channel status 329
 - work with status 334
 - work-with-channel choices 331
 - workload
 - balanced 283
 - Workload-balanced channel start 283
 - worksheet
 - WebSphere MQ for AIX
 - configuration 156
 - WebSphere MQ for HP-UX
 - configuration 172
 - WebSphere MQ for i5/OS
 - configuration 351
 - WebSphere MQ for Solaris
 - configuration 195
 - WebSphere MQ for Windows
 - configuration 130
 - WebSphere MQ for z/OS
 - configuration 267, 286
 - writing your own message channel
 - agents 70
 - WRKCLS command 350
 - WRKSBSD command 348

X

- XINETD 233
- XMITQ attribute 100
- XmitQName field 417

Sending your comments to IBM

If you especially like or dislike anything about this book, please use one of the methods listed below to send your comments to IBM.

Feel free to comment on what you regard as specific errors or omissions, and on the accuracy, organization, subject matter, or completeness of this book.

Please limit your comments to the information in this book and the way in which the information is presented.

To make comments about the functions of IBM products or systems, talk to your IBM representative or to your IBM authorized remarketer.

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

You can send your comments to IBM in any of the following ways:

- By mail, to this address:

User Technologies Department (MP095)
IBM United Kingdom Laboratories
Hursley Park
WINCHESTER,
Hampshire
SO21 2JN
United Kingdom

- By fax:
 - From outside the U.K., after your international access code use 44-1962-816151
 - From within the U.K., use 01962-816151
- Electronically, use the appropriate network ID:
 - IBM Mail Exchange: GBIBM2Q9 at IBMMAIL
 - IBMLink: HURSLEY(IDRCF)
 - Internet: idrcf@hursley.ibm.com

Whichever method you use, ensure that you include:

- The publication title and order number
- The topic to which your comment applies
- Your name and address/telephone number/fax number/network ID.



SC34-6931-00



Spine information:



WebSphere MQ

Intercommunication

Version 7.0