Unit 9. Distributed queuing

What this unit is about

This unit covers the concepts of interconnecting two or more queue managers in a network, allowing the distribution of messages between systems. Channels are explained, as well as the supporting objects needed for them to function correctly. Remote queue definitions are explained.

What you should be able to do

After completing this unit, you should be able to:

- Diagram the connection between two queue managers by using the required components
- Explain how channels are used by queue managers
- Configure WebSphere MQ channels
- Explain how to start and stop channels
- List some of the states channels can be in
- List strategies to access remote queues
- List considerations for data conversion

How you will check your progress

- Machine exercises
- Classroom discussion

References

SC34-6597 WebSphere MQ Script (MQSC) Command Reference

SC34-6587 WebSphere MQ Intercommunication

SC34-6061 WebSphere MQ Queue Manager Clusters

WebSphere MQ Clients

WebSphere MQ Application Programming Guide

WebSphere MQ System Administration Guide

WebSphere MQ for HP NonStop Server System Management Guide

Unit objectives

After completing this unit, you should be able to:

- Diagram the connection between two queue managers by using the required components
- Explain how channels are used by queue managers
- Configure WebSphere MQ channels
- Explain how to start and stop channels
- List some of the states channels can be in
- List strategies to access remote queues
- List considerations for data conversion

Figure 9-1. Unit objectives

Notes:

This unit is, perhaps, one of the most important in the entire course. Channel configuration and operations can be confusing. Once you have completed this unit and the practical exercise, you should have a better idea of what is involved.

With additional practice and use when you return to your job, you can find that they are not as difficult as they appear.

Distributed configuration overview topic objectives

After completing this topic, you should be able to:

- Define what is meant by distributed queuing
- List the components that make up a distributed queuing environment
- Differentiate between the channel types
- Define the role that transmission queues, listeners and channel initiators play
- Position the message channel exit
- List the configuration file parameters relevant to distributed queuing

Figure 9-3. Distributed configuration overview topic objectives WM203 / VM2032.0

Notes:

Distributed configuration overview

Overview of the components

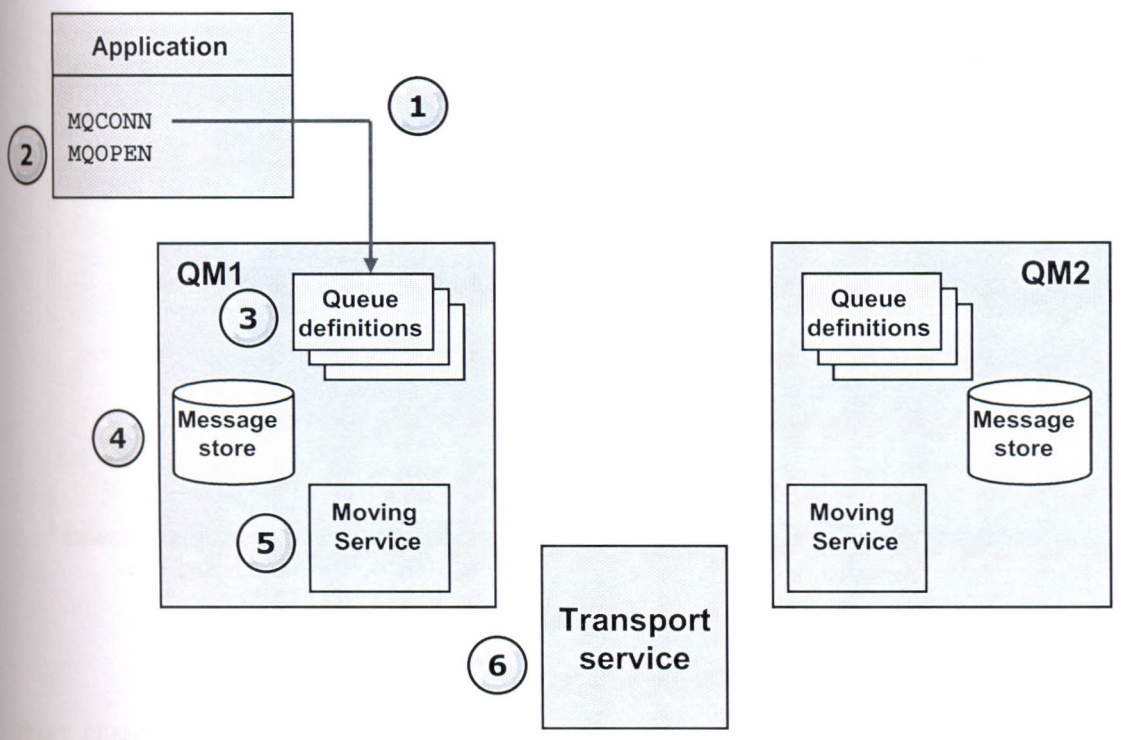


Figure 9-4. Overview of the components WM203 / VM2032.0

Notes:

The figure shows the high-level components of distributed queuing whereby a WebSphere MQ queue manager (QM1) is connected to another queue manager (QM2) so that resources and messages can be shared. An application connects to a queue manager and opens a queue to transmit messages, using a transport service, to another queue manager.

1. An application uses the MQCONN call to connect to a queue manager.
2. The application 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 process can be transparent to the application.

5. Each queue manager contains communications software called the *moving service* component by which 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 methods (depending on the platform):
   - Transmission Control Protocol/Internet Protocol (TCP/IP)
   - Network Basic Input/Output System (NetBIOS)
   - Sequenced Packet Exchange (SPX)
   - Systems Network Architecture (SNA)

The big picture of distributed queuing

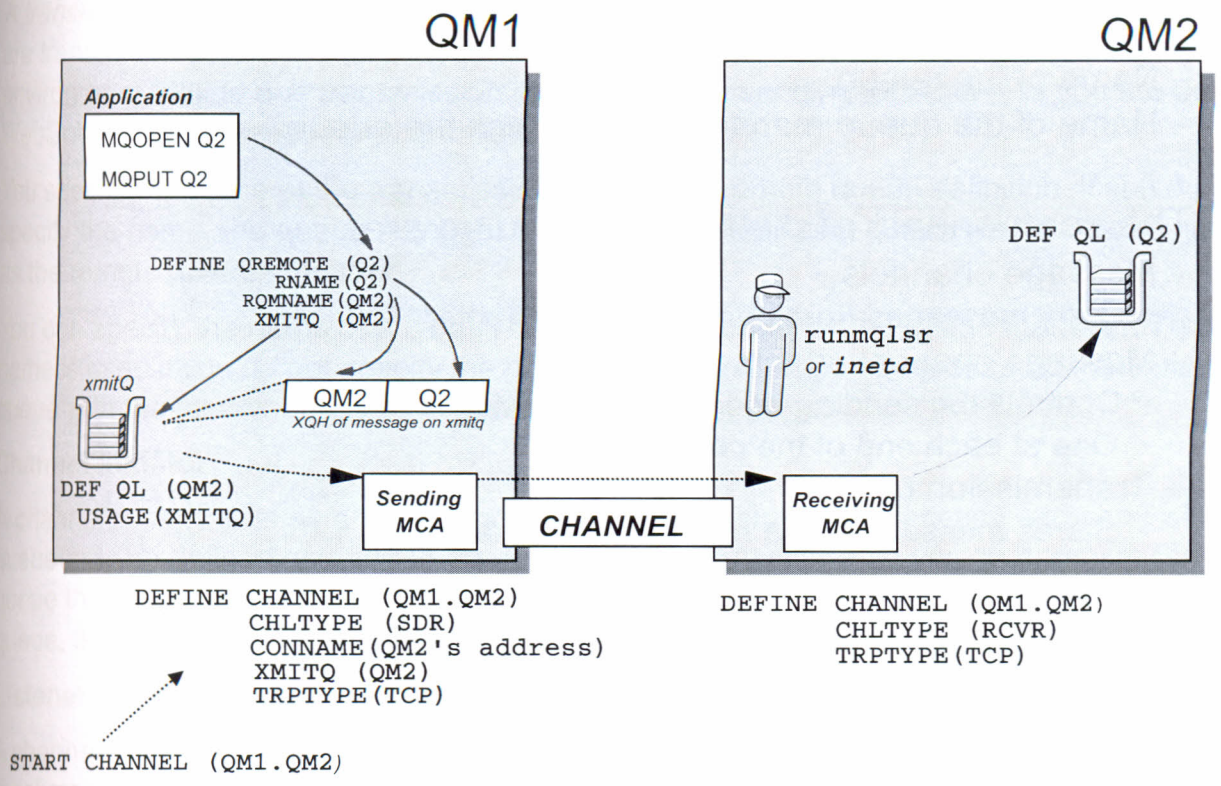


Figure 9-5. The big picture of distributed queuing WM203 / VM2032.0

Notes:

- Whether an application is putting a message on a local queue or to a remote queue is transparent to the application. However, an application always gets a message from a local queue.
- All queue managers use a common protocol for assured message delivery. A message is not lost in the event of a communications or system failure, nor is it ever delivered twice.
- A message destined for a remote queue manager is stored locally on a transmission queue until the massage channel agent (MCA) can send it.

Distributed queuing components

- A queue is identified by:
  - Name of the queue
  - Name of the queue manager which owns the queue
- The components of distributed queuing are:
  - Message channels
    - Carry messages from one QMGR to another
  - Message channel agents
    - Controls the sending and receiving of messages
    - One at each end of the channel
  - Transmission queues
    - Stores messages for a remote queue manager
  - Channel initiators and listeners
    - Act as trigger monitors for the sender channels
  - Channel-exit programs

Figure 9-6. Distributed queuing components WM203 / VM2032.0

Notes:

Message channels

The definition of each end of a message channel can be sender, receiver, server, requester, cluster sender, or cluster receiver.

A message channel is defined using the message channel type 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.

Message channel agents (MCA)

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 can be associated with a sender,

cluster-sender, server (fully qualified), or requester channel. A responder MCA can 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, 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. 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 name 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 Initiator

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

Listener

A channel listener program *listens* for incoming network requests and starts the appropriate receiver channel when it is needed.

Channel-exit programs

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 before 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.

Message channel combinations

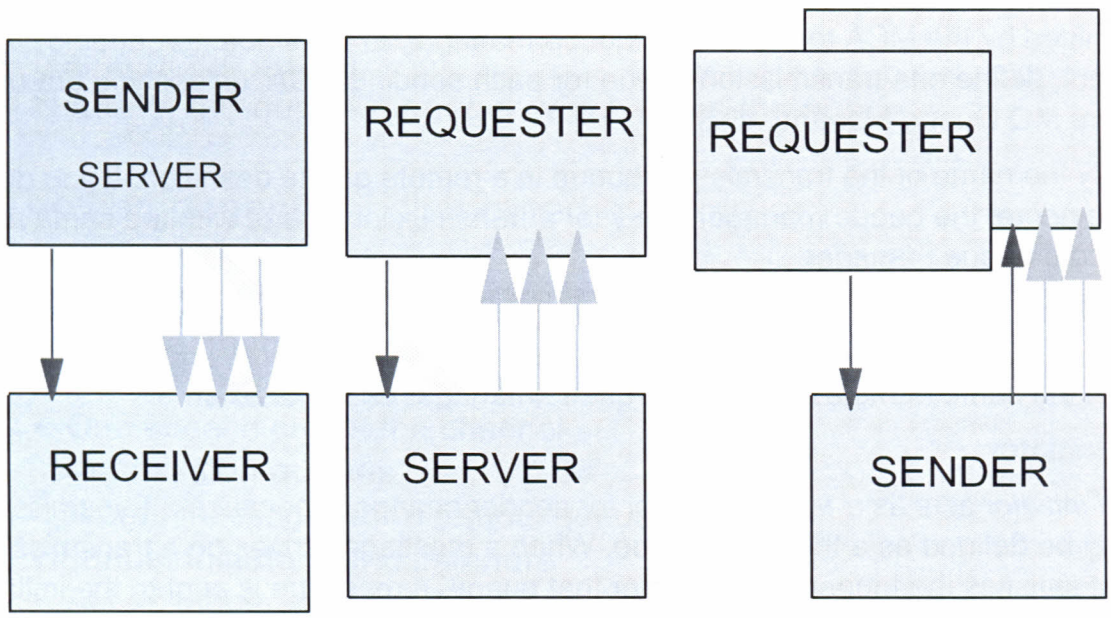


Figure 9-7. Message channel combinations WM203 / VM2032.0

Notes:

WebSphere MQ Intercommunication Guide explains the possible combinations in more detail.

A message channel agent (MCA) is a program that is instrumental in moving messages from one queue manager to another. A pair of MCAs act together to form a *message channel*.

A channel definition provides the parameters controlling the way an MCA operates. Every message channel has two definitions, one at each end of the channel. The name of the channel must be the same in both definitions.

Each end of a channel has a *type* and the definition of a channel at one end specifies the type of the channel at that end. The four possible types are:

- Sender
- Receiver
- Server
- Requester

A sender or server gets messages from a transmission queue and sends them over the network. A receiver or requester receives messages from the network and puts them on their respective destination queues.

A sender can initiate a communications connection with a receiver and then send messages to it. This configuration is the most common combination. A fully defined server may also perform the same role as a sender in this combination.

A requester can initiate a communications connection with a server which then sends messages to the same requester.

A requester can initiate a communications connection with a sender which promptly terminates the connection. The sender then starts a communications connection according to the information in its channel definition and sends messages to the partner it has started. This combination is known as *callback*.

Sender-receiver channels

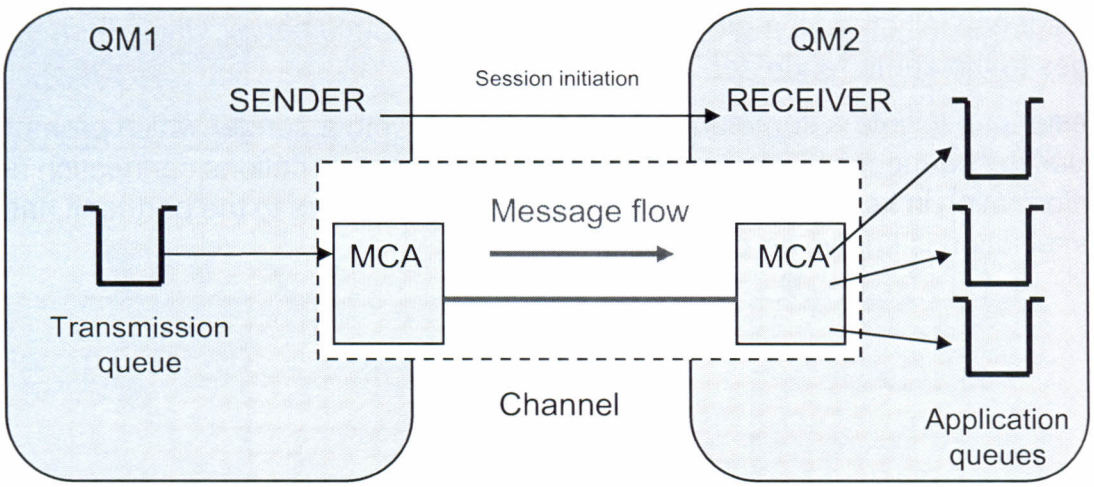


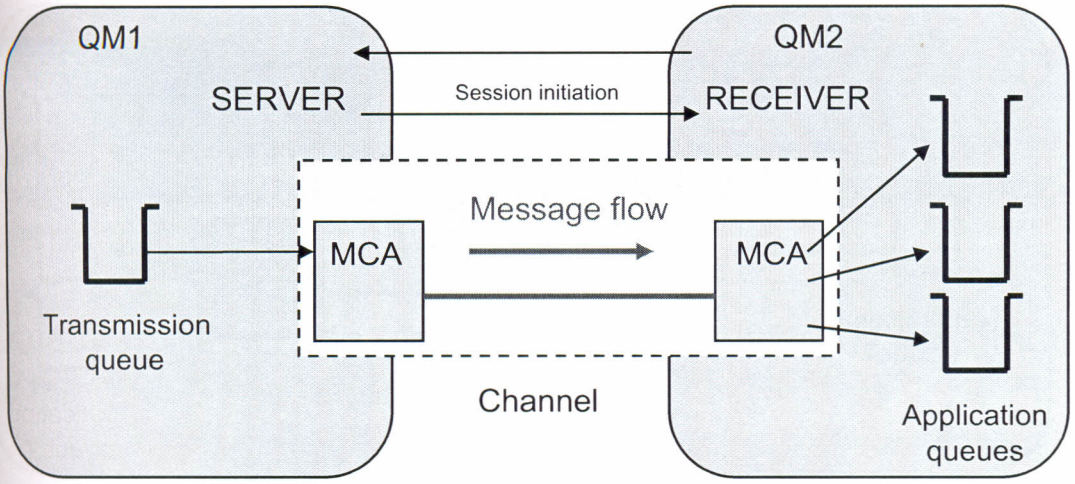
Figure 9-8. Sender-receiver channels WM203 / VM2032.0

Notes:

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.

Server-receiver channel 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.

Requester-server channels



Notes:

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 initiation 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.

Requester-sender channels

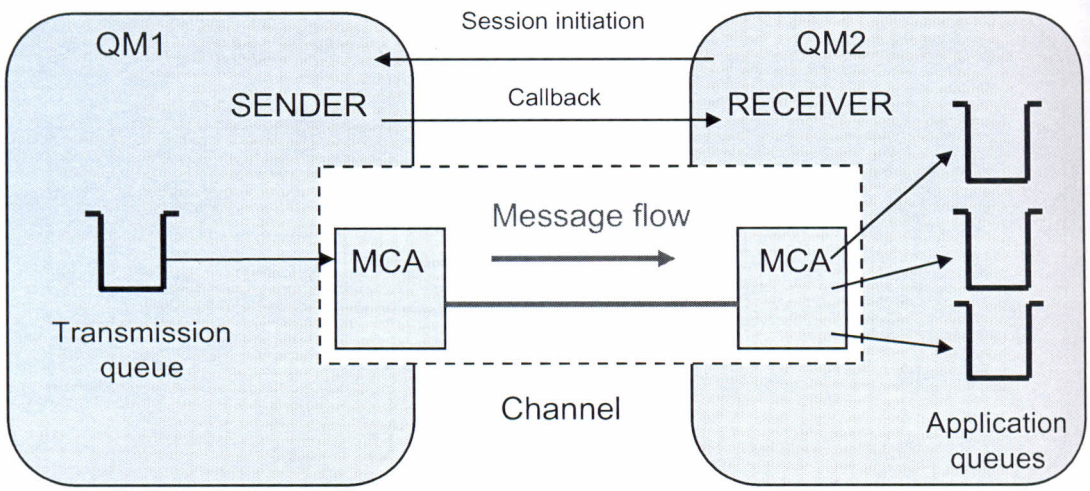


Figure 9-10. Requester-sender channels

Notes:

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

Choosing a transmission queue

1. Specify transmission queue in local definition of a remote queue

```
DEFINE  QREMOTE(BBB)  RNAME(YYY)  +
RQMNAME(QM2)  XMITQ(EXPRESS)
```

2. Name of the transmission queue is inferred from the name of the remote queue manager

3. Default transmission queue can be specified by an attribute of the queue manager object

```
ALTER  QMGR  DEFXMITQ(HOST)
```

4. Error (the MQOPEN fails)

Figure 9-11. Choosing a transmission queue WM203 / VM2032.0

Notes:

Example of using a default transmission queue

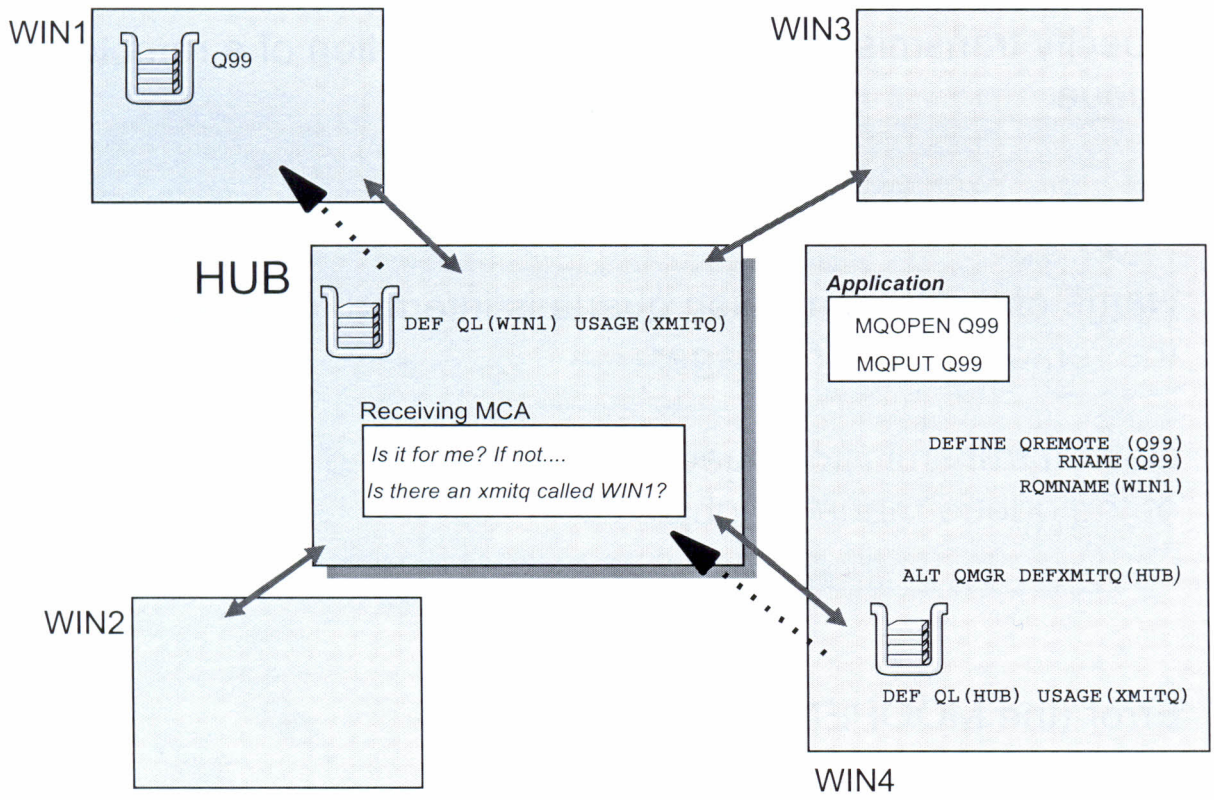


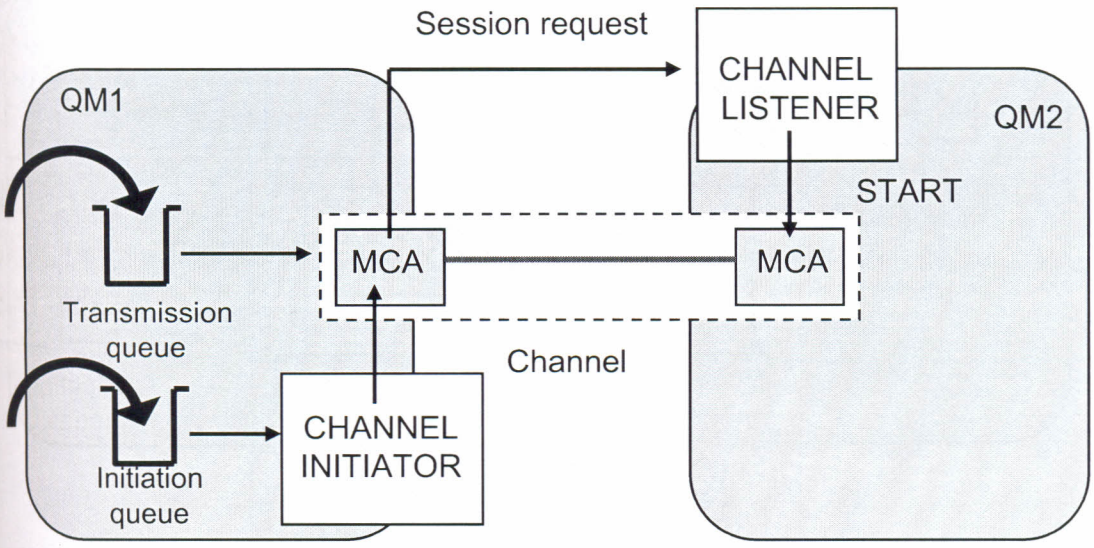
Figure 9-12. Example of using a default transmission queue WM203 / VM2032.0

Notes:

The default transmission queue is the destination for messages to be sent to a remote queue when no other suitable transmission queue is defined. Define it as a local queue, and that queue name placed in the queue manager parameter DEFXMITQ.

Default transmission queues are useful in a hub and spoke network environment. Because each of the spoke queue managers is directly connected only to the hub queue manager, defining the transmission queue to the hub as the default transmission queue for each spoke queue manager is a logical network design.

Channel initiators and listeners



Notes:

The figure shows a message flow that uses a channel initiator and a listener. A message is put on the initiation queue of QM1. The message triggers the channel initiator for that queue manager which starts the sender channel. The channel listener at QM2 receives the session request and starts the receiving MCA. Messages can then be transmitted from QM1 to QM2.

A *channel initiator* acts as a *trigger monitor* for sender channels, because a transmission queue can 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. Server channels can also be started in this way if the connection name of the partner is specified in the channel definition. 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.

Channel-exit programs

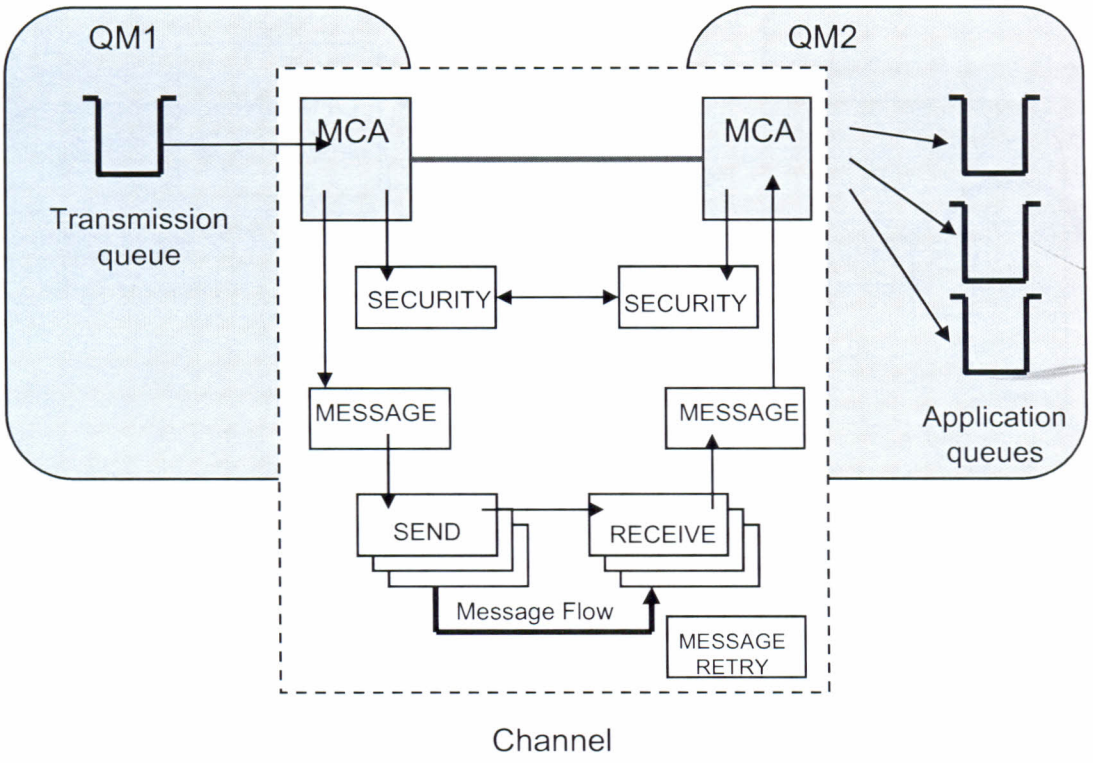


Figure 9-14. Channel-exit programs WM203 / VM2032.0

Notes:

The sequence of processing is as follows:

1. The security exits are called after the initial data negotiation between both ends of the channel. Security exits 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.

Configuration file stanzas for distributed queuing

The stanzas that relate to distributed queuing are:

- CHANNELS
- TCP
- LU62
- NETBIOS
- SPX (Windows XP and Windows 2003 Server only)
- EXITPATH

Figure 9-15. Configuration file stanzas for distributed queuing WM203 / VM2032.0

Notes:

The figure 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. The file is called `qm.ini`.

qm.ini stanza for distributed queuing

```
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=chltype    ; 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.
ADOPTNEWMCACHECK=      ; Specifies the type checking required.
          typecheck    ; The default is "NAME","ADDRESS", and "QM".

TCP:                   ; TCP entries
PORT=n                 ; 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 9-16. qm.ini stanza for distributed queuing WM203 / VM2032.0

Notes:

The figure includes a screen capture of parameters in the configuration file that are used to define the distributed queuing components.

Basic channel types

- For distributed queuing:
  - Sender
  - Server
  - Receiver
  - Requester
- For clustered environments
  - Cluster-sender
  - Cluster-receiver
- For MQ clients
  - Client-connection
  - Server-connection

Figure 9-19. Basic channel types

Notes:

Minimum definition

- Two channels required at minimum
  - SENDER
  - RECEIVER
- On the source queue manager:
  - Channel type of SENDER
    - XMITQ specifies the name of the transmission queue
    - CONNAME defines the connection name of the partner system
    - TRPTYPE specifies the transport protocol
- On the target queue manager:
  - Channel type RECEIVER
    - Same name as sender channel
    - TRPTYPE to specify the transport protocol
- Test channel using PING command

Figure 9-20. Minimum definition WM203 / VM2032.0

Notes:

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 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 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).

The receiver channel definitions can be generic. Generic channels 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 applies to them all.

Define channel command

```
DEF  CHL('channel') CHLTYPE(string) TRPTYPE(string)
```

- Required for definition

(*channel-name*)	Channel name up to 20 characters. Must be first parameter.
CHLTYPE(*string*)	Channel type Sender: SDR Receiver: RCVR Server: SVR Requestor: RQSTR
TRPTYPE	Transport type: DECNET, LU62, NETBIOS, SPX, TCP
CONNAME(*string*)	For SDR and RQSTR only, optional for SVR
XMITQ(*string*)	For SDR and SVR only

- Required for SNA LU6.2
  MODENAME(*string*)
  TPNAME(*string*)

Figure 9-21. Define channel command WM203 / VM2032.0

Notes:

The WebSphere MQ command to define a message channel at one end is DEFINE CHANNEL. Related commands are ALTER CHANNEL, DISPLAY CHANNEL, and DELETE CHANNEL.

Attributes not supplied on the DEFINE CHANNEL command are taken from the appropriate default channel object.

- SYSTEM.DEF.SENDER
- SYSTEM.DEF.RECEIVER
- SYSTEM.DEF.SERVER
- SYSTEM.DEF.REQUESTER

CHLTYPE must be included as the first parameter on both the DEFINE CHANNEL and ALTER CHANNEL commands.

TRPTYPE is a required parameter on the DEFINE CHANNEL command. If TRPTYPE is not specified, the value is taken from the appropriate default channel object.

Defining channels example

Hursley

```
DEF CHL('Hursley_Dallas') +
CHLTYPE(SDR) TRPTYPE(TCP) +
 CONNAME(9.84.100.1) +
  XMITQ('Dallas')

DEF QL('Dallas') USAGE(XMITQ)



DEF CHL('Dallas_Hursley') +
CHLTYPE(RCVR) TRPTYPE(TCP)
```

Dallas

```
DEF CHL('Hursley_Dallas') +
CHLTYPE(RCVR) TRPTYPE(TCP)



DEF CHL('Dallas_Hursley') +
CHLTYPE(SDR) TRPTYPE(TCP) +
 CONNAME(9.20.31.5) +
  XMITQ('Hursley')

DEF QL('Hursley') USAGE(XMITQ)
```


Figure 9-22. Defining channels example

Notes:

Each message channel has a two channel definitions, one at each end.

In the example, the sending end of each channel has a transmission queue with the same name as the queue manager at the other end of the channel.

Configuring TCP/IP for WebSphere MQ

- Configure the inet daemon, `inetd` (UNIX or Linux only)

`/etc/services`

```
MQSeries      1414/tcp      #MQSeries channel listener
```

`/etc/inetd.conf`

```
MQSeries stream  tcp  nowait  root  /usr/mqm/bin/amqcrsta amqcrsta
```

- Use the WebSphere MQ listener, `runmqlsr`
- To end the listener, `endmqlsr` command

Notes:

- Before any channels are started, you must ensure that TCP/IP is configured and that a listener is started which is alerted to any incoming requests.
- The inet daemon, inetd, is available for TCP/IP on UNIX and on Linux. It has to be configured for WebSphere MQ.
  - Change the two files as shown, adjusting path names as appropriate.
  - The inet daemon then has to be refreshed. On AIX, issue: refresh -s inetd
    - Add a line to /etc/inetd.conf

```
WebSphere MQ stream tcp nowait mqm /usr/mqm/bin/amqcrsta amqcrsta
[-m QMgrName]
```

    - Enter the command refresh -s inetd
- Port number 1414 is assigned by the Internet Assigned Numbers Authority to WebSphere MQ and, by default, WebSphere MQ uses this port number. However, if you are running multiple queue managers on a system, each needing to be able to

respond to incoming requests to start a channel, you need to specify additional port numbers.

- On Windows, no inet daemon is supplied and so the WebSphere MQ listener must be used instead.
- The WebSphere MQ listener, **runmqlsr**, is available on other queue managers. On WebSphere MQ for Windows, you can use the listener for all the supported communications protocols. For NetBIOS, IPX/SPX, and TCP/IP, it is the only option available. On the WebSphere MQ UNIX platforms, **runmqlsr** is also available.

The *WebSphere MQ Intercommunication* contains full details on how to configure TCP/IP for WebSphere MQ on all platforms except HP NonStop Server. For HP NonStop Server, consult the *System Management Guide.*

Listener object

- Create a listener object:

```
DEFINE LISTENER(LISTENER.TCP) TRPTYPE(TCP) +
   PORT(1414) +
   CONTROL(QMGR) +
   REPLACE
```

- Start the listener:

```
START LISTENER(LISTENER.TCP)
```


Figure 9-26. Listener object

Notes:

- A channel listener program is defined and run on each queue manager. A channel listener program *listens* for incoming network requests and starts the appropriate receiver channel when it is needed.
- The implementation of channel listeners is platform-specific, however there are some common features. On all WebSphere MQ platforms, the listener may be started using the MQSC command START LISTENER.
- On WebSphere MQ for Windows systems and UNIX systems, you can make the listener start automatically with the queue manager by setting the CONTROL attribute of the LISTENER object to QMGR or STARTONLY.

On Windows and UNIX:

- Use either the channel listener program provided by WebSphere MQ, or the facilities provided by the operating system. To start the WebSphere MQ channel listener use the RUNMQLSR command.
  - For example: `RUNMQLSR -t tcp -p 1414 -m QM1`