Starting a message channel

- WebSphere MQ commands

```
PING CHANNEL(QMA_QMB)
```

```
START CHANNEL(QMA_QMB)
```

- Control command

```
runmqchl  -c  QMA_QMB
```

- Channel attributes

BATCHSZ	Batch size
MAXMSGL	Maximum message length
SEQWRAP	Sequence number wrap
HBINT	Heartbeat interval
NPMSPEED	Nonpersistent message speed

Figure 9-27. Starting a message channel WM203 / VM2032.0

Notes:

- Check the network first (for example, issue a TCP/IP ping).
- Use the WebSphere MQ command PING CHANNEL to test a message channel configuration. It can only be used at the sender or server end of a channel, and only when the channel is not started.
- Take particular care that the channel definitions are correct.
- There are several reasons why a channel might fail to start. Some common problems are:
  - The definitions at both ends of a channel do not specify the same channel name. Remember, in particular, that the names of channels, like the names of queues, are case-sensitive.
  - A channel is not defined at both ends with compatible types.
  - There is a sequence number mismatch, often caused by deleting a channel definition at one end of a message channel and then redefining it, or deleting and creating a queue manager again, and so on.

- Some attributes specified in the channel definition at each end of a message channel are compared when the channel is started.

BATCHSZ - Maximum number of messages sent before a sync point is taken. A larger maximum batch size can improve throughput. The lower value from the two channel definitions is used.

MAXMSGL - Maximum message length that can be transmitted by the channel. The lower value from the two channel definitions is used.

SEQWRAP - Highest message sequence number before wrapping. The SEQWRAP values in the two channel definitions must match otherwise the channel fails to start. Messages are numbered internally in the Message Channel Protocol and the sequence numbers used will restart at 1 after reaching the value specified on the SEQWRAP parameter. Do not specify a low value.

HBINT - The time between heartbeat flows that are sent from a sending MCA to a receiving MCA when there are no messages on the transmission queue. A heartbeat flow can unblock a receiving MCA for which the STOP CHANNEL command has been issued. The higher value from the two channel definitions is used.

- This parameter is only valid on a queue manager which supports heartbeat flows, namely, WebSphere MQ for AIX, HP OpenVMS, HP-UX, Linux, i5/OS, Solaris, Windows, and z/OS. Heartbeat flows only occur when there is a queue manager that supports them at both ends of a channel.

NPMSPEED - The speed at which nonpersistent messages are to be sent. You can specify NORMAL or FAST. The default is FAST, which means that a nonpersistent message is sent outside of a batch and so becomes available for retrieval as soon as it is put on its destination queue. If the definitions at the sending and receiving ends of a channel do not specify the same value, or if one end does not support fast nonpersistent messages, then NORMAL is used.

- This parameter is only valid on WebSphere MQ for AIX, HP OpenVMS, HP-UX, Linux, i5/OS, Solaris, Windows, and z/OS.
- If a message cannot be delivered, it is put on the local dead letter queue. If it cannot be put there, the channel is stopped. However, if a fast nonpersistent message cannot be delivered and cannot be put on the dead letter queue, it is discarded and the channel remains open.

Channel initiator

- Two main functions:
  - Triggering a channel
  - Retrying channels
- Starts an MCA at sending end of a message channel
  - *UserData* attribute of process object specifies channel name

  Or
  - *TrigData* attribute of transmission queue specifies channel name
- Channel control parameters

`DISCINT`	Disconnect interval
`SHORTRTY, SHORTTMR`	Short retry count, short retry interval
`LONGRTY, LONGTMR`	Long retry count, long retry interval
`MRRTY, MRTMR`	Message retry count, message retry interval
`MCATYPE`	Message channel agent type
`BATCHINT`	Batch interval

Figure 9-28. Channel initiator WM203 / VM2032.0

Notes:

- The channel initiator is a special trigger monitor for starting a message channel. It also contains retry logic for use in case of difficulty in starting a channel or after an error on a channel.
- For triggering a channel, the attributes *ApplType* and *ApplId* of a process object are not required. Instead, the attribute *UserData* specifies the name of the channel to be started.
- You do not need to define a process in order to trigger a channel. Instead, the attribute *TriggerData* of the transmission queue specifies the name of the channel to be started.
- If you do not specify a channel name at all, the channel initiator searches the channel definitions until it finds a channel whose definition contains the name of the transmission queue that has caused the trigger event.
- For WebSphere MQ for HP NonStop Server, the default channel initiator can be configured in the TS/MP server class MQS-CHANINIT00 and start it by using the PATHCOM commands THAW SERVER and START SERVER. Full details of this option

can be found in the *WebSphere MQ for HP NonStop Server System Management Guide.*

- The WebSphere MQ command START CHINIT is not supported on this queue manager.

- The channel control parameters shown on the visual can be specified on the DEFINE CHANNEL command.

DISCINT - (SDR and SVR)
Maximum time to wait for a message on the transmission queue if it is empty. If no message arrives within this time, the channel closes down.

SHORTRTY, SHORTTMR - (SDR and SVR)
Short retry count and short retry time interval to control repeated attempts to establish a communications connection.

LONGRTY, LONGTMR - (SDR and SVR)
Long retry count and long retry time interval to control further repeated attempts to establish a communications connection.

MRRTY, MRTMR - (RCVR and RQSTR)
Message-retry count and message-retry interval when attempting to put a message on a destination queue. If every attempt fails, the MCA decides that it cannot deliver the message.

MCATYPE - (SDR, SVR, and RQSTR)
The value of this parameter may be THREAD or PROCESS. If THREAD is specified, each channel runs as a thread within the channel initiator process. If PROCESS is specified, each channel runs as a separate process.

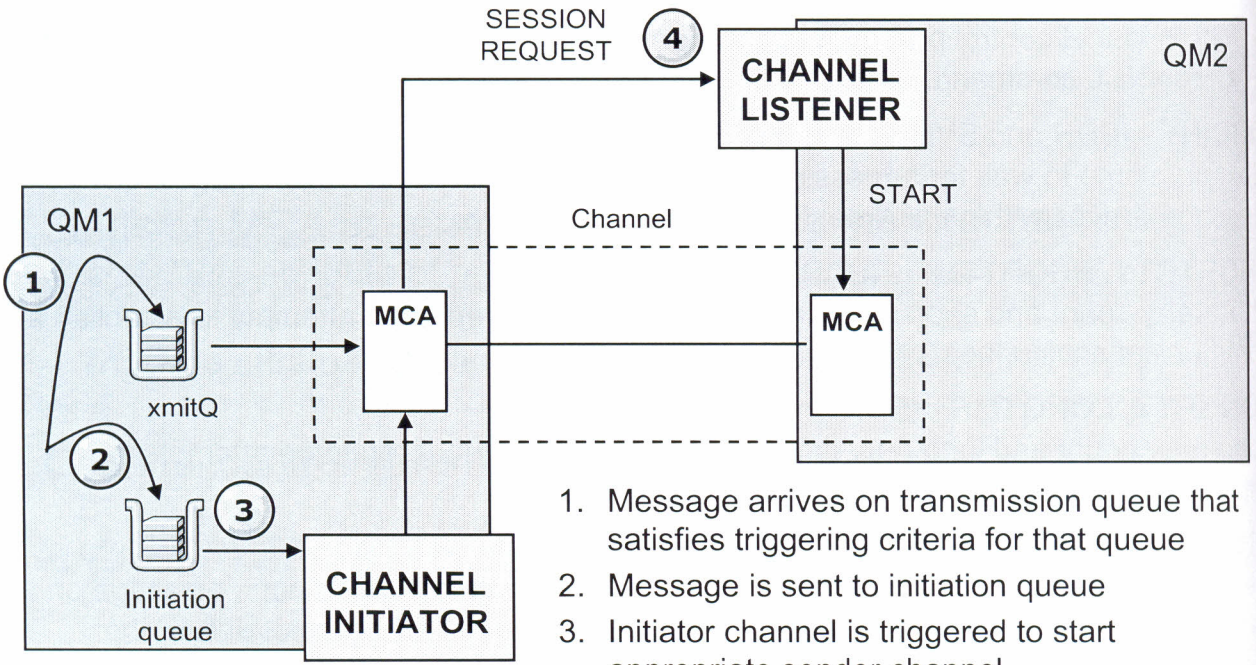
- This parameter is only supported on AIX, HP OpenVMS, HP-UX, Linux, i5/OS, Solaris, and Windows.

BATCHINT - (SDR and SVR)
The length of time during which a channel keeps a batch open if there are no messages on the transmission queue. Set this parameter to a value considerably less than the value of DISCINT.

- This parameter is only valid on AIX, HP OpenVMS, HP-UX, Linux, i5/OS, Solaris, Windows, and z/OS.

Of the parameters listed, only the SHORTRTY, SHORTTMR, LONGRTY, LONGTMR, and MCATYPE parameters are used by the channel initiator.

Channel initiator triggering



1. Message arrives on transmission queue that satisfies triggering criteria for that queue
2. Message is sent to initiation queue
3. Initiator channel is triggered to start appropriate sender channel
4. Channel listener program starts receiving MCAs.

Figure 9-29. Channel initiator triggering WM203 / VM2032.0

Notes:

Item 2: The Queue Manager places the trigger message on the INITIATION QUEUE

Item 3: The Channel Initiator gets the message from the INITIATION QUEUE and uses it to start the appropriate channel

Item 4: The sender channel causes the Listener to start the remote Receiver channel which then receives message sent from XMITQ

Channel states – DISPLAY CHSTATUS command (1 of 3)

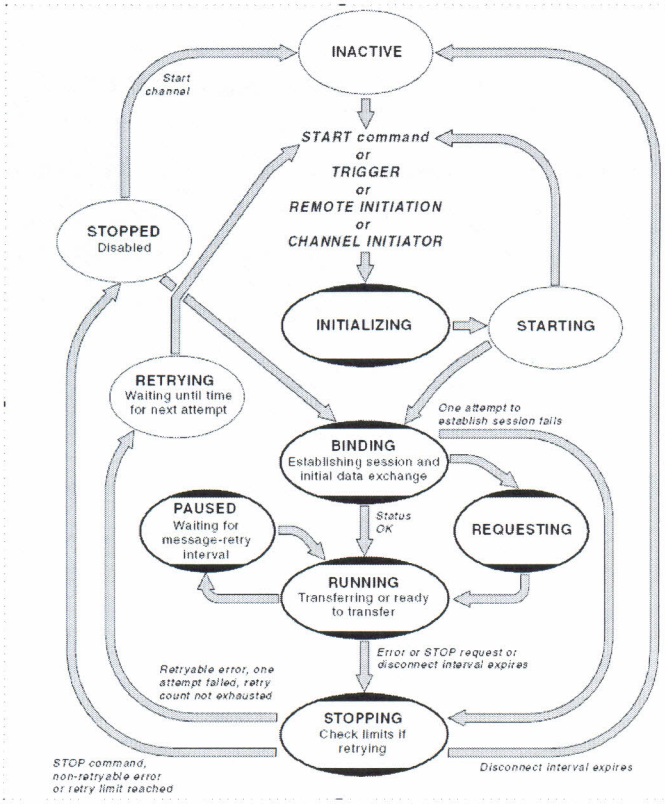


Figure 9-30. Channel states – DISPLAY CHSTATUS command (1 of 3) WM203 / VM2032.0

Notes:

When a channel is in one of the six states highlighted in the figure (INITIALIZING, BINDING, REQUESTING, RUNNING, PAUSED, or STOPPING), it is consuming resource and a process or thread is running; the channel is active.

Note

This diagram is expanded in the following pages for clarity.

Channel states – DISPLAY CHSTATUS command (2 of 3)

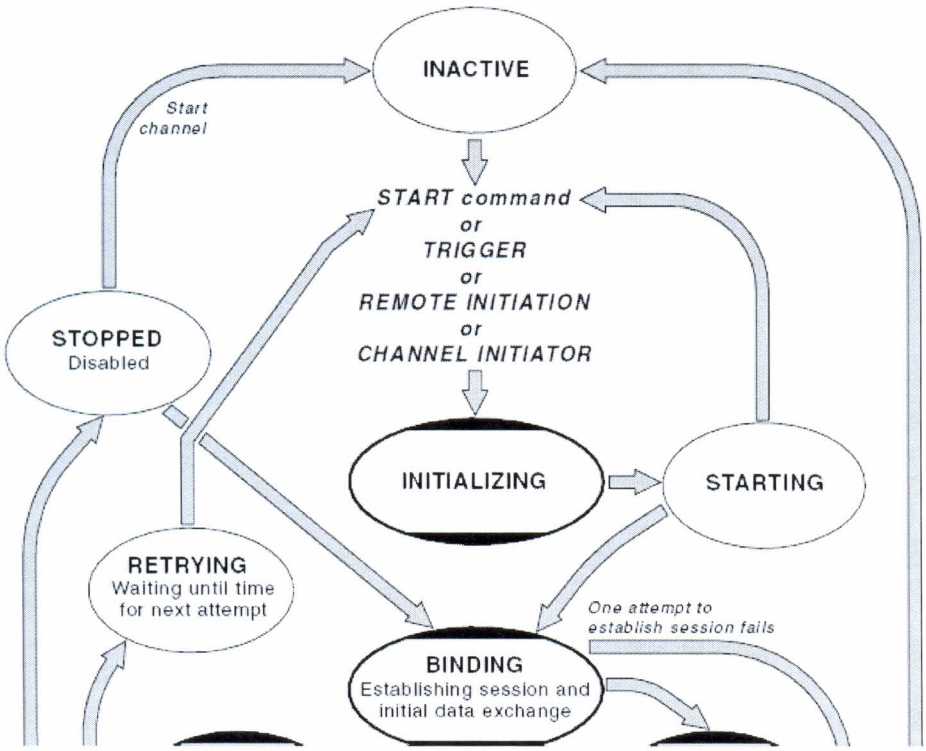


Figure 9-31. Channel states – DISPLAY CHSTATUS command (2 of 3) WM203 / VM2032.0

Notes:

- The current state of a channel can be determined by using the WebSphere MQ command **DISPLAY CHSTATUS**.
- The visual shows the possible states of a channel and the possible flows from one state to the next.
- START is not really a state. It indicates a start point for when the START CHANNEL command has been issued, or when a transmission queue has been triggered, or when a channel initiator has decided it is time for another retry attempt, or when there has been an incoming request to start a channel.

INITIALISING - A channel initiator is attempting to start a channel.

STARTING - A channel waits in this state if there is no active slot available. If there is an active slot immediately available, it remains in this state a short time.

Channel states – DISPLAY CHSTATUS command (3 of 3)

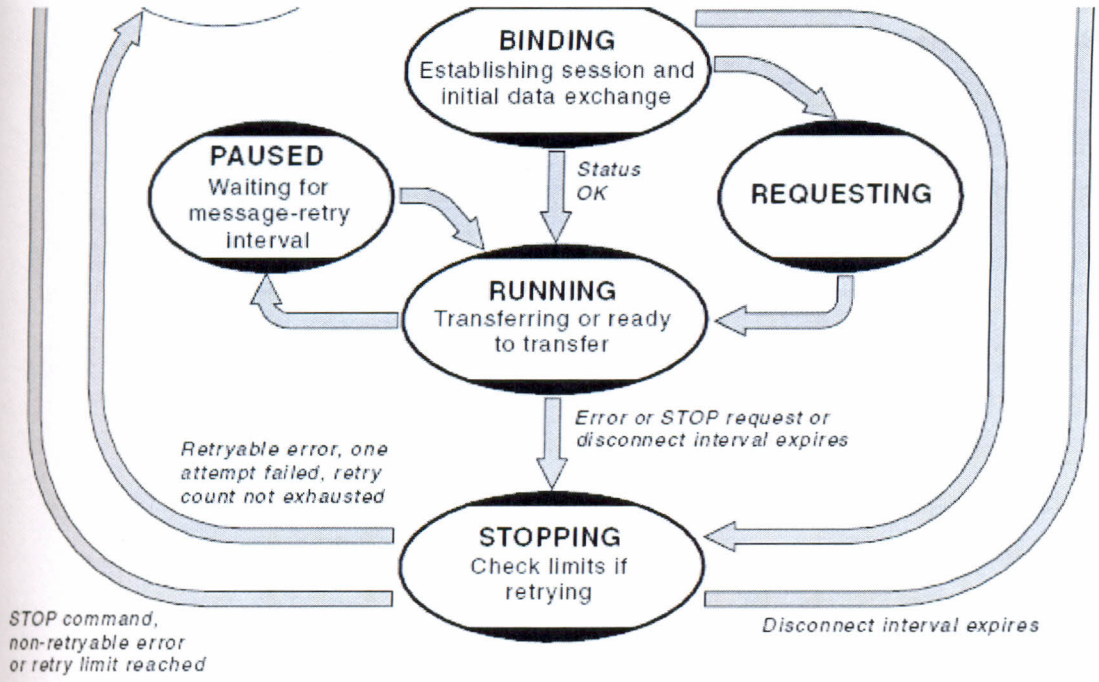


Figure 9-32. Channel states – DISPLAY CHSTATUS command (3 of 3) WM203 / VM2032.0

Notes:

BINDING - Establishing a communications connection and performing the initial data exchange.
REQUESTING - A requester is waiting for callback from a sender.
RUNNING - Transferring messages, or waiting for messages to arrive on the transmission queue.
PAUSED - Waiting for the message-retry interval to complete before attempting to put a message on its destination queue.
STOPPING - An error has occurred, or the STOP CHANNEL command has been issued, or the disconnect interval has expired.
RETRYING - Waiting until it is time for the channel initiator to make the next attempt to start the channel.
STOPPED - In this state, the channel is disabled and needs manual intervention to start it again.
INACTIVE - An inactive channel is one that has never been started or has disconnected normally.

Queue definitions for distributed queuing

- Local definition of a remote queue

```
DEFINE  QREMOTE(BBB) RNAME(YYY)  RQMNAME(QM2)
```

- A transmission queue must be created at the sending end of each message channel
  - `USAGE(XMITQ)` indicates its purpose
  - It can have any of the attributes of a local queue

```
DEFINE  QLOCAL(QM2)  USAGE(XMITQ)
```


Figure 9-35. Queue definitions for distributed queuing WM203 / VM2032.0

Notes:

- 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. Remote queues give location independence.
- The location of a remote queue can be made transparent to an application by creating a local definition of a remote queue using the WebSphere MQ command DEFINE QREMOTE. When an application issues an MQOPEN call, in the object descriptor, it can set *ObjectName* to the name of the local definition of the remote queue and *ObjectQMgrName* to blank. Later, the local definition of the remote queue can be redefined without requiring any change to the application code.
- As a rule, give a transmission queue the same name as the remote queue manager.

Define QREMOTE

- Remote queues are owned by another QMGR
- Define a remote queue definition

```
DEFINE QREMOTE(PAYROLL.QUERY) DESCR('Remote queue for QM2') REPLACE +
  RNAME(PAYROLL) RQMNAME(QM2) XMITQ(QM2)
```

- This example defines a remote queue called PAYROLL.QUERY on the local queue manager. The physical queue is called PAYROLL on a remote queue manager QM2. The transmission queue used by the local queue manager where the message is PUT is called QM2.

Figure 9-36. Define QREMOTE WM203 / VM2032.0

Notes:

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. Remote queues give location independence.

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

How to get to a remote queue

Methods used to access a remote queue manager:

- Multi-hopping
  - Using intermediate queue managers
  - Channel and XMITQ definitions required
- Channel sharing
  - Remote queue definitions specify the same XMITQ
- Using different channels
- Using clustering

Figure 9-37. How to get to a remote queue WM203 / VM2032.0

Notes:

You may not always have one channel between each source and target queue manager. These are 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 behavior is known as a *multi-hop*.
- Define channels between all the queue managers, and transmission queues on the intermediate queue managers.

Channel Sharing

- An application designer has the choice of forcing the 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.

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 are still delivered to the same target queues.

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 can 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*.

Information

More on clustering in a following unit.

Multi-hopping

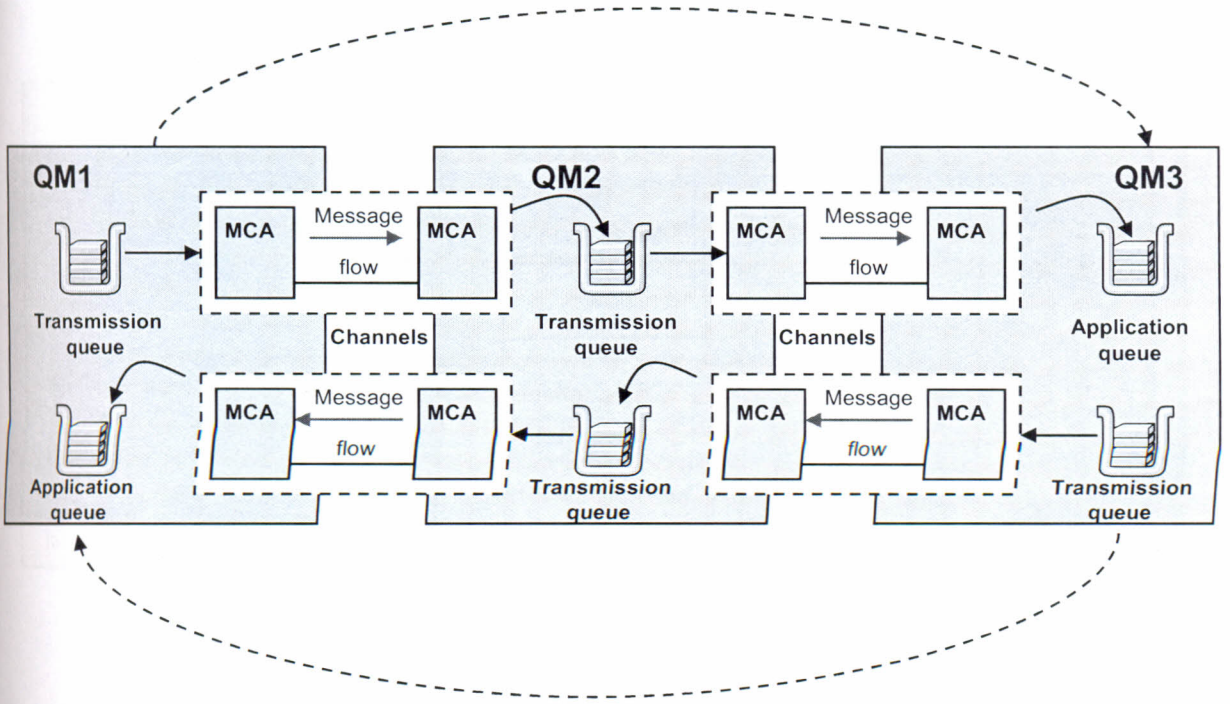


Figure 9-38. Multi-hopping

WM203 / VM2032.0

Notes:

- 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 behavior is known as a *multi-hop*.
- You need to define channels between all the queue managers, and transmission queues on the intermediate queue managers.
- The figure shows three queue managers. In order to send messages to the destination queue on QM3, an intermediate queue manager, QM2, is used.
- Messages are sent from QM1 to the transmission queue on QM2, and then on to the destination queue on QM3
- Channel definitions exist between QM1 and QM2, and between QM2 and QM3.

Channel sharing

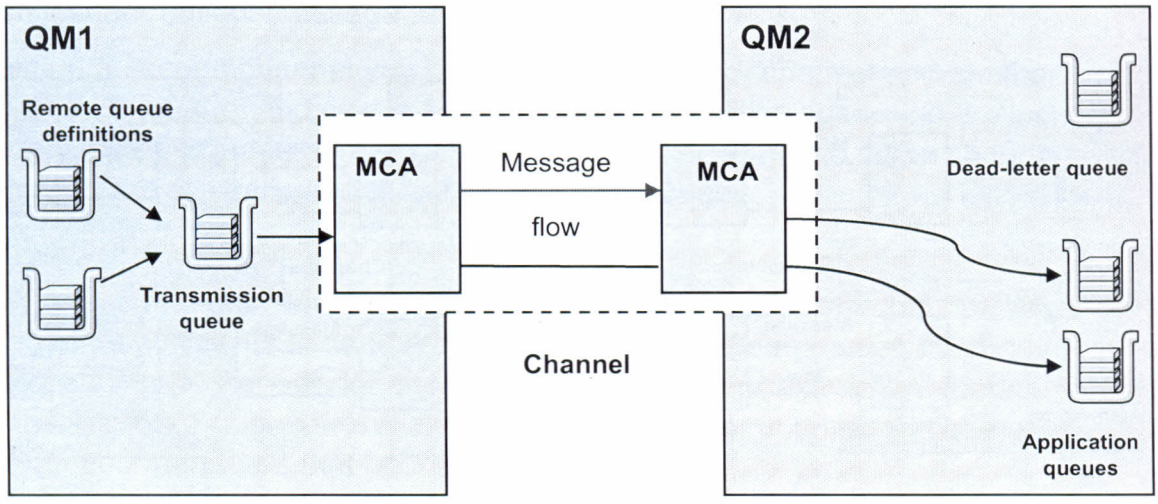


Figure 9-39. Channel sharing WM203 / VM2032.0

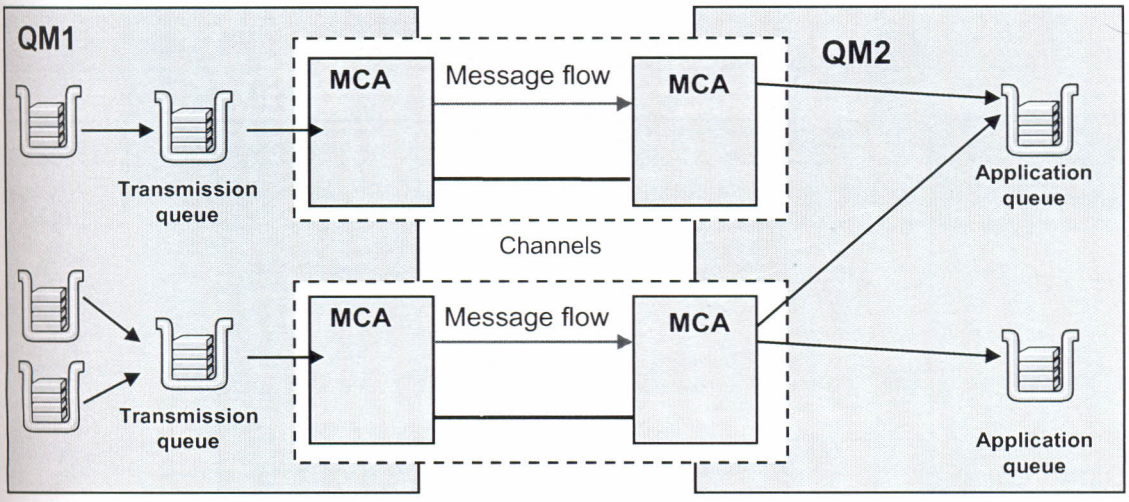
Notes:

This diagram shows a message flow from QM1 to QM2. Remote queue definitions on QM1 enable QM1 to put messages to a remote queue manager. The messages are sent using the transmission queue on QM1

An application designer, has the choice of forcing 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.

Using different channels



Notes:

The figure shows multiple channels between two queue managers. The sending queue manager, QM1 has two channels defined, with a transmission queue for each channel. Messages are sent to the receiving queue manager, QM2 using both 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 are still delivered to the same target queues.

Using a queue manager alias

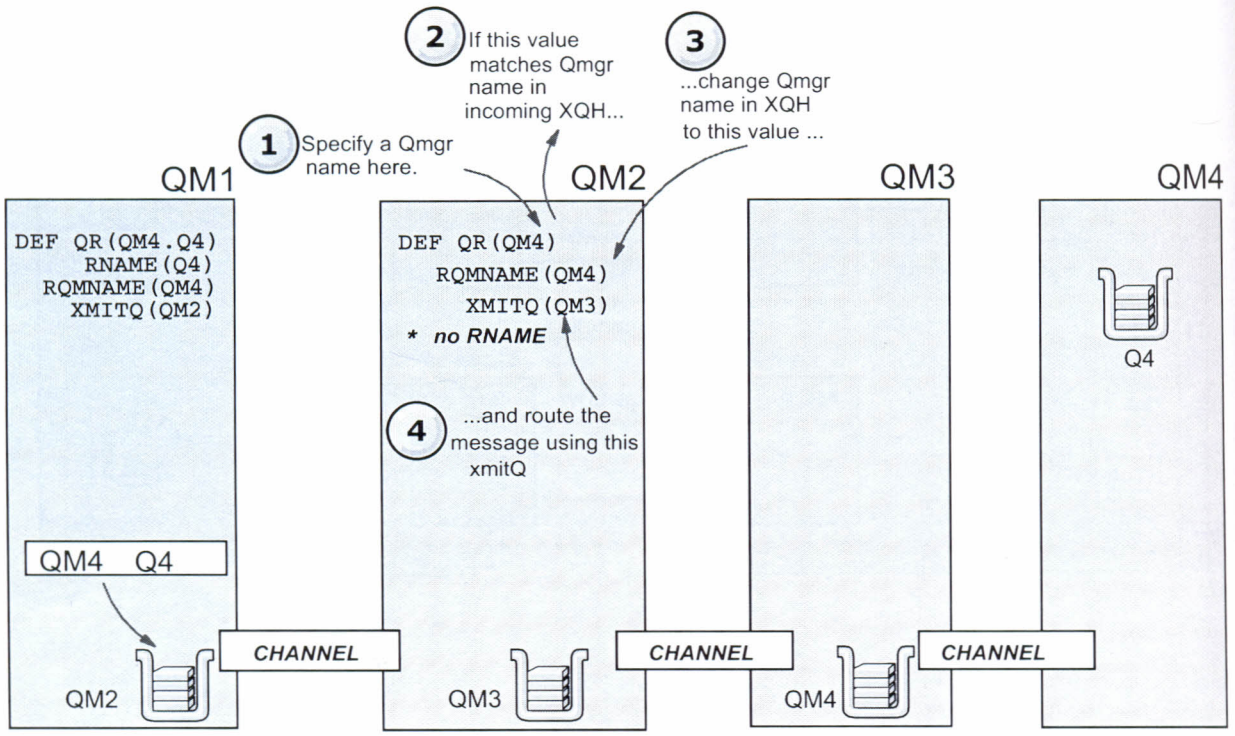


Figure 9-41. Using a queue manager alias

WM203 / VM2032.0

Notes:

- The WebSphere MQ command `DEFINE QREMOTE` is used to define a queue manager alias. The value of the RNAME parameter must be blank, which is the default value in any case.
- 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

Separating message flows

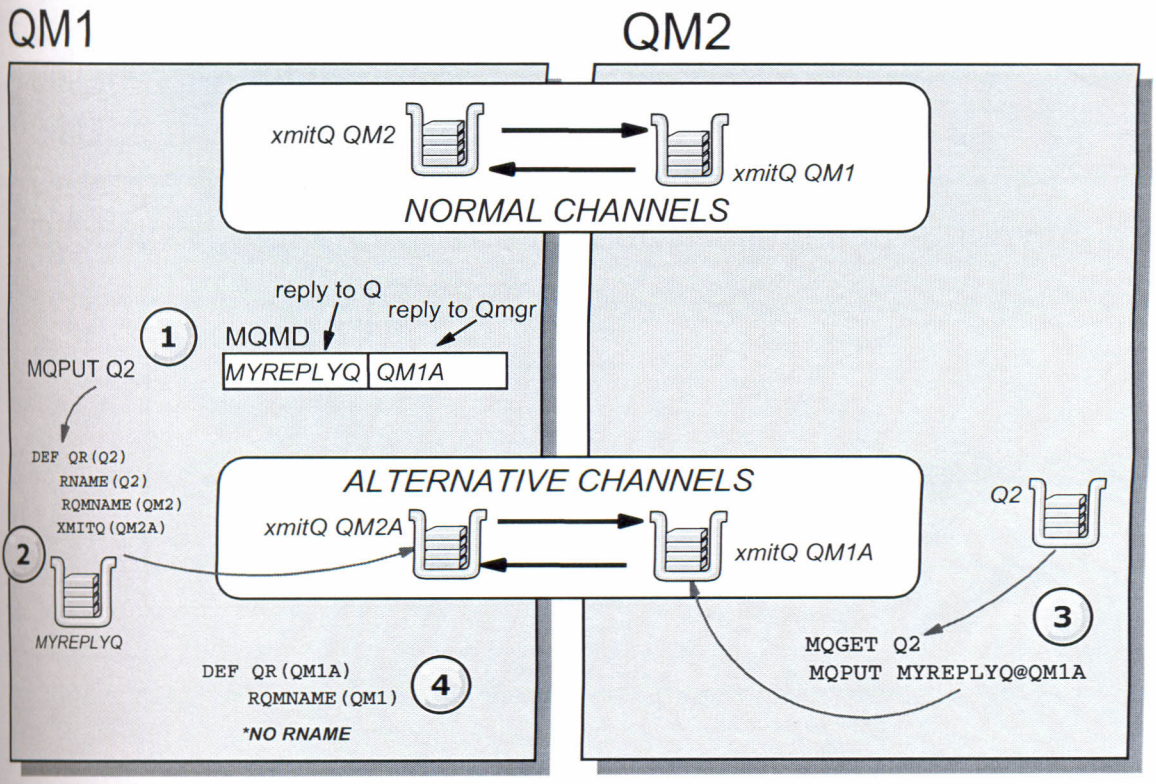


Figure 9-42. Separating message flows

WM203 / VM2032.0

Notes:

- On each queue manager, the normal transmission queue has the same name as the partner queue manager.
- On queue manager QM1, a local definition of a remote queue specifies an alternative transmission queue which can be used for sending messages to queue manager QM2.
- A reply-to queue alias is used to set the value of reply-to queue manager QM1A.
- There is also a queue manager alias definition which specifies QM1A as an alias of QM1. These three definitions enable request messages to be separated into two flows between queue managers QM1 and QM2, and enable the reply messages to be separated into two flows in the reverse direction.

When a message arrives at a queue manager

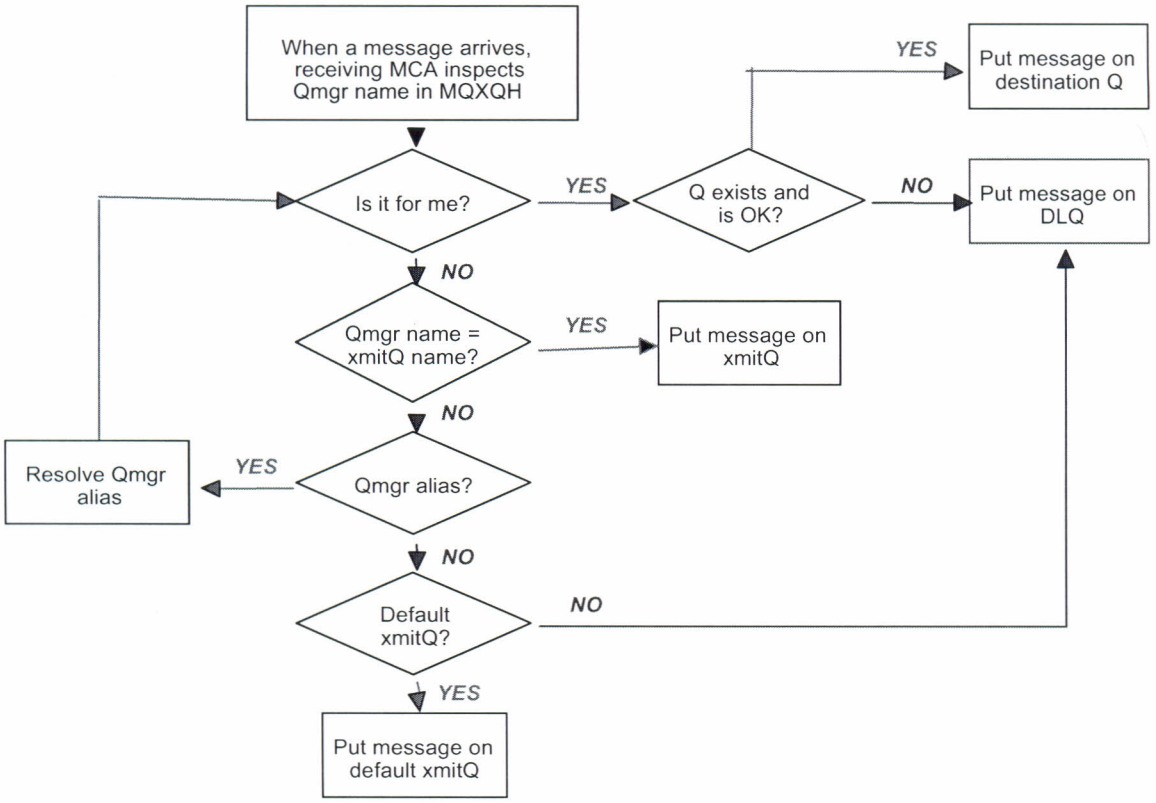


Figure 9-43. When a message arrives at a queue manager WM203 / VM2032.0

Notes:

Data conversion

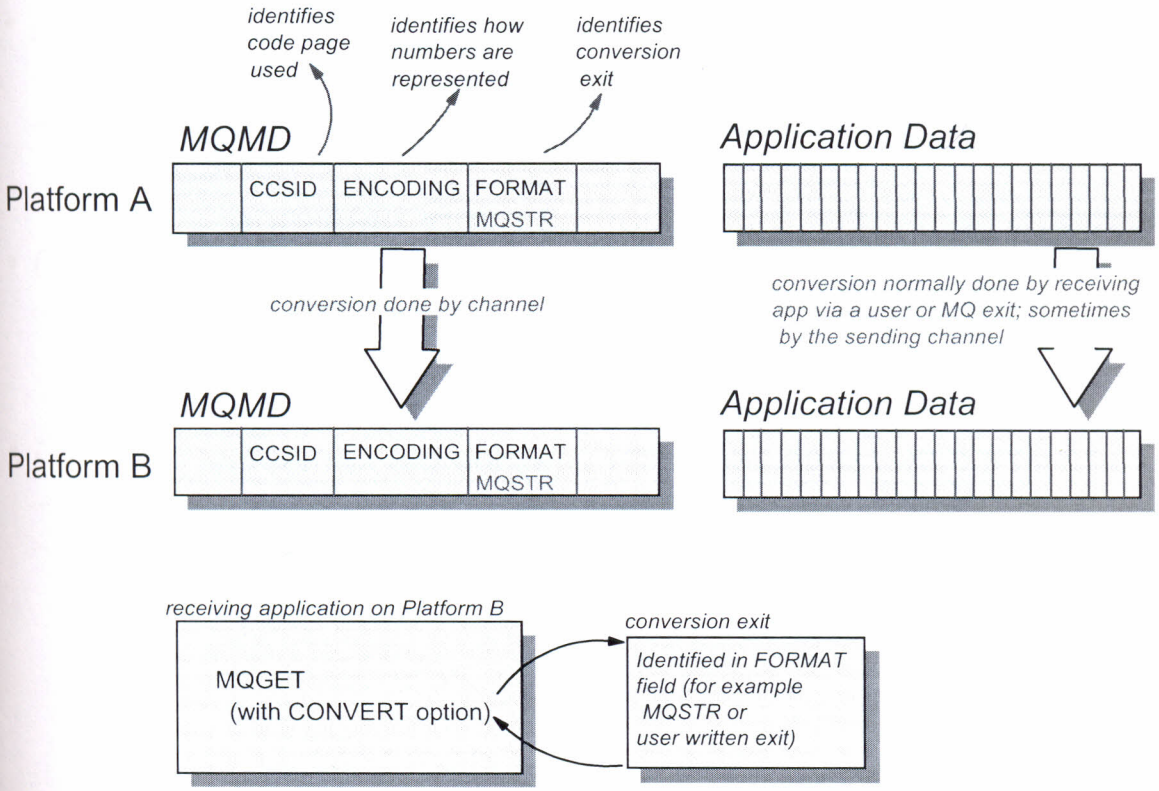


Figure 9-46. Data conversion

Notes:

Data representation

- Messages in a heterogeneous network
  - Character fields may need translation
  - Numeric fields may need transformation
- Message descriptor accompanies every message
  - Delivered to the receiving application
  - Always converted by WebSphere MQ
- Application data
  - Fields in the message descriptor describe the format and representation
  - Data conversion support is available

Figure 9-47. Data representation WM203 / VM2032.0

Notes:

When messages are being routed through a heterogeneous network of queue managers, there is a requirement to be able to handle different data representations.

- Some character fields may require translation from one character set to another.
- Some numeric fields may require transformation, such as byte reversal for integers.

Three fields in the message descriptor

- *Encoding*
  - Representation of the numeric data in the message
  - iSeries, Windows, and so forth
    - `MQENC_NATIVE`
- *CodedCharSetId*
  - Representation of the character data in the message
    - `MQCCSI_Q_MGR`
- *Format*
  - Indicates the nature of the data in the message
  - Values MQ . . . are reserved for WebSphere MQ

Notes:

There are three fields in the message descriptor which are used to support application data conversion. The first two fields specify the numeric and character representations of the application data. A message that is put with the initial values of these fields are correctly described. The *Format* field indicates the nature of the application data in a message.

Requesting application data conversion

- Request data conversion on MQGET call: `MQGMO_CONVERT`
  - *Encoding* and *CodedCharSetId*
    - On input, requested representation of the message
    - On output, what the application actually receives
  - Conversion performed, if necessary, on the basis of what is contained in the *Format* field
  - A warning, and the message returned in its original form, if the conversion cannot be done

OR

- Request data conversion at sending end of a message channel: `CONVERT(YES)`
  - Unconverted messages are put on the dead letter queue at the sending end

Figure 9-49. Requesting application data conversion WM203 / VM2032.0

Notes:

No application data conversion is performed by default; it must be requested. It might be requested by:

- Using the MQGMO_CONVERT option on the MQGET call. This option is the preferred approach on a queue manager that supports application data conversion.
- Specifying the parameter CONVERT(YES) on the channel definition at the sending end of a message channel. This option is a useful if the remote queue manager does not support application data conversion.

The preferred approach can be described as **receiver makes good**. The optimal situation is for the application data in a message to be converted only once.

What application data conversion can be done

- Some formats are built-in, and data conversion is performed by a built-in conversion routine
  - A message consisting entirely of characters
  - A message structure defined by WebSphere MQ
- User written data conversion exit is required when:
  - Format of a message is defined by application, not by WebSphere MQ
  - Message with a built-in format fails to convert
- There is an WebSphere MQ utility to help write a data conversion exit

Figure 9-50. What application data conversion can be done WM203 / VM2032.0

Notes:

The utility to help write a conversion exit is `crtmqcvx`.

What applications should do

- Put messages with the following values in the *Encoding* and *CodedCharSetId* fields:
  - `MQENC_NATIVE` for native encoding
  - `MQCCSI_Q_MGR` for the same CCSID as the queue manager
- Put all messages with a format name:
  - `MQFMT_STRING` for a message consisting entirely of characters
- Use the `MQGMO_CONVERT` option on the MQGET call to check what is delivered by the call
- If necessary, use `CONVERT(YES)` at the sending end of a message channel

Figure 9-51. What applications should do WM203 / VM2032.0

Notes:

Applications should not manipulate the CodedCharSetId field; instead, allow the queue manager to use its own defaults.

- Put every message with the following values in the *Encoding* and *CodedCharSetId* fields of the message descriptor.
  - MQENC_NATIVE, for native encoding.
  - MQCCSI_Q_MGR, for the same CCSID as the queue manager.
- Put every message with a format name in the *Format* field of the message descriptor, for example,
  - MQFMT_STRING, for a message consisting entirely of characters.
- Use the MQGMO_CONVERT option on the MQGET call. Check what is delivered by the MQGET call; the message may not have been converted.

  And, for completeness, one action for the administrator.

- If messages are being sent to a queue manager which does not support application data conversion, use the CONVERT(YES) option at the sending end of a message channel.

Successful conversion clearly depends on the sender of a message correctly setting the *Encoding*, *CodedCharSetId*, and Format fields in the message descriptor. Application programmers should get into the habit of correctly setting the *Encoding*, *CodedCharSetId*, and Format fields in the message descriptor even if currently the destination of the message is such that no conversion is necessary.

Checkpoint questions

1. What are six types of message channels?
2. True or false: A transmission queue is required for every remotely connected queue manager.
3. What action prompts the channel initiator to start a sender channel?
4. True or false: A common naming convention for a transmission queue is to use the same name as the targeted remote queue manager.
5. What WebSphere MQ control command shows the current state of a channel?
6. What are the methods used to access a remote queue?

Notes: