# Unit 6. Implementing distributed queuing

## Estimated time

01:30

## Overview

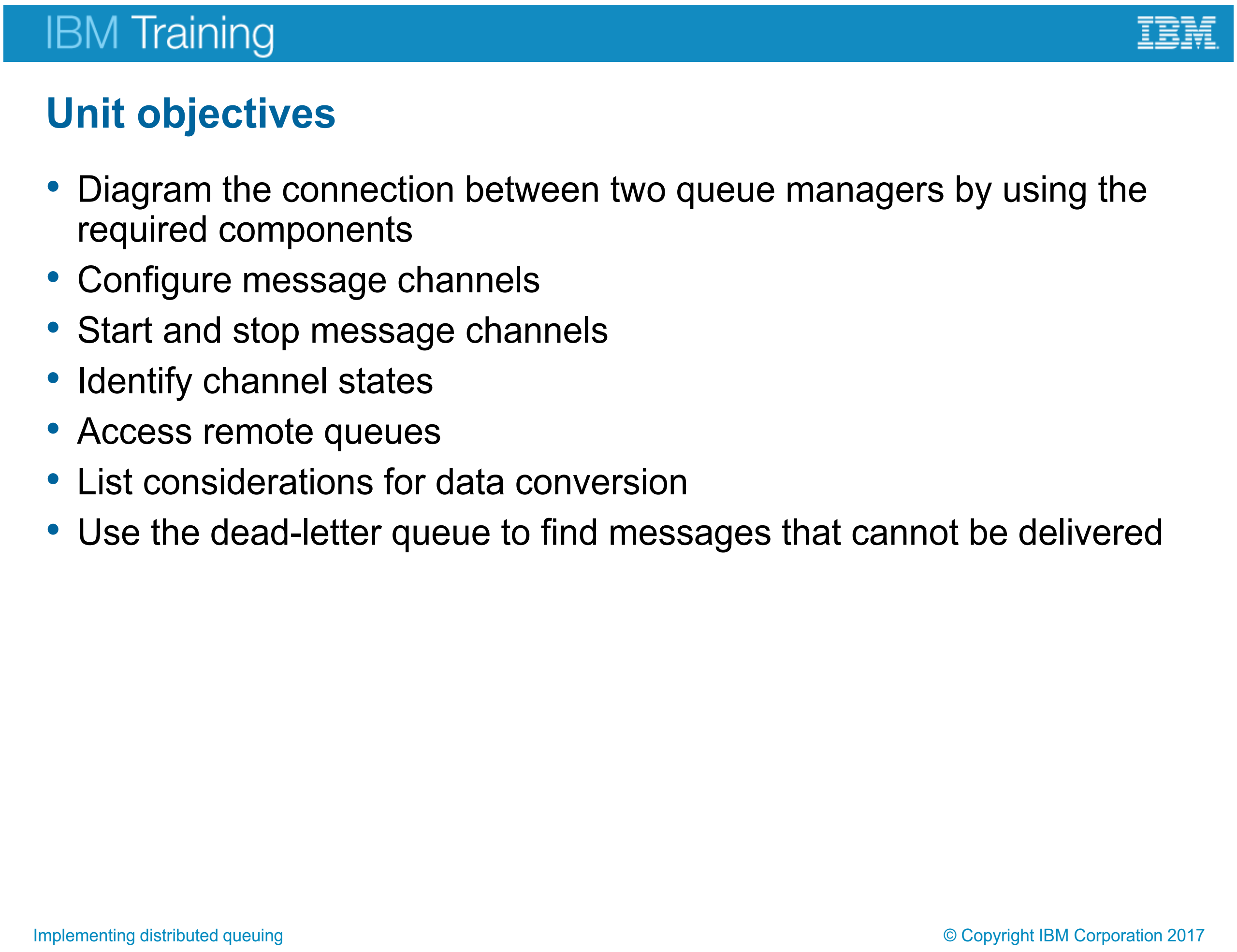
In this unit, you learn how to interconnect two or more queue managers in a network, allowing the distribution of messages between systems. You also learn about message channels, the supporting objects that are needed for channels to function correctly, and remote queue definitions.

## How you will check your progress

- Review questions
- Hands-on exercises

## References

IBM Knowledge Center for IBM MQ V9



IBM Training

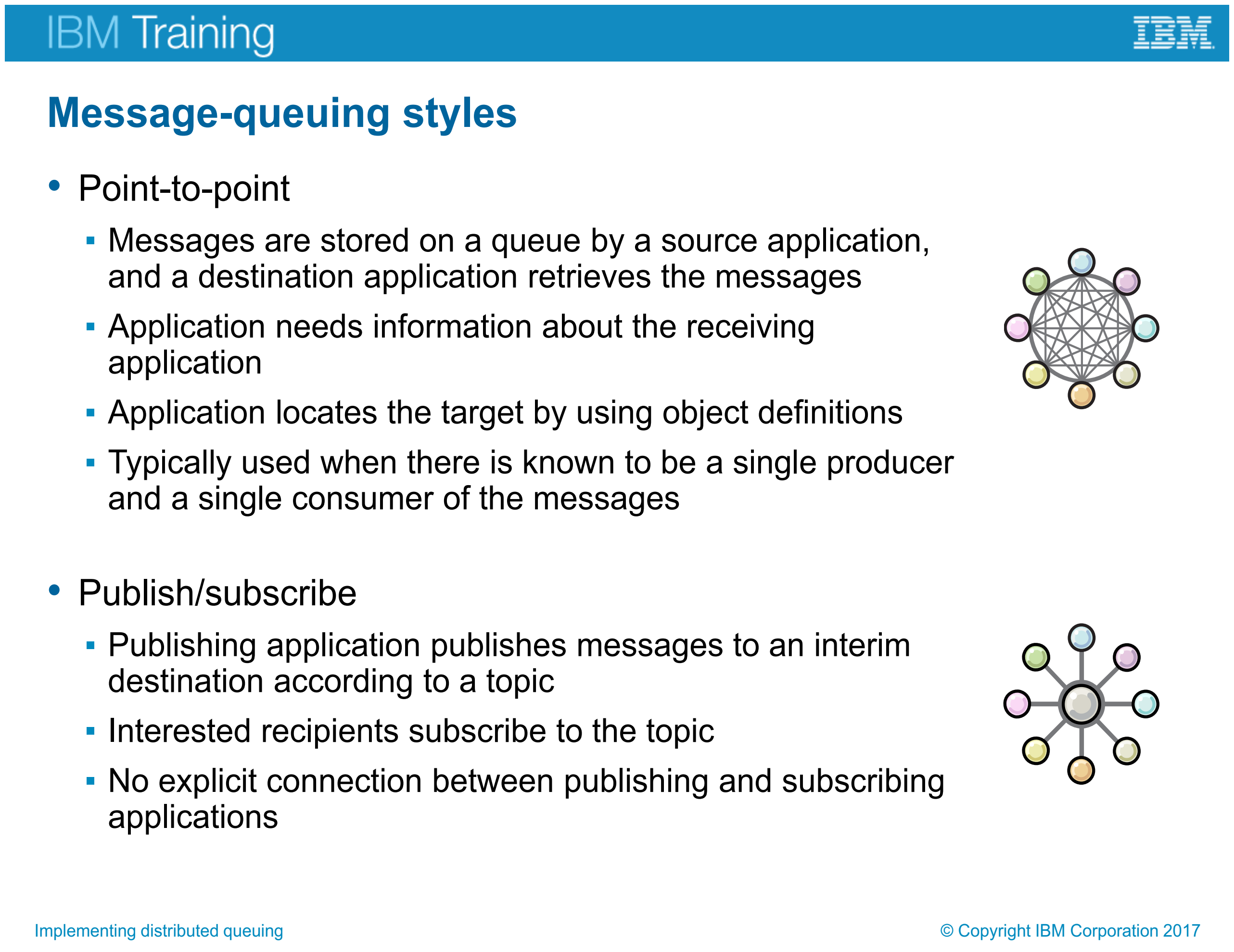
## Unit objectives

- Diagram the connection between two queue managers by using the required components
- Configure message channels
- Start and stop message channels
- Identify channel states
- Access remote queues
- List considerations for data conversion
- Use the dead-letter queue to find messages that cannot be delivered

Implementing distributed queuing

© Copyright IBM Corporation 2017

*Figure 6-1. Unit objectives*



IBM Training

## Message-queuing styles

- Point-to-point
  - Messages are stored on a queue by a source application, and a destination application retrieves the messages
  - Application needs information about the receiving application
  - Application locates the target by using object definitions
  - Typically used when there is known to be a single producer and a single consumer of the messages
- Publish/subscribe
  - Publishing application publishes messages to an interim destination according to a topic
  - Interested recipients subscribe to the topic
  - No explicit connection between publishing and subscribing applications

Implementing distributed queuing

© Copyright IBM Corporation 2017

*Figure 6-2. Message-queuing styles*

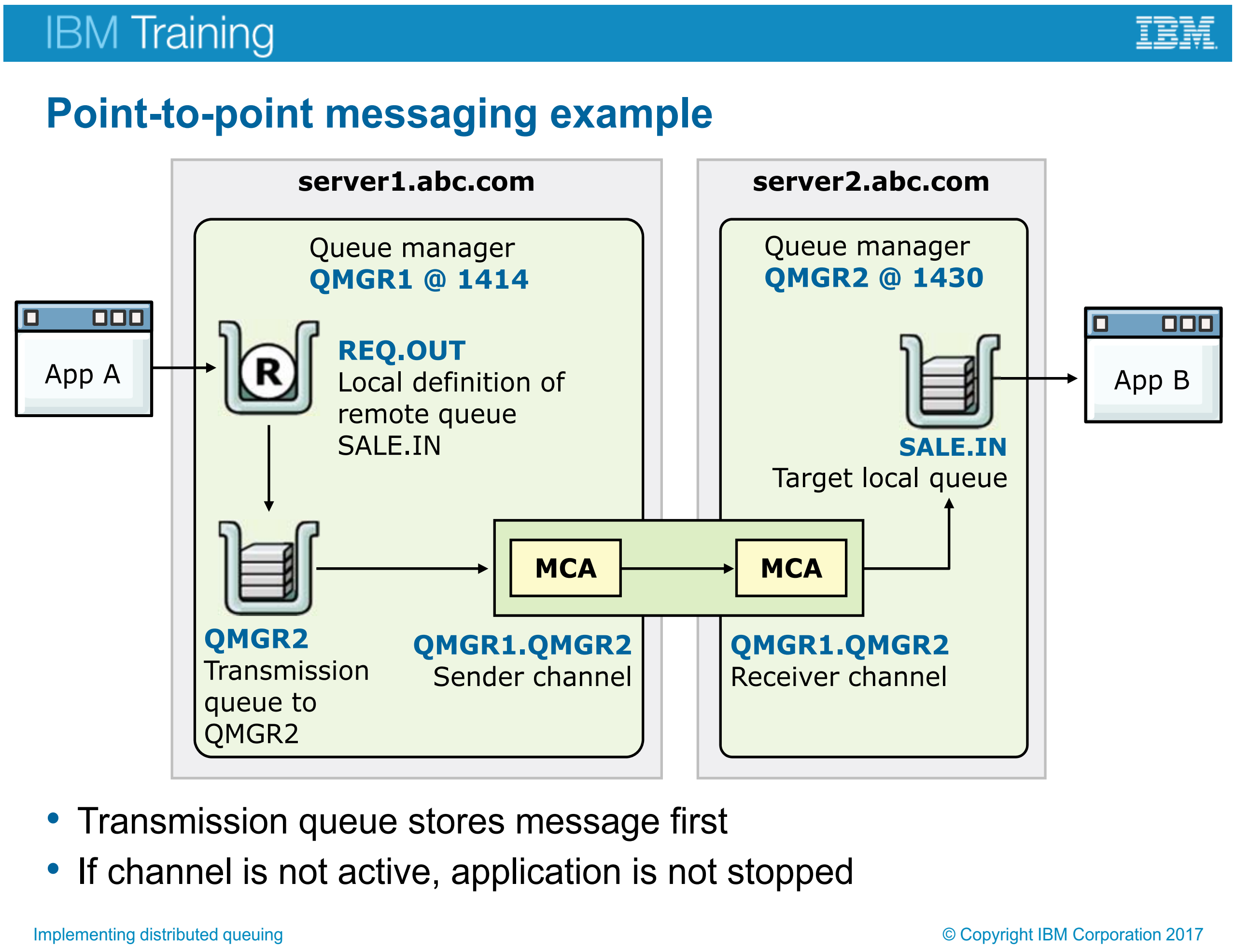
MQ supports two message-queuing styles: point-to-point and publish/subscribe.

In point-to-point messaging, a sending application must know information about the receiving application before it can send a message to that application. For example, the sending application might need to know the name of the queue to which to send the information, and might also specify a queue manager name. The point-to-point messaging style requires connections across all queue managers that need to get or put the message.

In publish/subscribe messaging, a copy of each message that is published by a publishing application is delivered to every interested application. There might be many, one, or no interested applications. In publish/subscribe, an interested application is known as a subscriber and the messages are queued on a queue that is identified by a subscription.

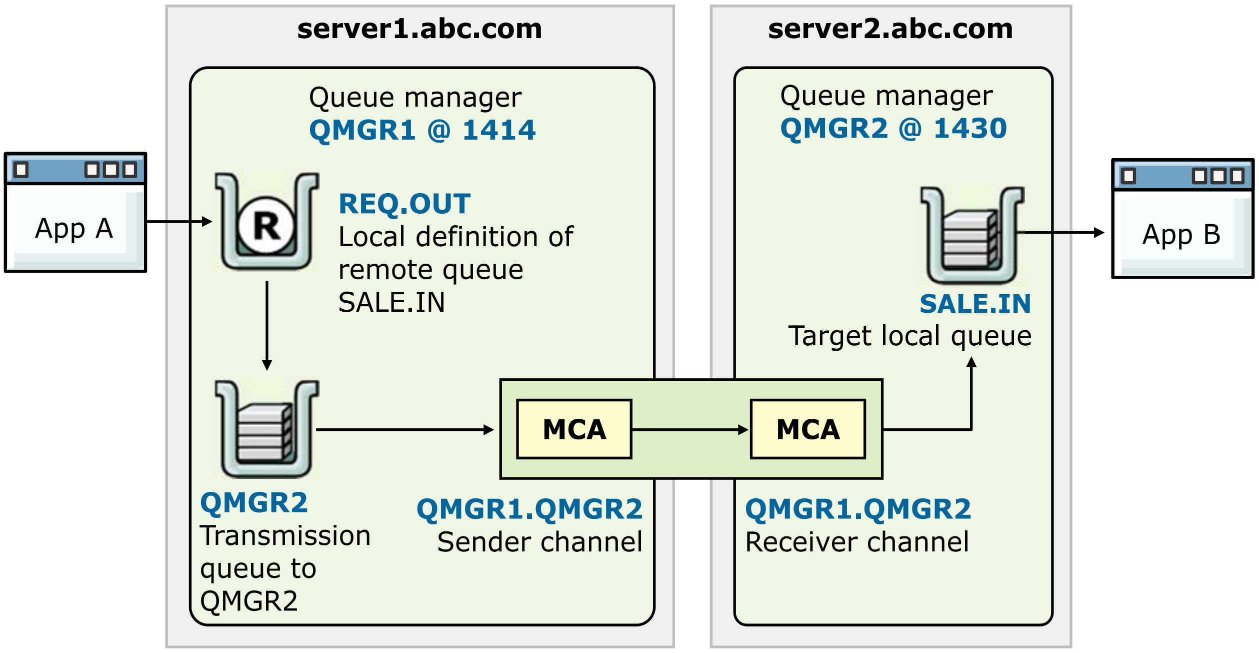
With publish/subscribe messaging, you can decouple the provider of information from the consumers of that information. The sending application and receiving application do not need to know as much about each other for the information to be sent and received.

In this unit, you learn about the point-to-point messaging style.



IBM Training

## Point-to-point messaging example



- Transmission queue stores message first
- If channel is not active, application is not stopped

Implementing distributed queuing © Copyright IBM Corporation 2017

*Figure 6-3. Point-to-point messaging example*

This figure reviews the MQ components for point-to-point messaging.

In the example, the sending and receiving applications are on different servers and connect to different queue managers.

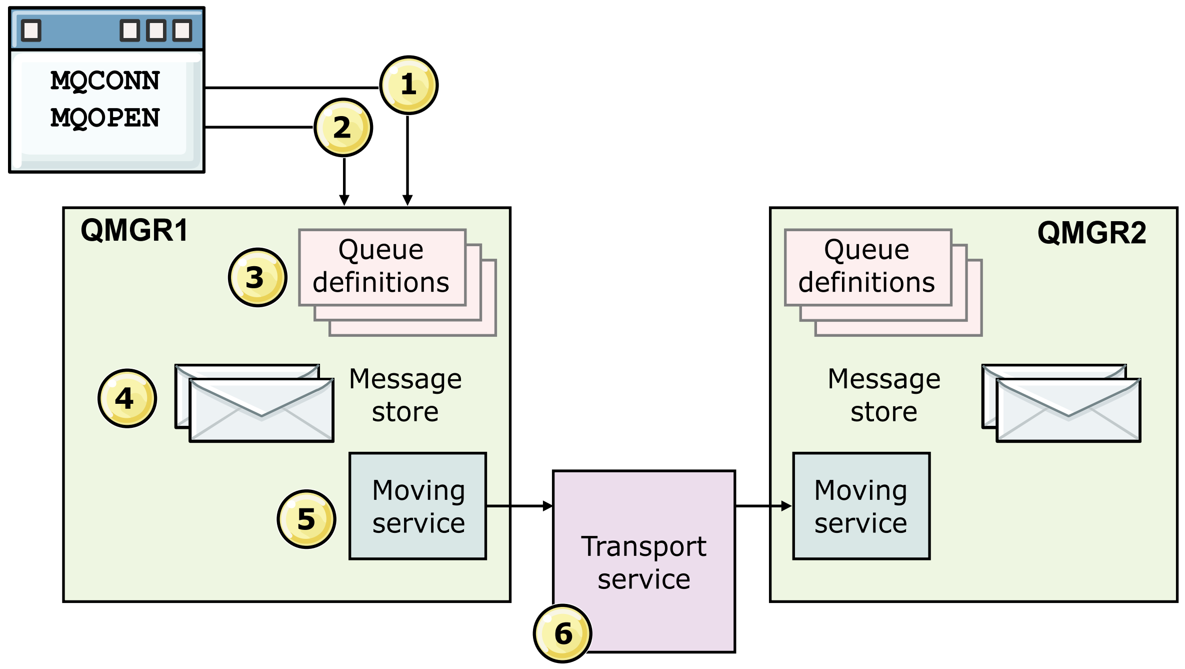
In this example, App A puts a message on REQ.OUT, which is a local definition of the remote queue SALE.IN on QMGR2 on server **server1.abc.com**. The queue manager to which App A is connected, QMGR1, knows that REQ.OUT points to a queue on QMGR2 on server **server2.abc.com** and puts the message on the transmission queue for QMGR2. Asynchronously, queue manager QMGR1 transmits the message to queue manager QMGR2 over a sender channel. QMGR2 receives the message over the compatible receiver channel and puts it on the target queue SALE.IN. The message then becomes available for App B to get.

This implementation provides a level of decoupling because if necessary, you can change REQ.OUT to point to another queue manager and another queue without impacting the application.

When an application opens a queue, it is the queue manager to which the application is connected that recognizes whether the queue is local or remote. If it is a remote queue, the queue manager stores messages that are destined for that queue on a staging queue that is called a transmission queue. By using a transmission queue, the application that puts the messages can continue to operate even if the communications link is down. Like any other queue, a transmission queue stores messages securely until the messages can be processed.

IBM Training

## Distributed queuing components overview (1 of 2)



Implementing distributed queuing

*Figure 6-4. Distributed queuing components overview (1 of 2)*

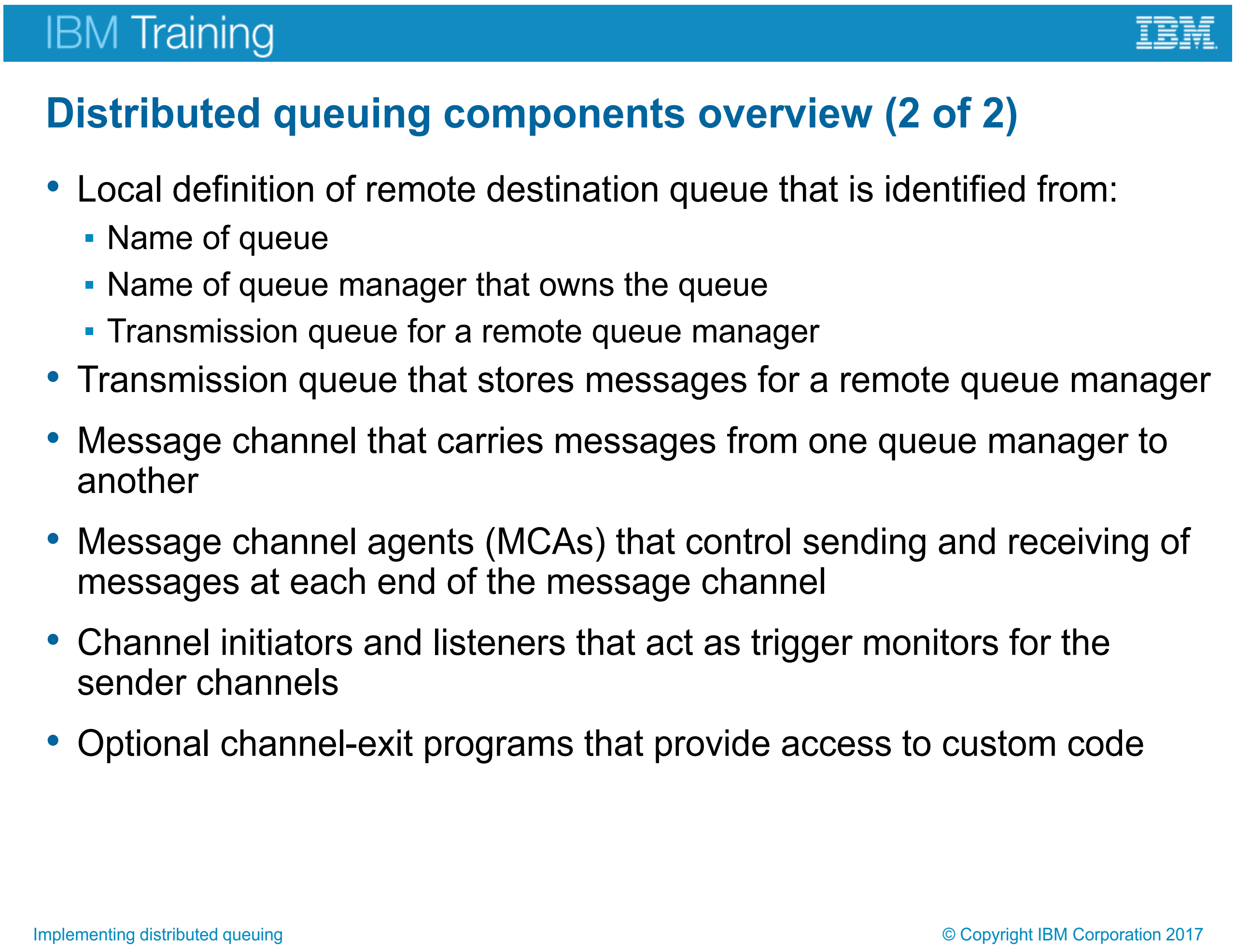
The figure shows the high-level components of distributed queuing.

In the figure, an MQ queue manager (QMGR1) is connected to another queue manager (QMGR2) so that resources and messages can be shared. An application connects to a queue manager and opens a queue to transmit messages to another queue manager.

1. The application sends an MQI MQCONN call to connect to a queue manager.
2. The application sends an MQI MQOPEN call to open a queue so that it can put messages on it.
3. The queue manager has a definition for each of its queues, specifying information such as the maximum number of messages that are 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 might not be apparent to the application.
5. Each queue manager contains communication software that is called the *moving service* component by which the queue manager can communicate with other queue managers.
6. The *transport service* connects to the target queue manager and opens a queue to transmit the messages.

The transport service is independent of the queue manager and can be any one of the following methods (depending on the operating system):

- Transmission Control Protocol/Internet Protocol (TCP/IP)
- Network Basic Input/Output System (NetBIOS)
- Sequenced Packet Exchange (SPX)
- Systems Network Architecture (SNA)



IBM Training

## Distributed queuing components overview (2 of 2)

- Local definition of remote destination queue that is identified from:
  - Name of queue
  - Name of queue manager that owns the queue
  - Transmission queue for a remote queue manager
- Transmission queue that stores messages for a remote queue manager
- Message channel that carries messages from one queue manager to another
- Message channel agents (MCAs) that control sending and receiving of messages at each end of the message channel
- Channel initiators and listeners that act as trigger monitors for the sender channels
- Optional channel-exit programs that provide access to custom code

Implementing distributed queuing © Copyright IBM Corporation 2017

*Figure 6-5. Distributed queuing components overview (2 of 2)*

A *transmission queue* is a special type of local queue that stores messages before the MCA transmits them to the remote queue manager. In a distributed-queuing environment, define one transmission queue for each sending MCA, unless you are using 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. Optionally, 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.

A *message channel* carries messages from one queue manager to another. 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 by using the message channel type that is defined at one end, and a compatible type at the other end. Possible combinations are sender-receiver, requester-server, requester-sender (callback), server-receiver, and cluster sender-cluster receiver.

Do not confuse message channels with MQI channels. Message channels connect MQ queue managers. MQI channels connect MQ clients and MQ servers.

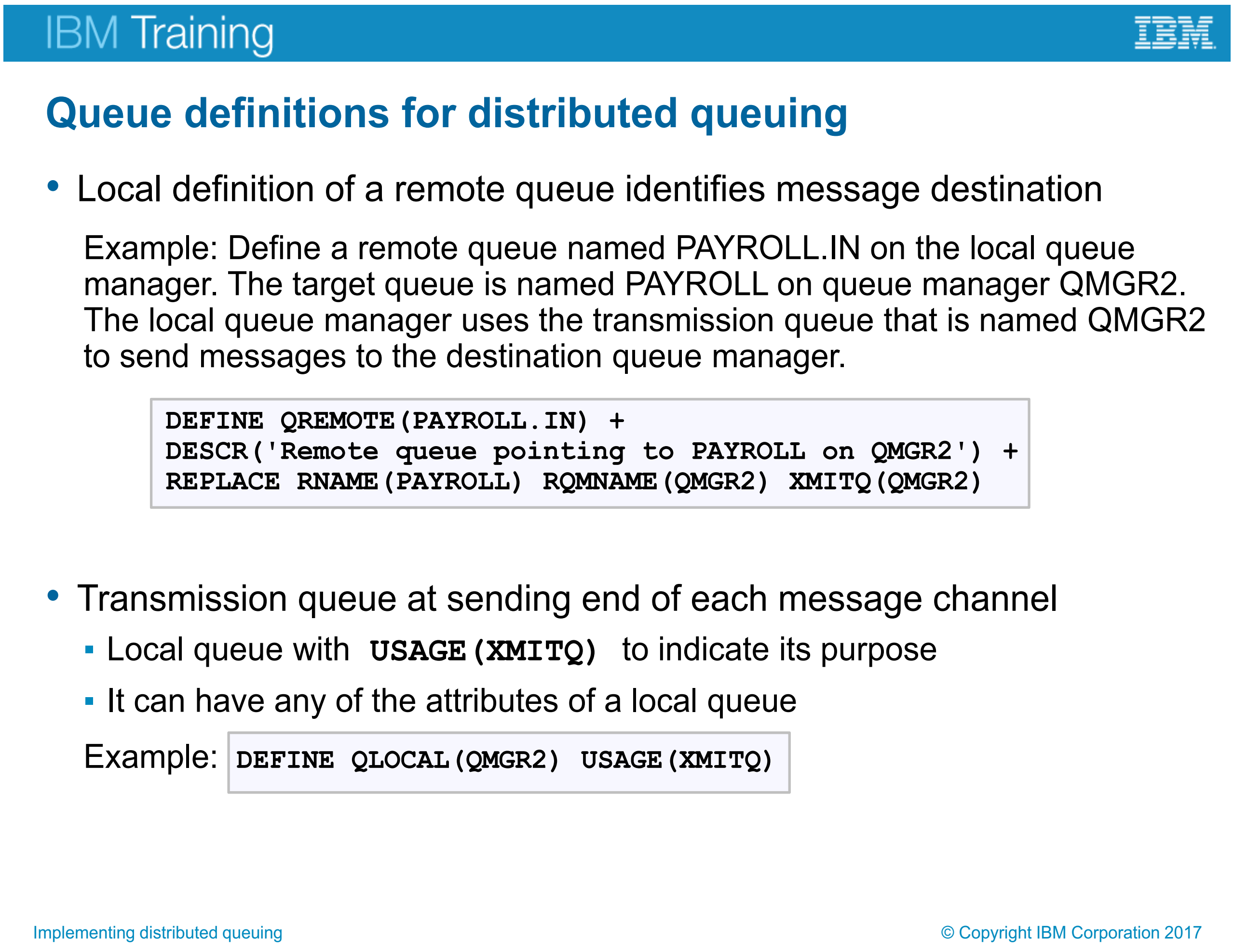
The *message channel agent* (MCA) controls sending and receiving of messages. One MCA takes messages from the transmission queue and puts them on the communication link. The other MCA receives messages and delivers them onto a queue on the remote queue manager.

An MCA is a *caller MCA* if it initiated the communication; otherwise, it is 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.

A *channel initiator* acts as a *trigger monitor* for sender channels because a transmission queue might 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.

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

MQ calls *channel-exit programs* at defined places in the processing carried out by the MCA.



IBM Training

## Queue definitions for distributed queuing

- Local definition of a remote queue identifies message destination

  Example: Define a remote queue named PAYROLL.IN on the local queue manager. The target queue is named PAYROLL on queue manager QMGR2. The local queue manager uses the transmission queue that is named QMGR2 to send messages to the destination queue manager.

```
DEFINE QREMOTE(PAYROLL.IN) +
DESCR('Remote queue pointing to PAYROLL on QMGR2') +
REPLACE RNAME(PAYROLL) RQMNAME(QMGR2) XMITQ(QMGR2)
```

- Transmission queue at sending end of each message channel
  - Local queue with **`USAGE(XMITQ)`** to indicate its purpose
  - It can have any of the attributes of a local queue

  Example: `DEFINE QLOCAL(QMGR2) USAGE(XMITQ)`

Implementing distributed queuing © Copyright IBM Corporation 2017

*Figure 6-6. Queue definitions for distributed queuing*

Two special queues are required for distributed queuing: a transmission queue and a local definition of a remote queue.

The transmission queue is a local queue that is used when a queue manager forwards messages to a remote queue manager through a message channel. For clarity, give a transmission queue the same name as the remote queue manager.

It is also necessary to create a transmission queue at the sending end of each message channel.

You can define a transmission queue by using the MQSC command **`DEFINE QLOCAL`** with the **`USAGE(XMITQ)`** attribute or by using MQ Explorer.

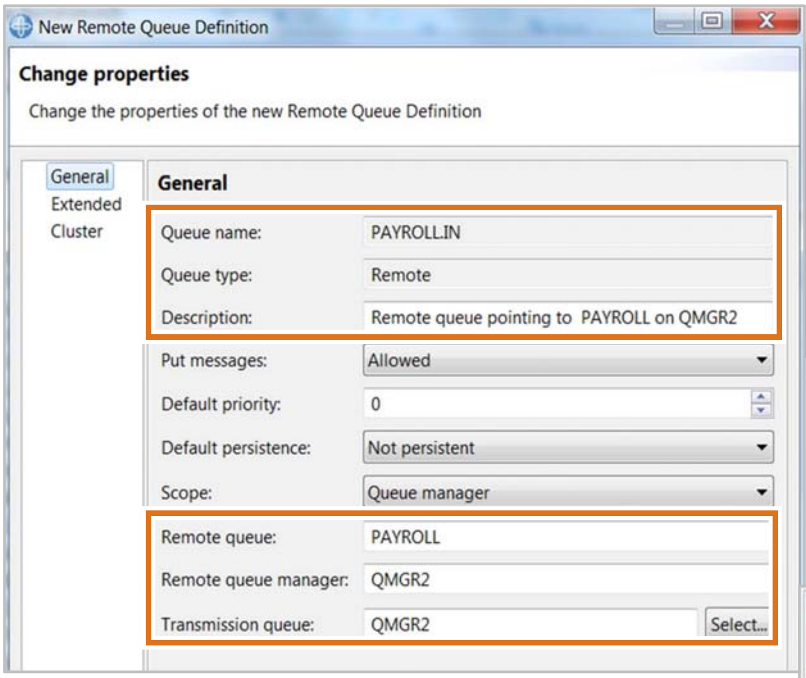
Applications can retrieve messages only from local queues. They can put messages on local queues or remote queues. Thus, a queue manager might have remote queue definitions and a definition for each of its local queues. Remote queue definitions are definitions for queues that another queue manager owns.

The advantage of remote queue definitions is that they allow an application to put a message to a remote queue without specifying the name of the remote queue, the remote queue manager, or the transmission queue. Remote queues give location independence.

The location of a remote queue can be hidden from an application by using the MQSC command **`DEFINE QREMOTE`** to create a local definition of a remote queue. You can also use MQ Explorer to create a local definition of a remote queue. The **XMITQ** attribute identifies the transmission queue.

IBM Training

## Using IBM MQ Explorer to define a remote queue



New Remote Queue Definition

**Change properties**

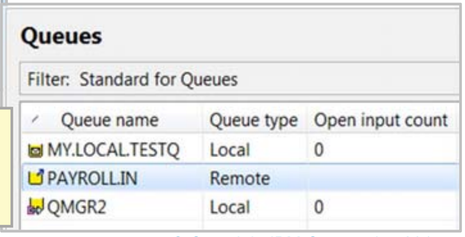
Change the properties of the new Remote Queue Definition

General
Extended
Cluster

**General**

| | |
|---|---|
| Queue name: | PAYROLL.IN |
| Queue type: | Remote |
| Description: | Remote queue pointing to PAYROLL on QMGR2 |
| Put messages: | Allowed |
| Default priority: | 0 |
| Default persistence: | Not persistent |
| Scope: | Queue manager |
| Remote queue: | PAYROLL |
| Remote queue manager: | QMGR2 |
| Transmission queue: | QMGR2 |

1. Right-click **Queues**, and then click **New > Remote Queue Definition**
2. Enter a name for local definition of remote queue
3. Specify remote queue definition properties



**Queues**

Filter: Standard for Queues

| Queue name | Queue type | Open input count |
|---|---|---|
| MY.LOCAL.TESTQ | Local | 0 |
| PAYROLL.IN | Remote | |
| QMGR2 | Local | 0 |

Icon and **Queue type** identifies remote queue definition in **Queues** content view

Implementing distributed queuing © Copyright IBM Corporation 2017

*Figure 6-7. Using IBM MQ Explorer to define a remote queue*

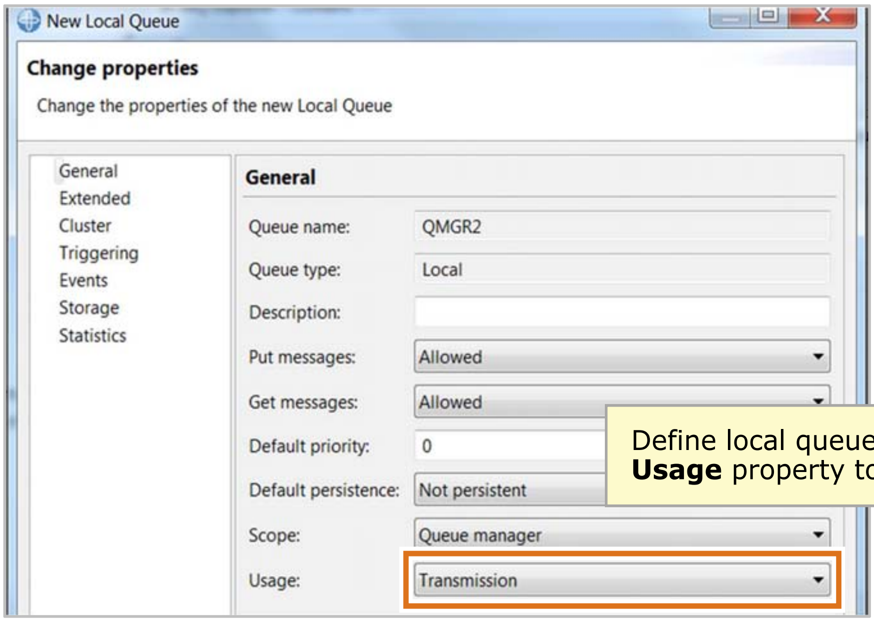
You can use MQ Explorer to define a local definition of a remote queue.

This figure lists the steps for creating a local definition of a remote queue. The properties are consistent with the attributes that you would use if you created the local definition of the remote queue by using the MQSC **`DEFINE QREMOTE`** command.

A local definition of a remote queue is identified by the **Queue type** of **Remote** in the MQ Explorer **Queues** view.

IBM Training

## Using IBM MQ Explorer to define a transmission queue



New Local Queue

**Change properties**

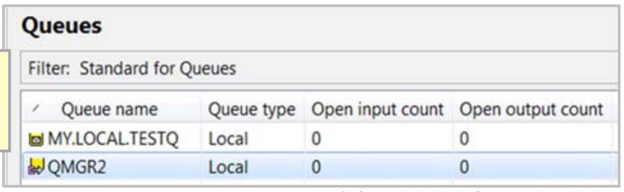
Change the properties of the new Local Queue

General
Extended
Cluster
Triggering
Events
Storage
Statistics

**General**

| | |
|---|---|
| Queue name: | QMGR2 |
| Queue type: | Local |
| Description: | |
| Put messages: | Allowed |
| Get messages: | Allowed |
| Default priority: | 0 |
| Default persistence: | Not persistent |
| Scope: | Queue manager |
| Usage: | Transmission |

Define local queue and set **General > Usage** property to **Transmission**



**Queues**

Filter: Standard for Queues

| Queue name | Queue type | Open input count | Open output count |
|---|---|---|---|
| MY.LOCAL.TESTQ | Local | 0 | 0 |
| QMGR2 | Local | 0 | 0 |

Icon identifies transmission queue in **Queues** content view

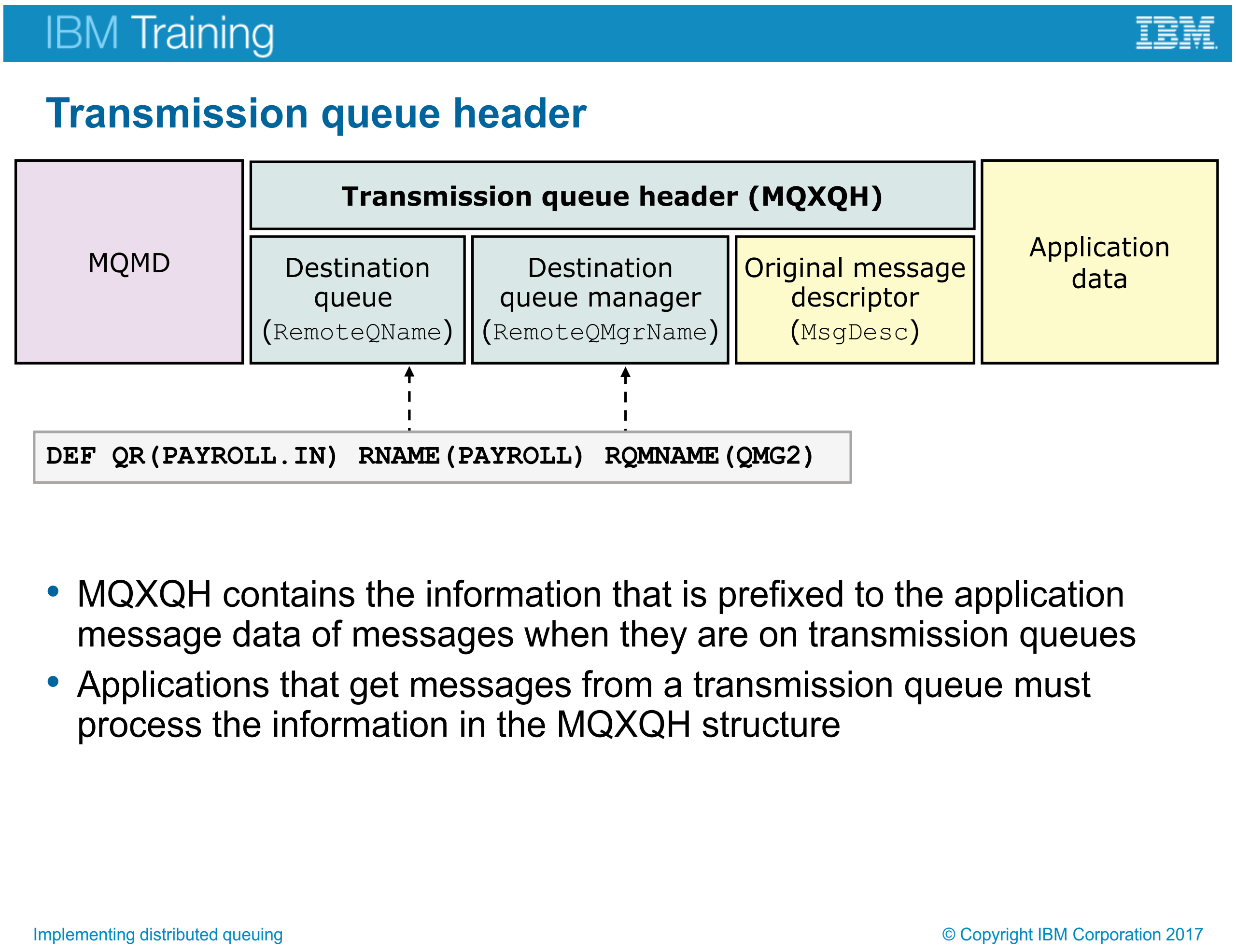
Implementing distributed queuing

© Copyright IBM Corporation 2017

*Figure 6-8. Using IBM MQ Explorer to define a transmission queue*

You can use MQ Explorer to define a transmission by changing the **Usage** property to **Transmission** when you create or modify a local queue.

The icon next to the queue name in the **Queues** view in MQ Explorer helps you to distinguish between a local queue and a transmission queue. In the example in the figure, QMGR2 is a transmission queue; MY.LOCAL.TESTQ is a local queue.


IBM Training
MQMD
Transmission queue header (MQXQH)
Destination queue (RemoteQName)
Destination queue manager (RemoteQMgrName)
Original message descriptor (MsgDesc)
Application data

## Transmission queue header

```
DEF QR(PAYROLL.IN) RNAME(PAYROLL) RQMNAME(QMG2)
```

- MQXQH contains the information that is prefixed to the application message data of messages when they are on transmission queues
- Applications that get messages from a transmission queue must process the information in the MQXQH structure

Implementing distributed queuing © Copyright IBM Corporation 2017

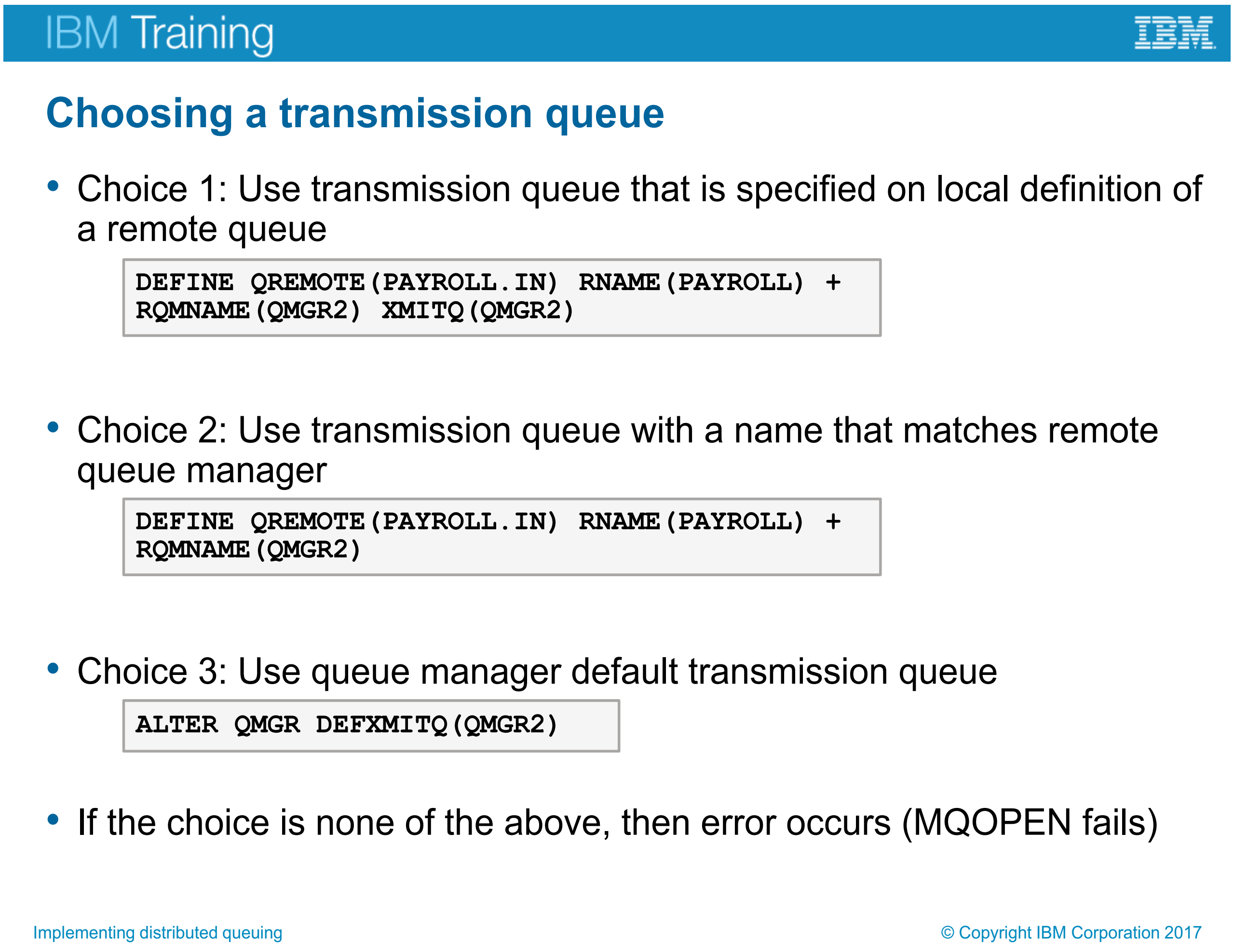
*Figure 6-9. Transmission queue header*

The MQ transmission queue header (MQXQH) structure describes the information that is prefixed to the application message data of messages when they are on transmission queues.

Data in the MQXQH must be in the character set and encoding of the local queue manager.

In addition to a structure identifier and version number, the MQXQH also contains the name of the destination queue, the name of the destination queue manager, and the original message descriptor.

The queue manager generates the message descriptor in the header when the message is placed on the transmission queue. Some of the fields in the separate message descriptor are copied from the message descriptor that the application provides on the MQPUT or MQPUT1 call. The separate message descriptor is the one that is returned to the application when the message is removed from the transmission queue.



IBM Training

## Choosing a transmission queue

- Choice 1: Use transmission queue that is specified on local definition of a remote queue

```
DEFINE QREMOTE(PAYROLL.IN) RNAME(PAYROLL) +
RQMNAME(QMGR2) XMITQ(QMGR2)
```

- Choice 2: Use transmission queue with a name that matches remote queue manager

```
DEFINE QREMOTE(PAYROLL.IN) RNAME(PAYROLL) +
RQMNAME(QMGR2)
```

- Choice 3: Use queue manager default transmission queue

```
ALTER QMGR DEFXMITQ(QMGR2)
```

- If the choice is none of the above, then error occurs (MQOPEN fails)

Implementing distributed queuing © Copyright IBM Corporation 2017

*Figure 6-10. Choosing a transmission queue*

A queue manager attempts to apply one of the following rules, in the stated order, to determine which transmission queue to use.

1. Use the transmission queue that is named explicitly in the **XMITQ** attribute in the remote queue definition.

   An example MQSC command that creates a remote queue with an explicit transmission queue that is named EXPRESS is:
   **DEF QR(BBB) RNAME(YYY) RQMNAME(QM2) XMITQ(EXPRESS)**

2. Use the transmission queue with the same name as the remote queue manager. In this case, the remote queue definition does not specify a transmission queue. For example, if the remote queue manager name is QMGR2, MQ assumes that the transmission queue is a transmission queue that is named QMGR2.

3. Use the queue manager default transmission queue. When you define a queue manager, you can specify a default transmission queue. The default transmission queue is used when no other suitable transmission queue is available for sending messages to another queue manager. If you define a default transmission queue, you must also define a channel to serve the queue. If you do not define a channel to serve the queue, messages that are put onto the default transmission queue are not transmitted to the remote queue manager and remain on the queue.

An example command for defining a default transmission queue is:
**`ALTER QMGR DEFXMITQ(HOST)`**

4. If the queue manager cannot find a transmission queue by attempting to apply rules 1 – 3, an error is reported.

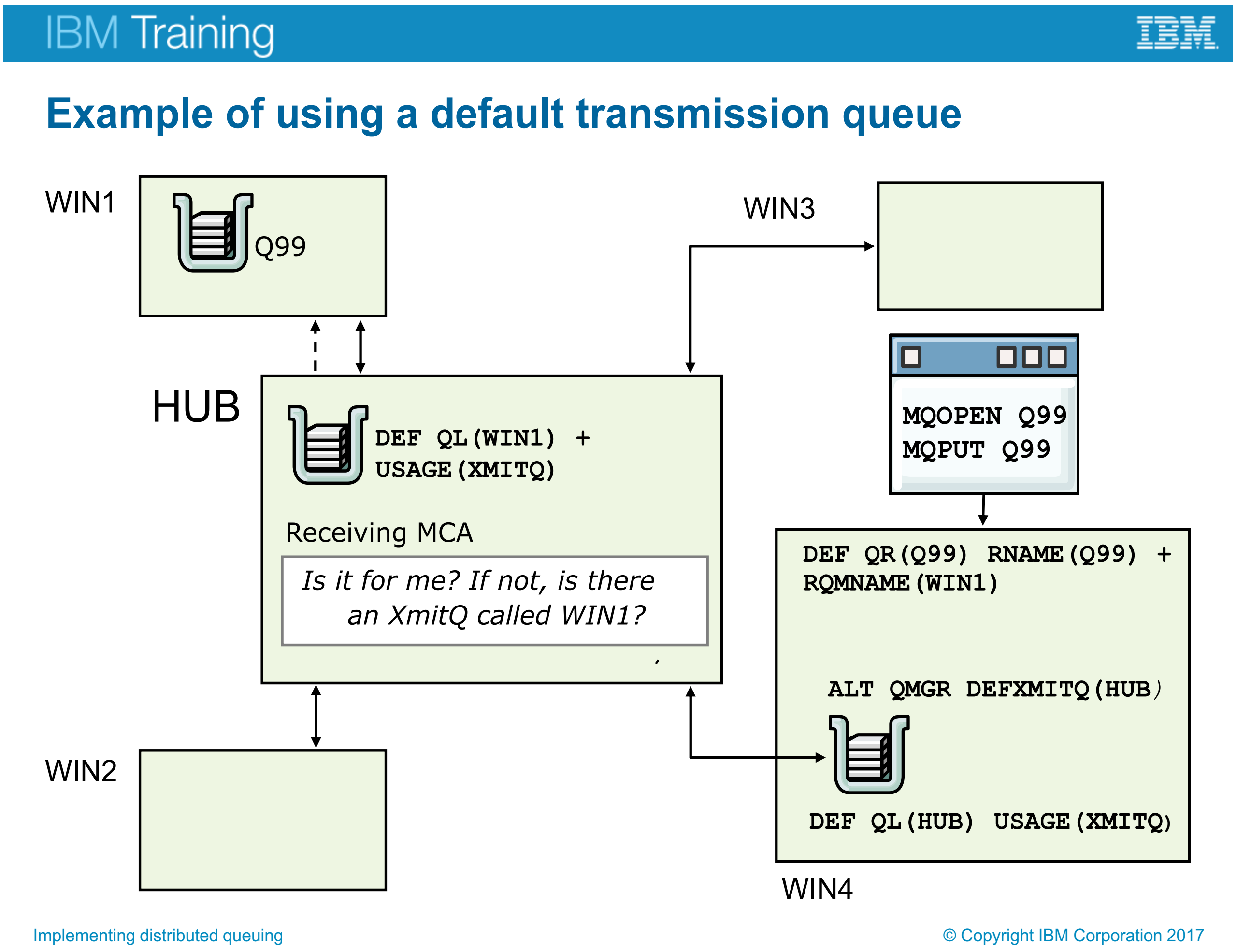


*Figure 6-11. Example of using a default transmission queue*

The default transmission queue is the destination for messages that are sent to a remote queue when no other suitable transmission queue is defined.

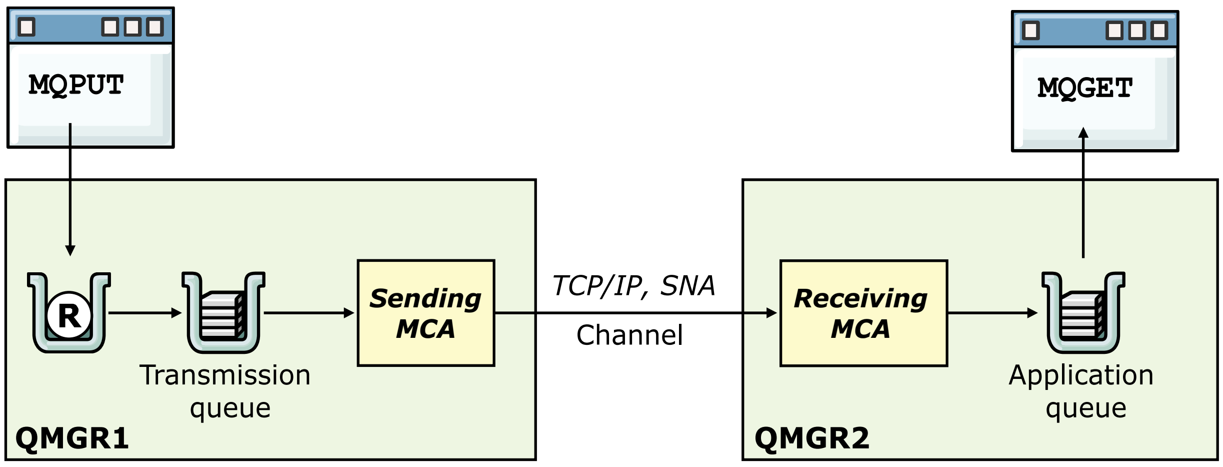
Default transmission queues are useful in a hub and spoke network environment. In a hub and spoke environment, each of the spoke queue managers is directly connected only to the hub queue manager. Defining the transmission queue to the hub as the default transmission queue for each spoke queue manager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

The figure shows an example of a hub and spoke design.

The default transmission queue is defined as a local queue with the **`USAGE(XMITQ)`** attribute. That queue name is then referenced in the queue manager **`DEFXMITQ`** attribute. In this design, the remote queue definition does not specify a transmission queue.

IBM Training

## Message channel



- One-way link that connects two queue managers by using an MCA
- Each end of a message channel has a separate definition

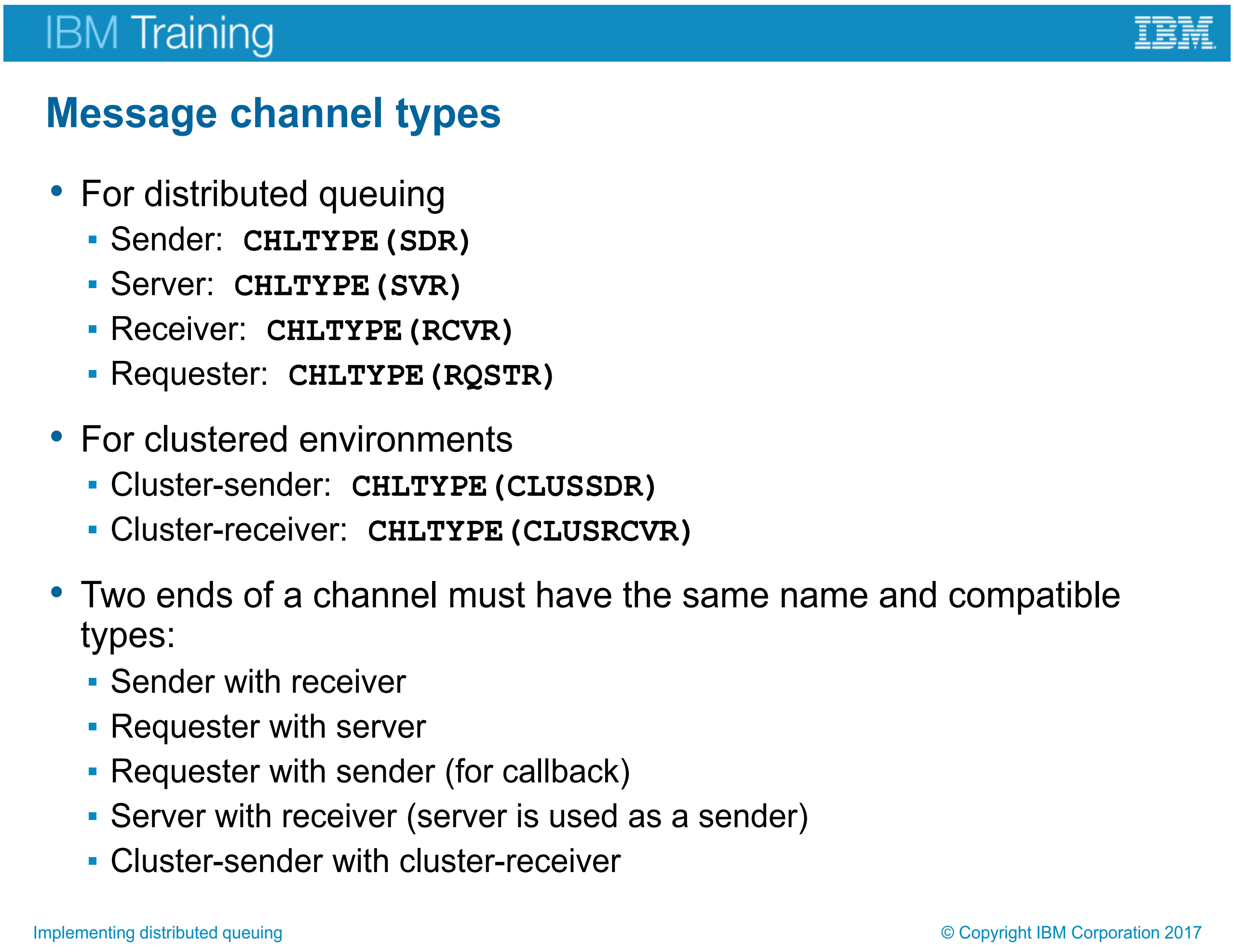
*Figure 6-12. Message channel*

In this unit, a message channel is a one-way communication link between two queue managers for the transmission of messages. As shown in the figure, a message channel consists of an MCA at the sending queue manager, an MCA at the receiving queue manager, and a communications connection between the two. The communications connection might be an SNA LU6.2 conversation or a TCP connection, for example.

Each end of a message channel has a separate definition. Both definitions contain the name of the message channel. Among other things, the definition at each end of a message channel also indicates:

- Whether it is the sending end or the receiving end of the channel
- The communications protocol to use

A transmission queue is required at the sending end of a message channel. So, only the definition of the message channel at the sending end contains the name of the transmission queue. One way to ensure that the message is sent to the correct destination is to give the transmission queue the same name as the name of the destination queue manager.



IBM Training

## Message channel types

- For distributed queuing
  - Sender: **`CHLTYPE(SDR)`**
  - Server: **`CHLTYPE(SVR)`**
  - Receiver: **`CHLTYPE(RCVR)`**
  - Requester: **`CHLTYPE(RQSTR)`**
- For clustered environments
  - Cluster-sender: **`CHLTYPE(CLUSSDR)`**
  - Cluster-receiver: **`CHLTYPE(CLUSRCVR)`**
- Two ends of a channel must have the same name and compatible types:
  - Sender with receiver
  - Requester with server
  - Requester with sender (for callback)
  - Server with receiver (server is used as a sender)
  - Cluster-sender with cluster-receiver

Implementing distributed queuing © Copyright IBM Corporation 2017

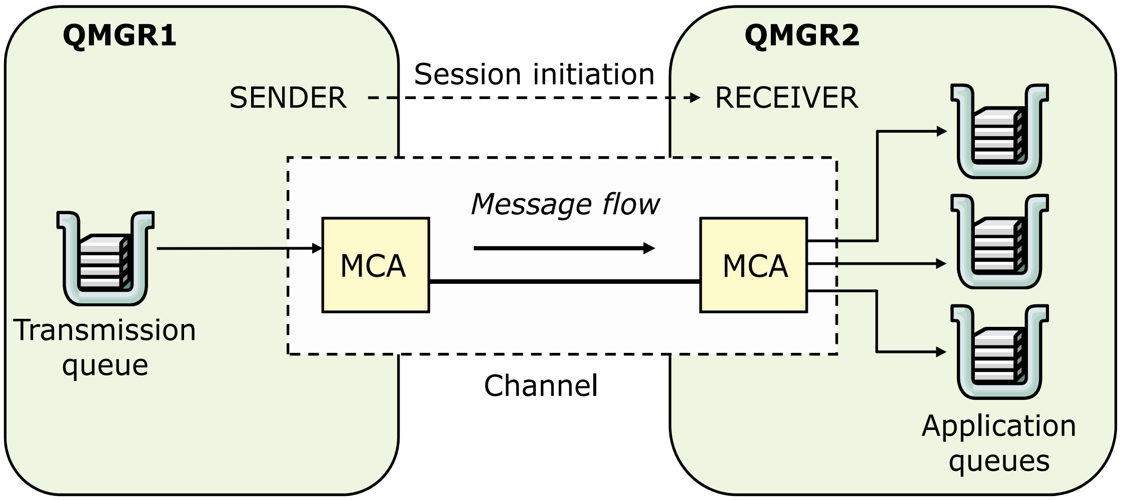
*Figure 6-13. Message channel types*

The figure lists the basic channel types for distributed queuing, clustered environments, and MQ clients.

You learn more the distributed queuing channel types in this unit. You learn more about the MQ client and cluster channels later in this course.

IBM Training

## Sender to receiver message channel



- Sender
  - Starts the channel
  - Requests the receiver at the other end of the channel to start
  - Sends messages from its transmission queue to the receiver
- Receiver puts the messages on the destination queue

Implementing distributed queuing © Copyright IBM Corporation 2017

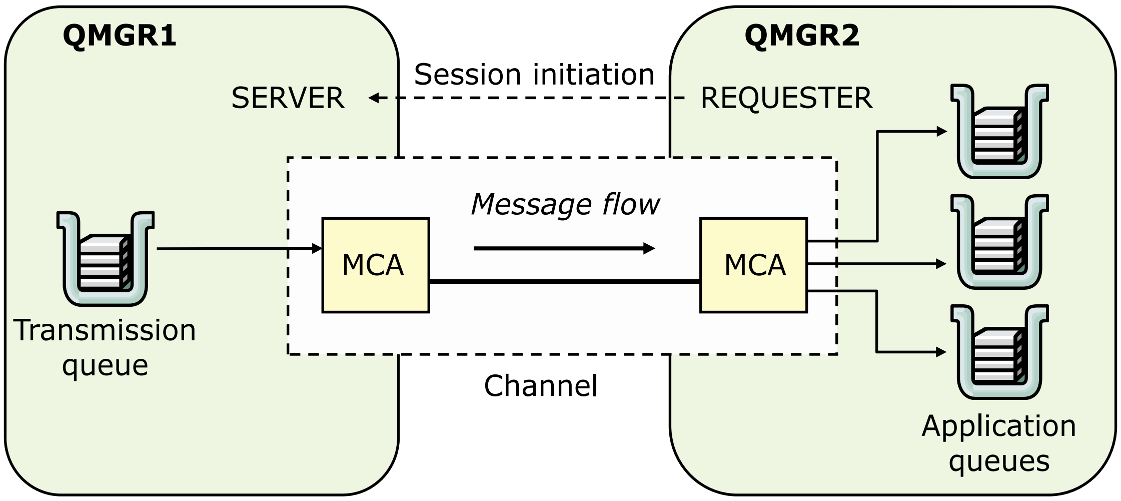
*Figure 6-14. Sender to receiver message channel*

With a *sender to receiver message channel*, a sender on one system starts the channel so that it can send messages to the receiver on 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.

A *server to receiver message channel* is similar to a sender to receiver channel except that it applies only to fully qualified servers. A fully qualified server has server channels with the connection name of the server that is specified in the channel definition. Channel startup must be initiated at the server end of the link.

IBM Training

## Requester to server message channel



- Requester
  - Starts the channel so that it can receive messages
  - Requests the server at the other end of the channel to start
- Server sends messages to the requester from the transmission queue defined in its channel definition

Implementing distributed queuing © Copyright IBM Corporation 2017

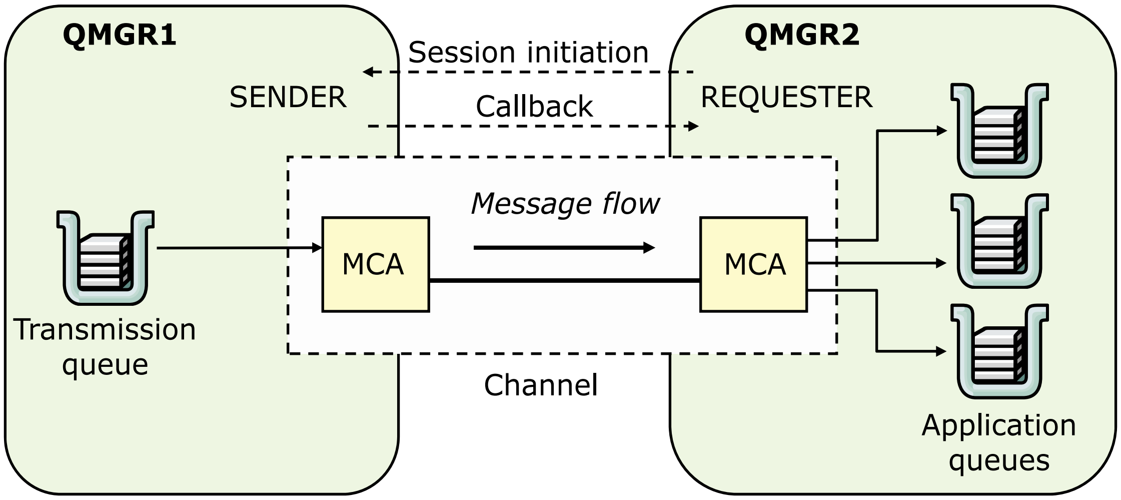
*Figure 6-15. Requester to server message channel*

With a requester to server message channel, a requester on 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 that is defined in its channel definition.

A server channel can also start the communication and send messages to a requester, but this initiation applies only to fully qualified servers. A requester can start a fully qualified server, or a fully qualified server can start a communication with a requester.

IBM Training

## Requester to sender message channel

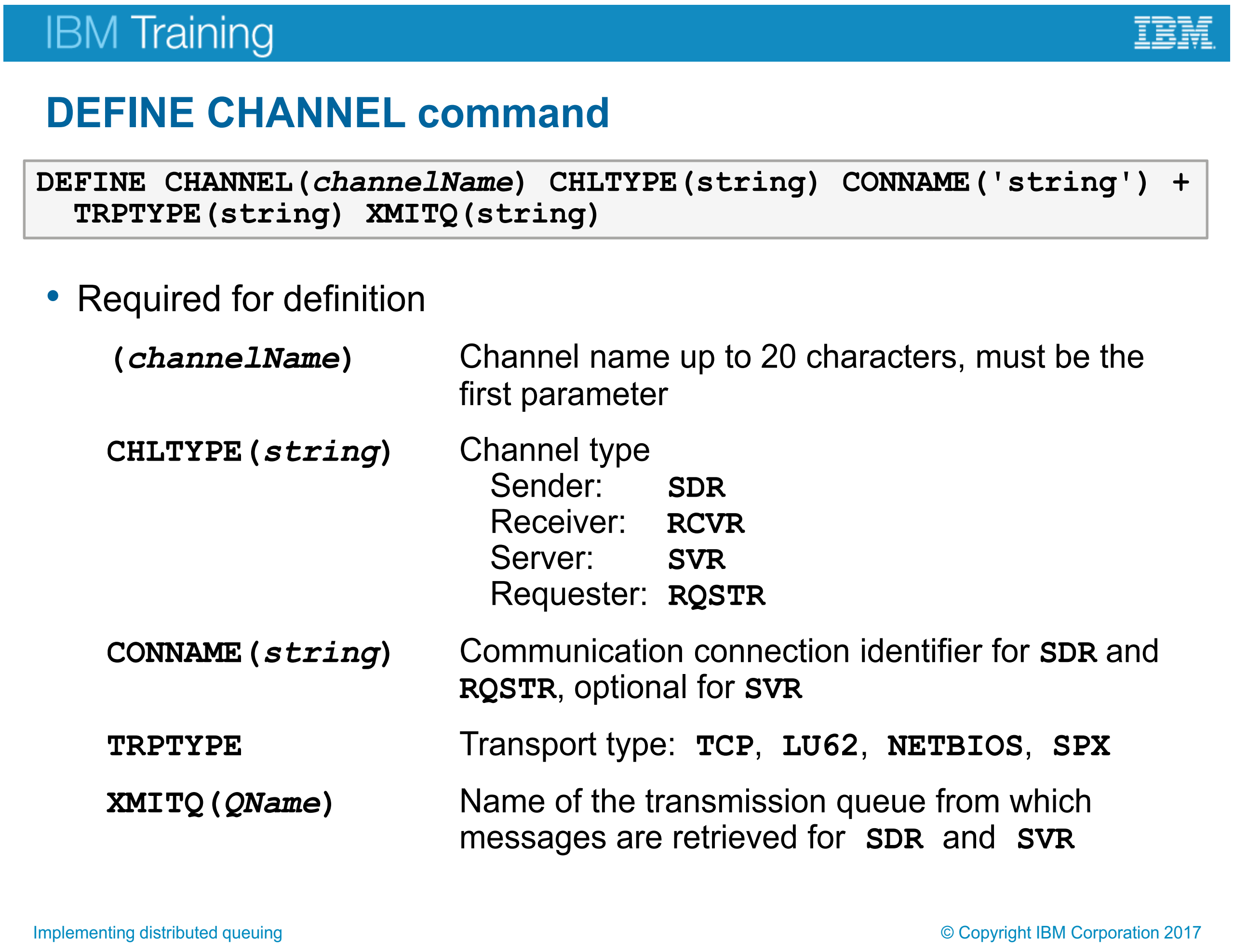


- Requester starts the channel
- Sender
  - Terminates the call
  - Restarts the communication according to information in its channel definition
  - Sends messages from the transmission queue to the requester

Implementing distributed queuing © Copyright IBM Corporation 2017

*Figure 6-16. Requester to sender message channel*

With a requester to sender message channel, the requester starts the channel and the sender ends the call. The sender then restarts the communication according to information in its channel definition (known as *call back*). It sends messages from the transmission queue to the destination queue of the requester.



IBM Training

## DEFINE CHANNEL command

```
DEFINE CHANNEL(channelName) CHLTYPE(string) CONNAME('string') +
  TRPTYPE(string) XMITQ(string)
```

- Required for definition

| | |
|---|---|
| **(*channelName*)** | Channel name up to 20 characters, must be the first parameter |
| **CHLTYPE(*string*)** | Channel type<br>Sender: **SDR**<br>Receiver: **RCVR**<br>Server: **SVR**<br>Requester: **RQSTR** |
| **CONNAME(*string*)** | Communication connection identifier for **SDR** and **RQSTR**, optional for **SVR** |
| **TRPTYPE** | Transport type: **TCP**, **LU62**, **NETBIOS**, **SPX** |
| **XMITQ(*QName*)** | Name of the transmission queue from which messages are retrieved for **SDR** and **SVR** |

Implementing distributed queuing

© Copyright IBM Corporation 2017

*Figure 6-17. DEFINE CHANNEL command*

The MQSC command to define a message channel is **DEFINE CHANNEL**. Related commands are **ALTER CHANNEL**, **DISPLAY CHANNEL**, and **DELETE CHANNEL**.

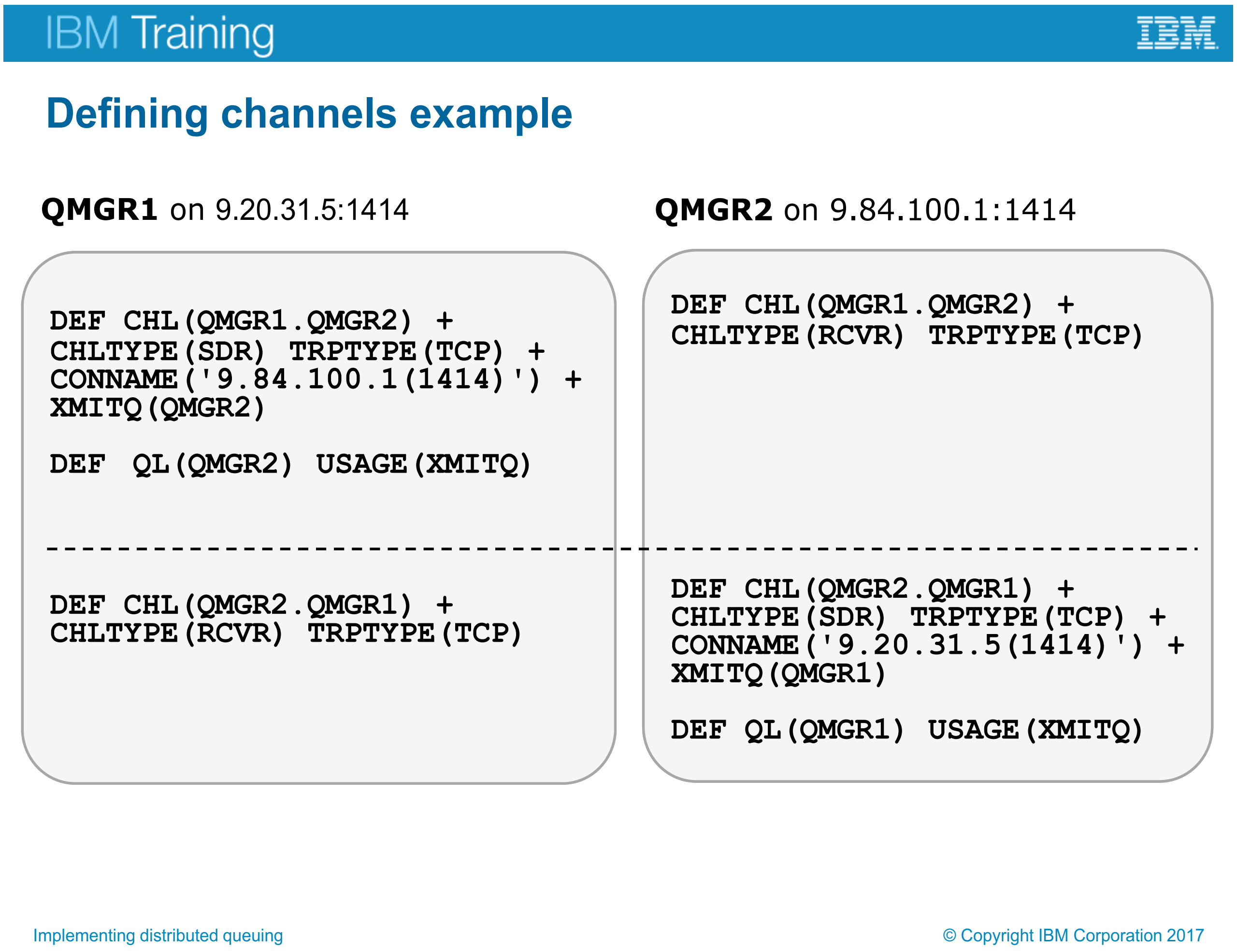
The rules for naming a channel are the same as the rules for naming a queue except that the name of a channel is limited to 20 characters. The channel definition at each end of a channel must specify the same channel name.

Attributes not supplied on the **DEFINE CHANNEL** command are taken from the appropriate default channel object based on the channel type (**CHLTYPE**) attribute.

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

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

The value of the **CONNAME** parameter depends on the communication protocol that is used and in some cases, on the operating system. For TCP/IP, it might be the IP address or the host name of the system on which the remote queue manager is running.



IBM Training

## Defining channels example

**QMGR1** on 9.20.31.5:1414

```
DEF CHL(QMGR1.QMGR2) +
CHLTYPE(SDR) TRPTYPE(TCP) +
CONNAME('9.84.100.1(1414)') +
XMITQ(QMGR2)

DEF  QL(QMGR2) USAGE(XMITQ)
```

```
DEF CHL(QMGR2.QMGR1) +
CHLTYPE(RCVR)  TRPTYPE(TCP)
```

**QMGR2** on 9.84.100.1:1414

```
DEF CHL(QMGR1.QMGR2) +
CHLTYPE(RCVR)  TRPTYPE(TCP)
```

```
DEF CHL(QMGR2.QMGR1) +
CHLTYPE(SDR) TRPTYPE(TCP) +
CONNAME('9.20.31.5(1414)') +
XMITQ(QMGR1)

DEF QL(QMGR1) USAGE(XMITQ)
```

Implementing distributed queuing

© Copyright IBM Corporation 2017

*Figure 6-18. Defining channels example*

This figure shows the MQSC commands that you would enter on each queue manager to create two-way communication between the queue managers. Each queue manager has two channel definitions. QMGR1 has a sender channel to QMGR2 and receiver channel from QMGR2. QMGR2 has a receiver channel for QMGR1 and a sender channel to QMGR1.

The definition of a channel at each end of the channel must specify the same channel name. The example follows the convention that the name of a transmission queue is the same as the name of the queue manager at the other end channel.

For TCP/IP, you can use either the IP address or the host name on the **CONNAME** parameter. The port number is the port number of the queue manager listener, which must be running.

IBM Training

## Defining channels in IBM MQ Explorer (1 of 2)

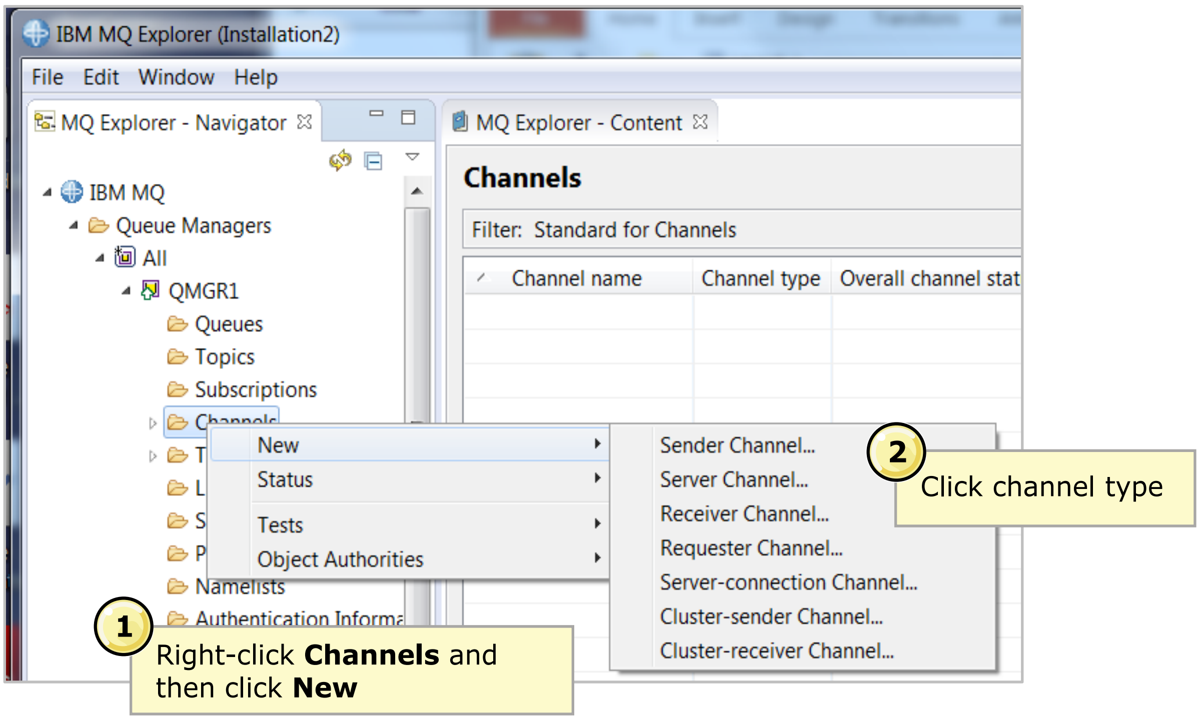


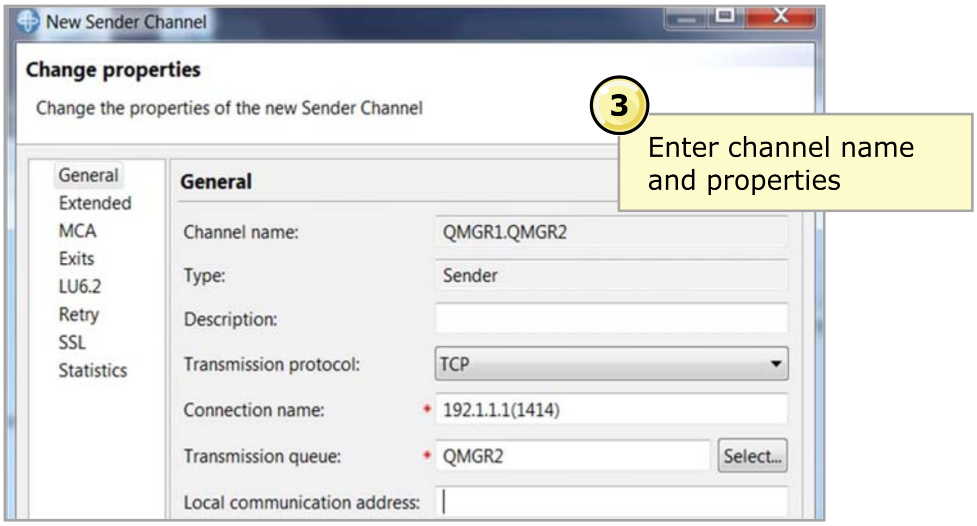
*Figure 6-19. Defining channels in IBM MQ Explorer (1 of 2)*

You can also use MQ Explorer to define channels.

1. In the **MQ Explorer - Navigator** view, expand the queue manager folder.
2. Under the queue manager folder, right-click **Channels**, click **New**, and then click the channel type.

IBM Training

## Defining channels in IBM MQ Explorer (2 of 2)


New Sender Channel

Change properties

Change the properties of the new Sender Channel

General

| | |
|---|---|
| Channel name: | QMGR1.QMGR2 |
| Type: | Sender |
| Description: | |
| Transmission protocol: | TCP |
| Connection name: * | 192.1.1.1(1414) |
| Transmission queue: * | QMGR2 |
| Local communication address: | |

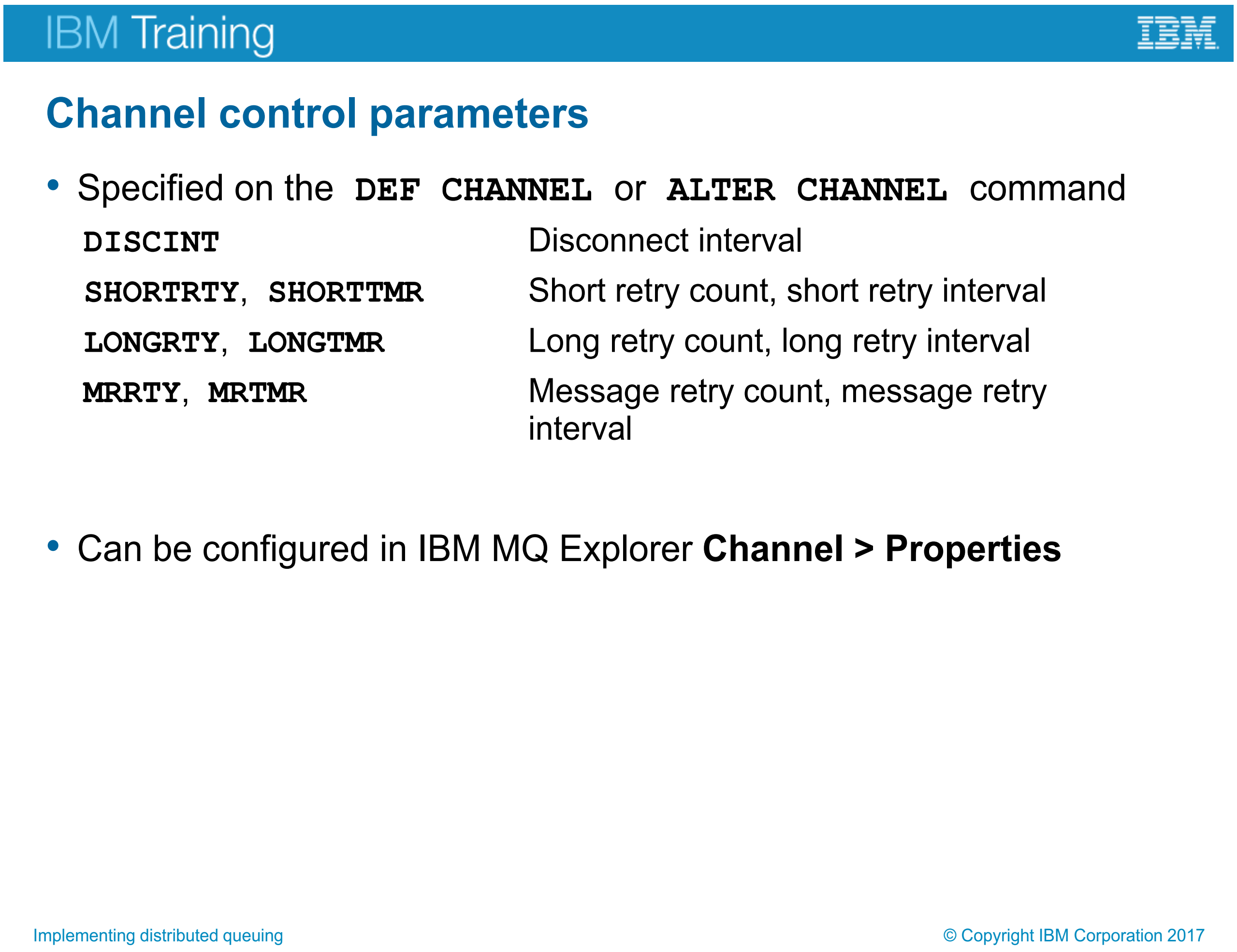
| | |
|---|---|
| • **MCA** | Specify how MCA program of the channel definition runs |
| • **Exits** | Configure any user exits |
| • **LU6.2** | If MCA uses the LU6.2 transport protocol, configure MCA |
| • **Retry** | If channel cannot connect to remote queue manager, configure channel behavior |
| • **SSL** | Specify SSL settings for this end of the channel |
| • **Statistics** | Control collection of statistics and monitoring data for the channel |

Implementing distributed queuing © Copyright IBM Corporation 2017

*Figure 6-20. Defining channels in IBM MQ Explorer (2 of 2)*

3. Configure the channel properties.

   The figure summarizes some of the channel definition property categories. You must provide values for the required properties.



IBM Training

## Channel control parameters

- Specified on the `DEF CHANNEL` or `ALTER CHANNEL` command

| | |
|---|---|
| `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 |

- Can be configured in IBM MQ Explorer **Channel > Properties**

Implementing distributed queuing © Copyright IBM Corporation 2017

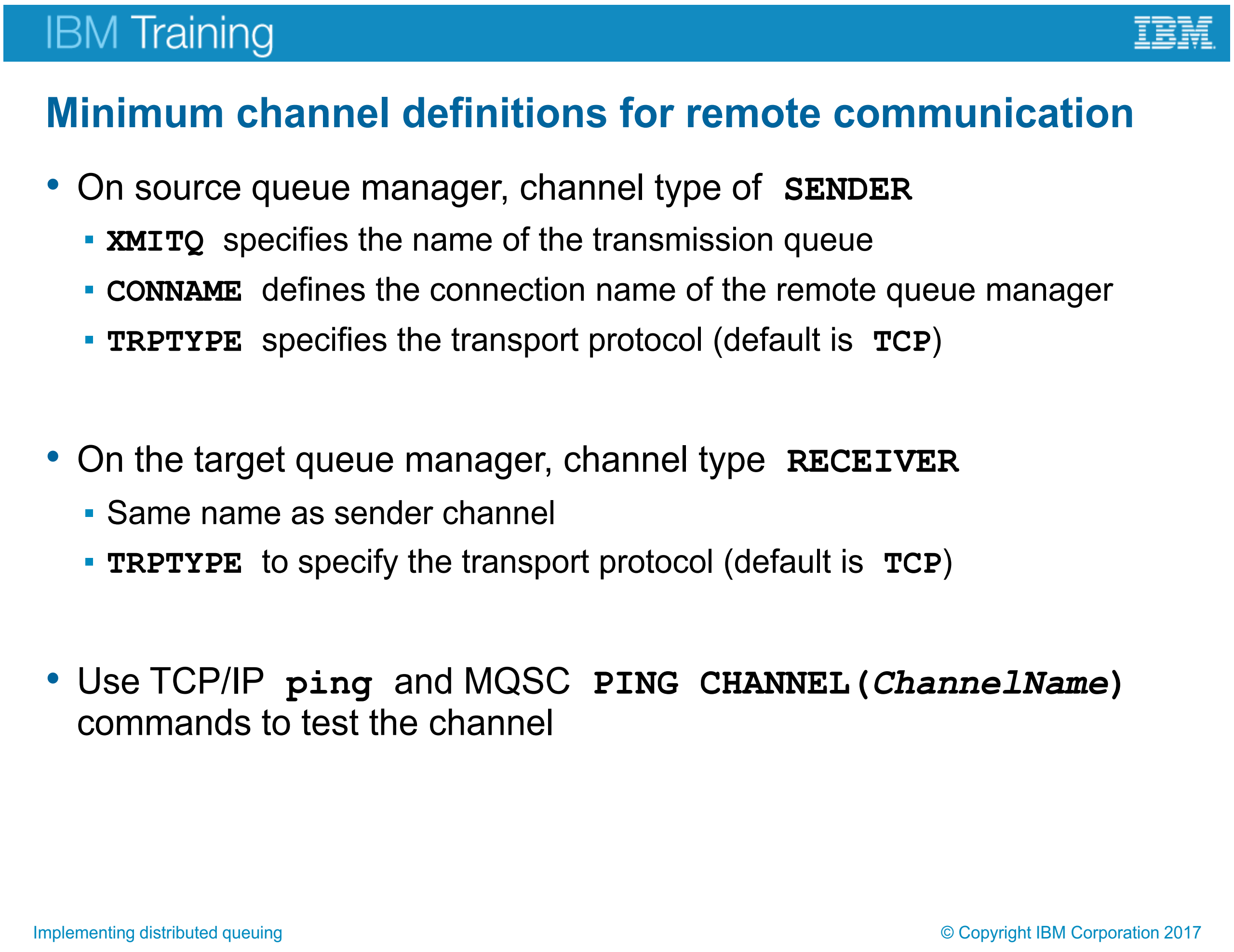
*Figure 6-21. Channel control parameters*

The channel control parameters that are shown in the figure can be specified on the **`DEFINE CHANNEL`** or **`ALTER CHANNEL`** command, or in MQ Explorer.

- **DISCINT** (on SDR and SVR channels): 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.
- **SHORTRTY, SHORTTMR** (on SDR and SVR channels): Short retry count and short retry time interval to control repeated attempts to establish a communications connection.
- **LONGRTY, LONGTMR** (on SDR and SVR channels): Long retry count and long retry time interval to control further repeated attempts to establish a communications connection.
- **MRRTY, MRTMR** (on RCVR and RQSTR channels): Message-retry count and message-retry interval when MQ attempts to put a message on a destination queue. If every attempt fails, the MCA decides that it cannot deliver the message.

The disconnect interval and retry attributes provide some automation in the operation of channels. They allow channels to stop during periods of inactivity and restart when more messages arrive.

Of the parameters that are listed, the channel initiator uses only the **SHORTRTY**, **SHORTTMR**, **LONGRTY**, **LONGTMR**, and **MCATYPE** parameters.



IBM Training

## Minimum channel definitions for remote communication

- On source queue manager, channel type of **`SENDER`**
  - **`XMITQ`** specifies the name of the transmission queue
  - **`CONNAME`** defines the connection name of the remote queue manager
  - **`TRPTYPE`** specifies the transport protocol (default is **`TCP`**)
- On the target queue manager, channel type **`RECEIVER`**
  - Same name as sender channel
  - **`TRPTYPE`** to specify the transport protocol (default is **`TCP`**)
- Use TCP/IP **`ping`** and MQSC **`PING CHANNEL(`*`ChannelName`*`)`** commands to test the channel

Implementing distributed queuing © Copyright IBM Corporation 2017

*Figure 6-22. Minimum channel definitions for remote communication*

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

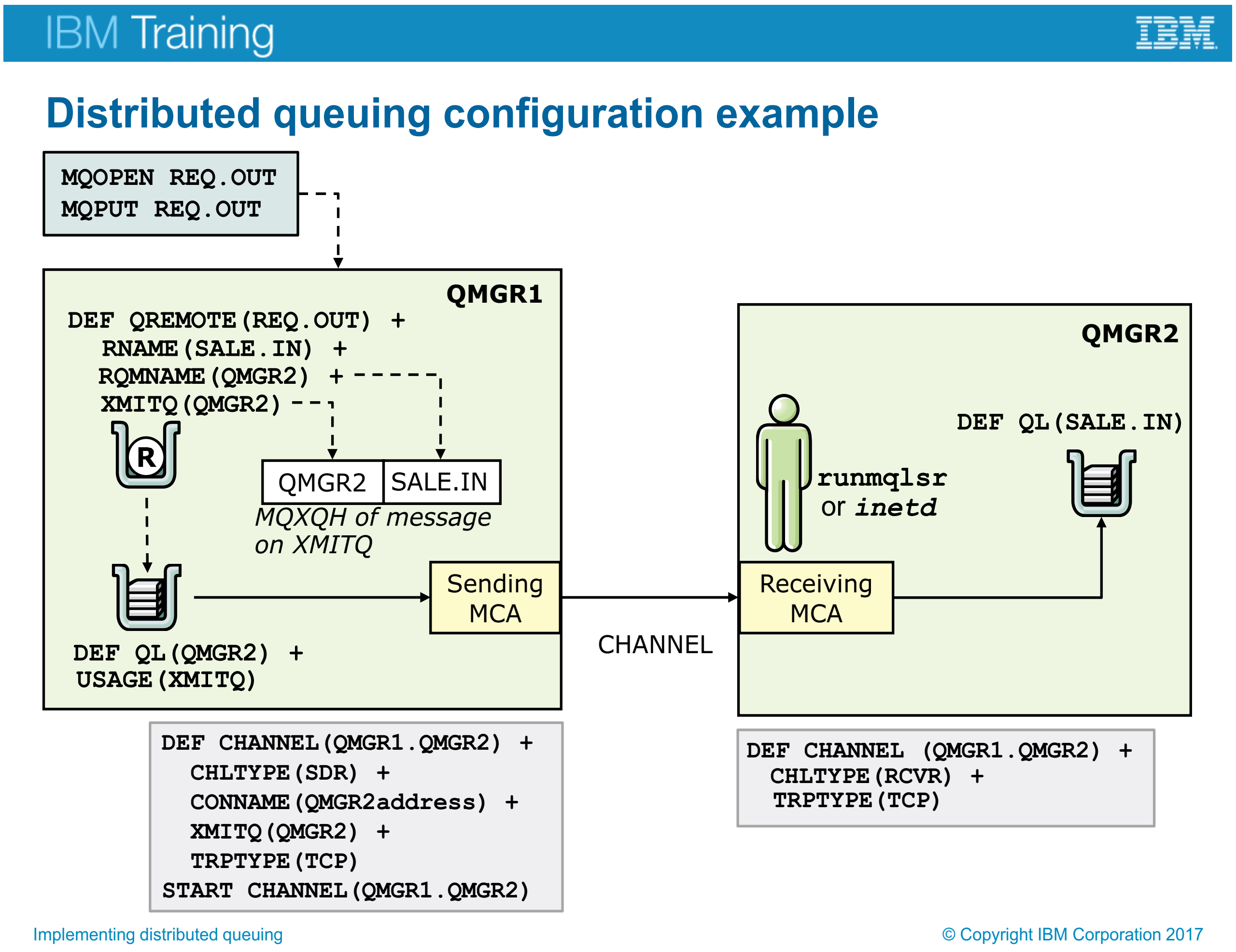
On the source queue manager, define a channel with a channel type of **`SENDER`**. You must specify:

- The name of the transmission queue to use (the **`XMITQ`** attribute)
- The connection name of the remote system (the **`CONNAME`** attribute)
- The communication protocol type (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. Also, specify the communication protocol on the channel (the **`TRPTYPE`** attribute).

The receiver channel definitions can be generic. Generic channels mean that if you have several queue managers that communicate with the same receiver, the sending channels can all specify the same name for the receiver, and one receiver definition applies to them all.

To verify the connection, use the TCP/IP **`ping`** command to verify network connectivity between the servers. Upon a successful **`ping`**, use the MQSC **`PING CHANNEL`** command to verify the channel.



## Distributed queuing configuration example

```
DEF CHANNEL(QMGR1.QMGR2) +
   CHLTYPE(SDR) +
   CONNAME(QMGR2address) +
   XMITQ(QMGR2) +
   TRPTYPE(TCP)
START CHANNEL(QMGR1.QMGR2)
```

```
DEF CHANNEL (QMGR1.QMGR2) +
  CHLTYPE(RCVR) +
  TRPTYPE(TCP)
```

Implementing distributed queuing
© Copyright IBM Corporation 2017

*Figure 6-23. Distributed queuing configuration example*

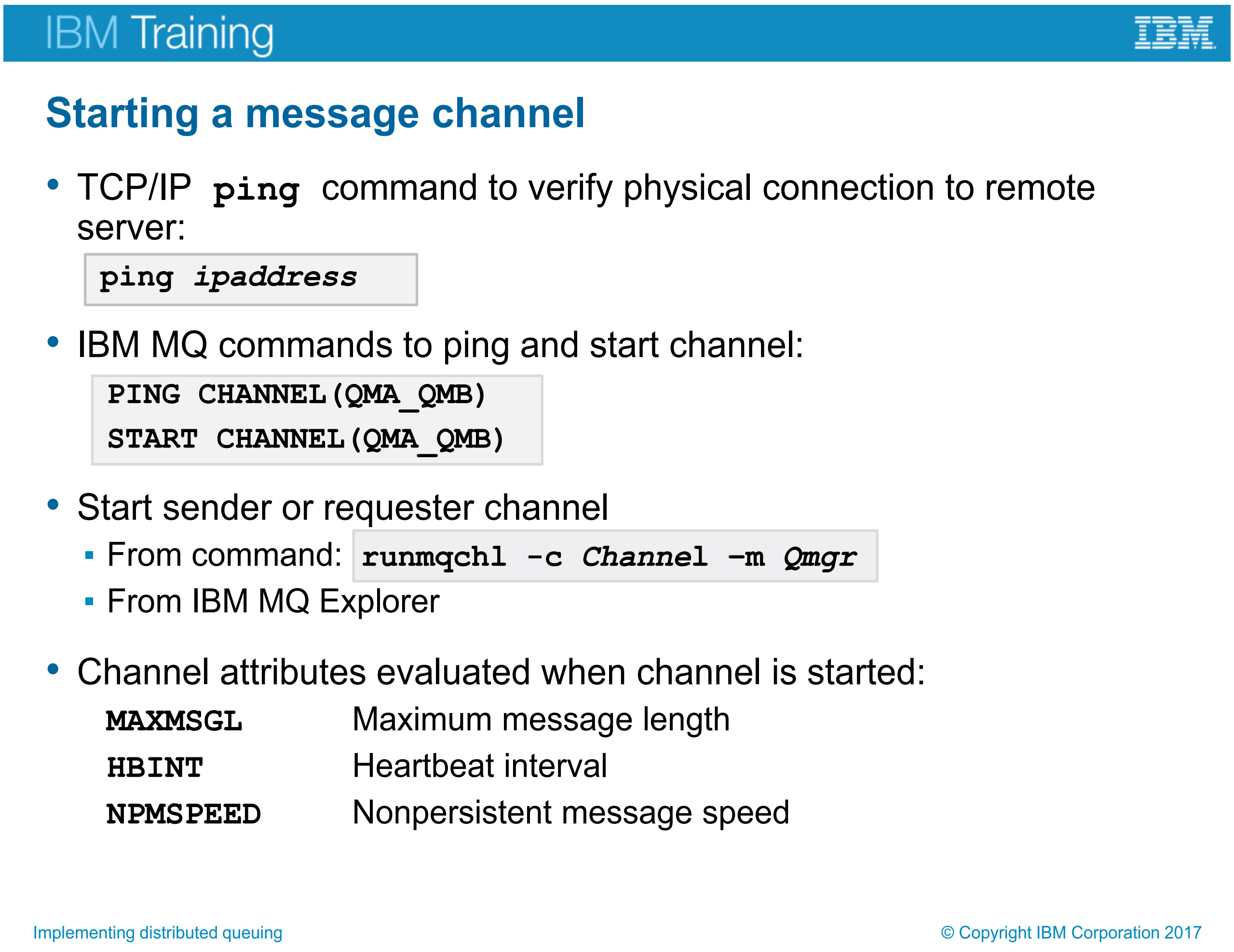
Whether an application is putting a message on a local queue or to a remote queue is not apparent to the application. However, an application always gets a message from a local queue.

A persistent message is not lost when a communications or system failure occurs, nor is it ever delivered twice.

A message that is destined for a remote queue manager is stored locally on a transmission queue until the MCA can send it.

In the example, a remote queue and transmission queue are defined on queue manager QMGR1. Channels are defined on both the sender queue manager, QMGR1, and the receiving queue manager, QMGR2. The sender channel is started on the sender by using the **START CHANNEL** command. The sender and receiver channels are identified by the same name to avoid any confusion with channels that might be defined for connections to other queue managers in the network.

The example also shows that the MQXQH contains the queue manager name and the target queue name that were specified on the local definition of the remote queue.



IBM Training

## Starting a message channel

- TCP/IP `ping` command to verify physical connection to remote server:

```
ping ipaddress
```

- IBM MQ commands to ping and start channel:

```
PING CHANNEL(QMA_QMB)
START CHANNEL(QMA_QMB)
```

- Start sender or requester channel
  - From command: `runmqchl -c Channel -m Qmgr`
  - From IBM MQ Explorer

- Channel attributes evaluated when channel is started:

| | |
|---|---|
| **MAXMSGL** | Maximum message length |
| **HBINT** | Heartbeat interval |
| **NPMSPEED** | Nonpersistent message speed |

Implementing distributed queuing © Copyright IBM Corporation 2017

*Figure 6-24. Starting a message channel*

Before you start a channel, verify the physical connection by using the TCP/IP **ping** command. Always check that the network is working before you attempt to start a channel.

Use the MQSC **PING CHANNEL** command to test a message channel configuration. It can be used at the sender or server end of a channel, and only when the channel is not started.

Start a channel by using the MQSC **START CHANNEL** command. Optionally, you can use the **runmqchl** control command to start a sender (SDR) or a requester (RQSTR) channel. The channel runs synchronously. You can also use MQ Explorer to start a channel.

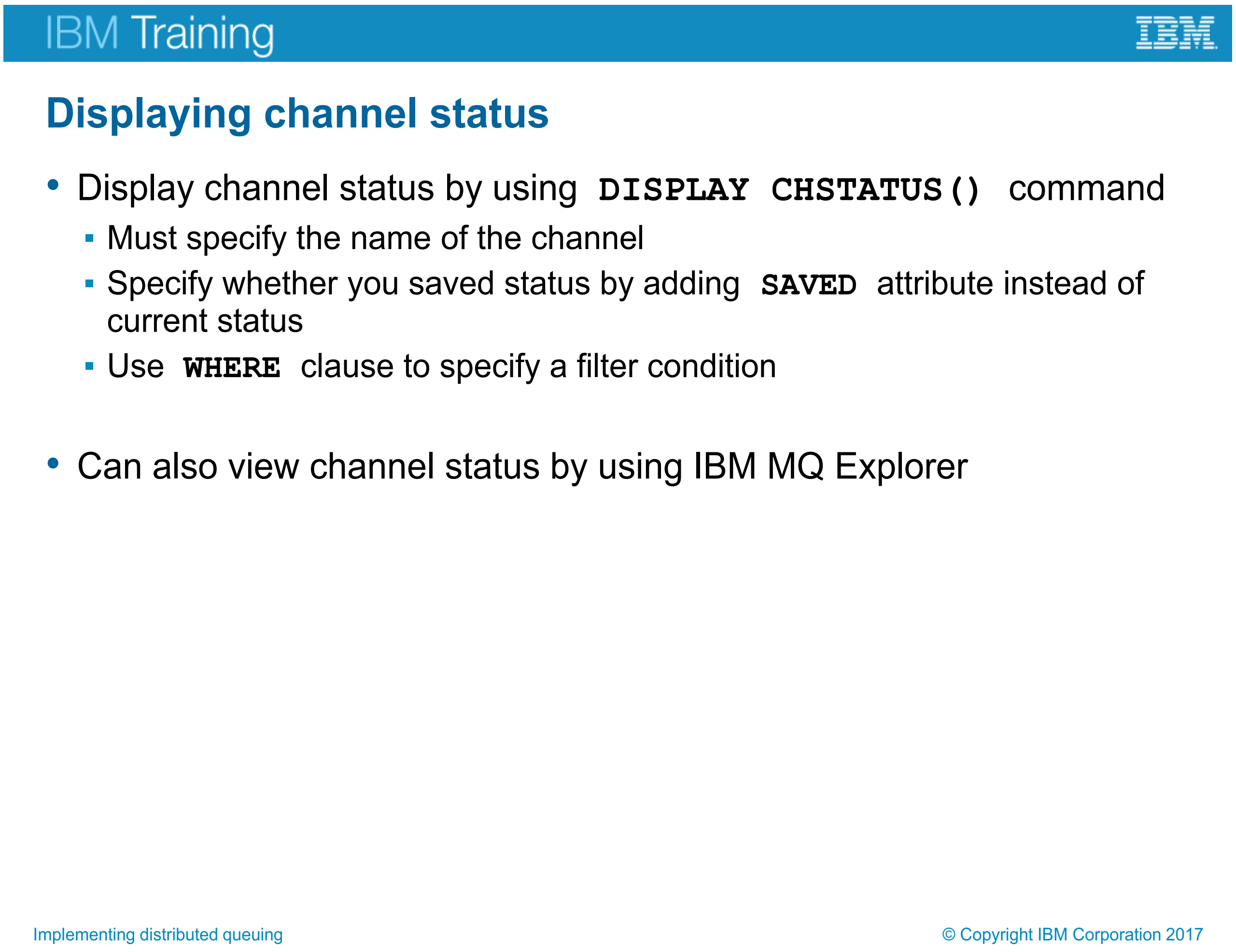
Take particular care that the channel definitions are correct. Some attributes that are specified in the channel definition at each end of a message channel are compared when the channel is started:

- **MAXMSGL** is the maximum message length that the channel can transmit. The lower value from the two channel definitions is used.
- **HBINT** is the time between heartbeat flows that are sent from a sending MCA to a receiving MCA when no messages exist on the transmission queue. A heartbeat flow can unblock a receiving MCA for which the **STOP CHANNEL** command is entered. The higher value from the two channel definitions is used.

- **NPMSPEED** is the speed at which nonpersistent messages are sent. You can specify **NORMAL** or **FAST**. The default is **FAST**, which means that a nonpersistent message is sent outside of a batch and 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.

  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.

One of the most common problems in MQ is getting your first message channel to work. After you get the first message channel to work, things become much simpler.



IBM Training

## Displaying channel status

- Display channel status by using **`DISPLAY CHSTATUS()`** command
  - Must specify the name of the channel
  - Specify whether you saved status by adding **`SAVED`** attribute instead of current status
  - Use **`WHERE`** clause to specify a filter condition
- Can also view channel status by using IBM MQ Explorer

Implementing distributed queuing © Copyright IBM Corporation 2017

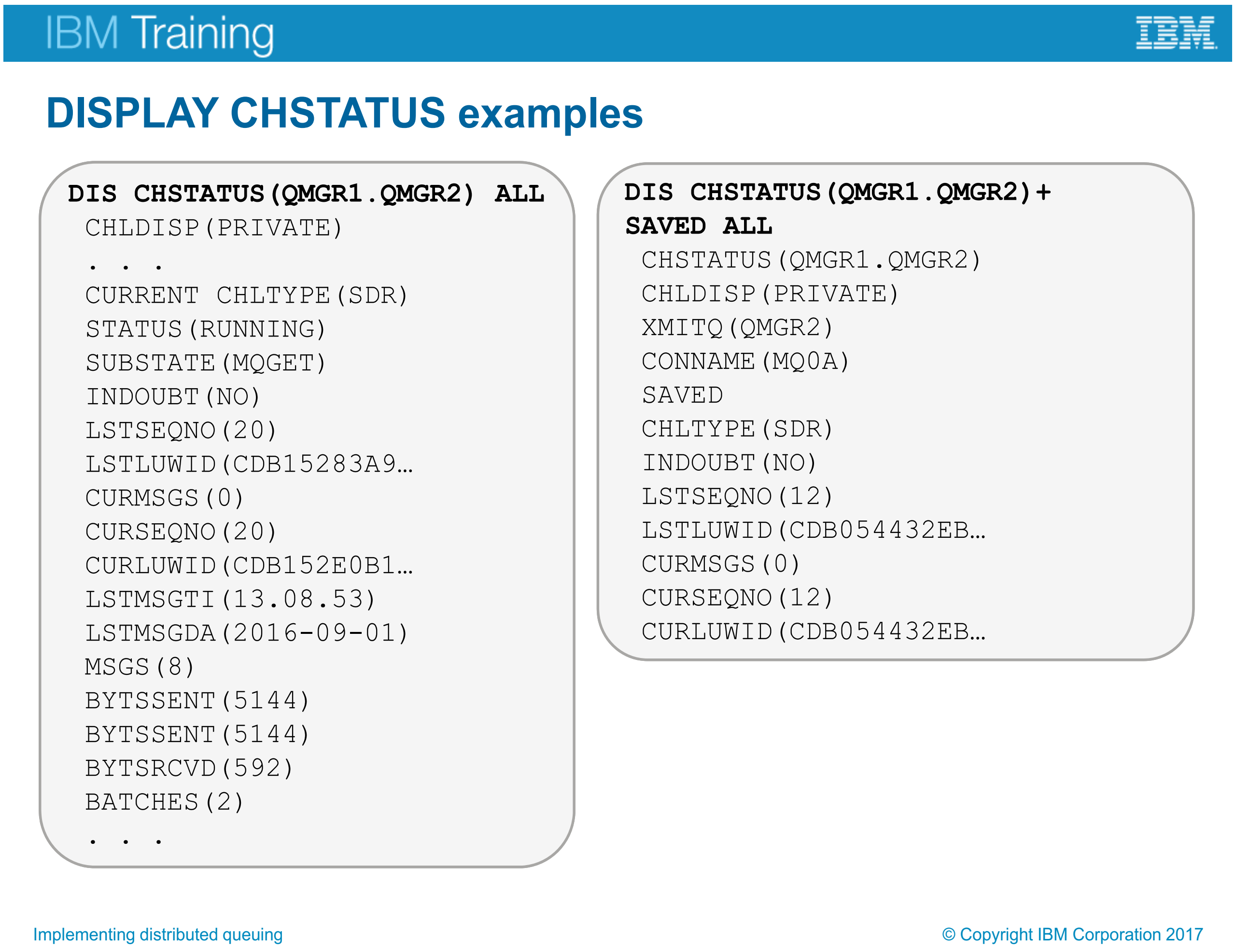
*Figure 6-25. Displaying channel status*

You can use the **`DISPLAY CHSTATUS`** command or MQ Explorer to display the status of one or more channels.

When you use the **`DISPLAY CHSTATUS`** command, you must specify whether you want the *current status data* or the *saved status data*. By default, current status data is returned if saved status data is not explicitly requested.

- The current status data for a channel is data that is derived from the entry for the channel in the channel status table. An inactive channel has no current status data.
- The saved status data for a channel is data that is derived from the status message for the channel on the synchronization queue, or from the in-doubt status message if the channel is in-doubt. An inactive channel can have status data that is saved.

Certain status data is returned only when current status data is requested. These status fields are referred to as *current-only* status fields. The values of current-only status fields are derivable only from data that is stored in the channel status table. They cannot be derived from data that is stored in scratchpad objects or messages on the synchronization queue. The figure lists the keywords that are used on the **`DISPLAY CHSTATUS`** command to request these fields.


IBM Training

## DISPLAY CHSTATUS examples

```
DIS CHSTATUS(QMGR1.QMGR2) ALL
 CHLDISP(PRIVATE)
 . . .
 CURRENT CHLTYPE(SDR)
 STATUS(RUNNING)
 SUBSTATE(MQGET)
 INDOUBT(NO)
 LSTSEQNO(20)
 LSTLUWID(CDB15283A9…
 CURMSGS(0)
 CURSEQNO(20)
 CURLUWID(CDB152E0B1…
 LSTMSGTI(13.08.53)
 LSTMSGDA(2016-09-01)
 MSGS(8)
 BYTSSENT(5144)
 BYTSSENT(5144)
 BYTSRCVD(592)
 BATCHES(2)
 . . .
```

```
DIS CHSTATUS(QMGR1.QMGR2)+
SAVED ALL
 CHSTATUS(QMGR1.QMGR2)
 CHLDISP(PRIVATE)
 XMITQ(QMGR2)
 CONNAME(MQ0A)
 SAVED
 CHLTYPE(SDR)
 INDOUBT(NO)
 LSTSEQNO(12)
 LSTLUWID(CDB054432EB…
 CURMSGS(0)
 CURSEQNO(12)
 CURLUWID(CDB054432EB…
```

Implementing distributed queuing

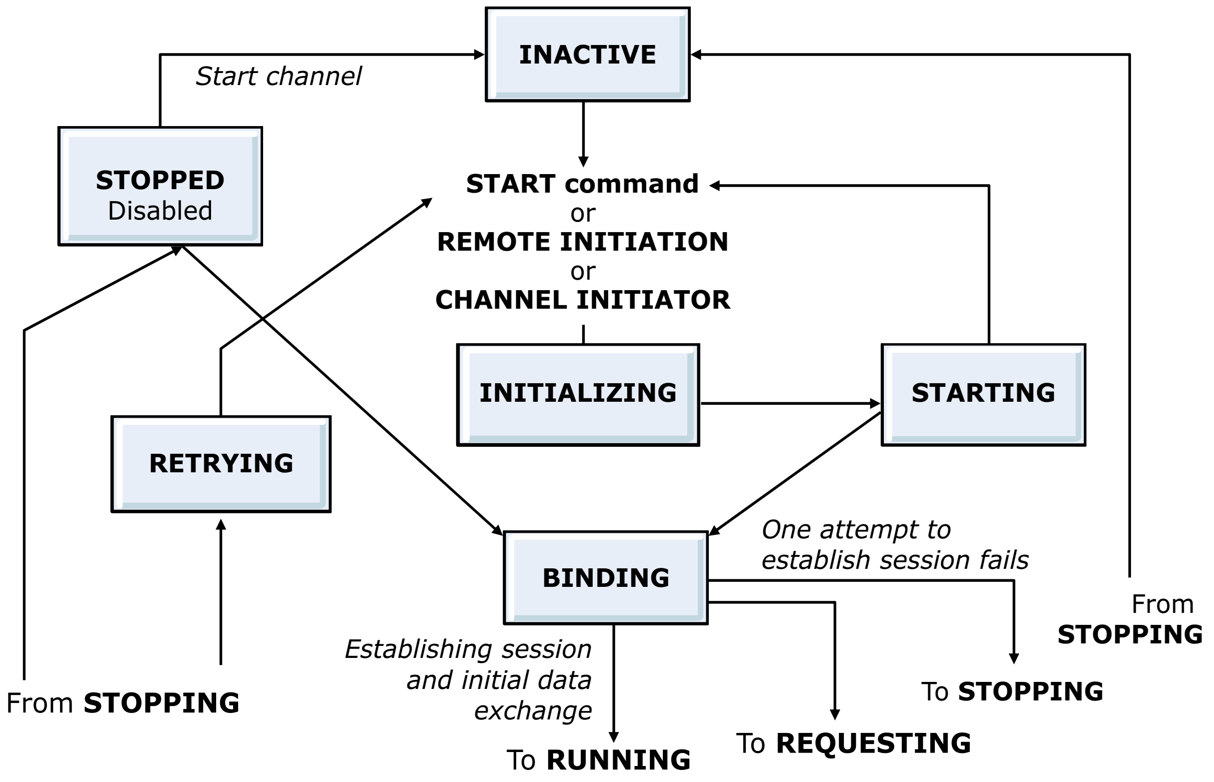
© Copyright IBM Corporation 2017

*Figure 6-26. DISPLAY CHSTATUS examples*

This figure shows two examples of the **`DISPLAY CHSTATUS`** command and a summary of the information the command returns. The example on the right includes the **`SAVED`** attribute and displays the saved status data.

IBM Training

## Channel states (1 of 2)



Implementing distributed queuing © Copyright IBM Corporation 2017

*Figure 6-27. Channel states (1 of 2)*

The current state of a channel can be determined by using the MQSC **DISPLAY CHSTATUS** command. This figure and the figure on the next page show the possible states of a channel and the possible flows from one state to the next.

When a channel is in the INITIALIZING, BINDING, REQUESTING, RUNNING, PAUSED, or STOPPING state, it is using resources and a process or thread is running; the channel is active.

INACTIVE is not really a state. It indicates a start point for when the **START CHANNEL** command is entered, a transmission queue is triggered, a channel initiator issues a retry, or an incoming request to start a channel exists.

When a channel is in the INITIALIZING state, a channel initiator is attempting to start a channel.

A channel stays in the STARTING state when no active slot is available. If an active slot is immediately available, it remains in this state for a short time.

The RETRYING state indicates that the channel is waiting until it is time for the channel initiator to make the next attempt to start the channel. An error can cause a channel to go into the RETRYING state when it is possible that the problem might clear itself. Otherwise, it goes into the STOPPED state. Manual intervention is required to start a channel that is in the STOPPED state.

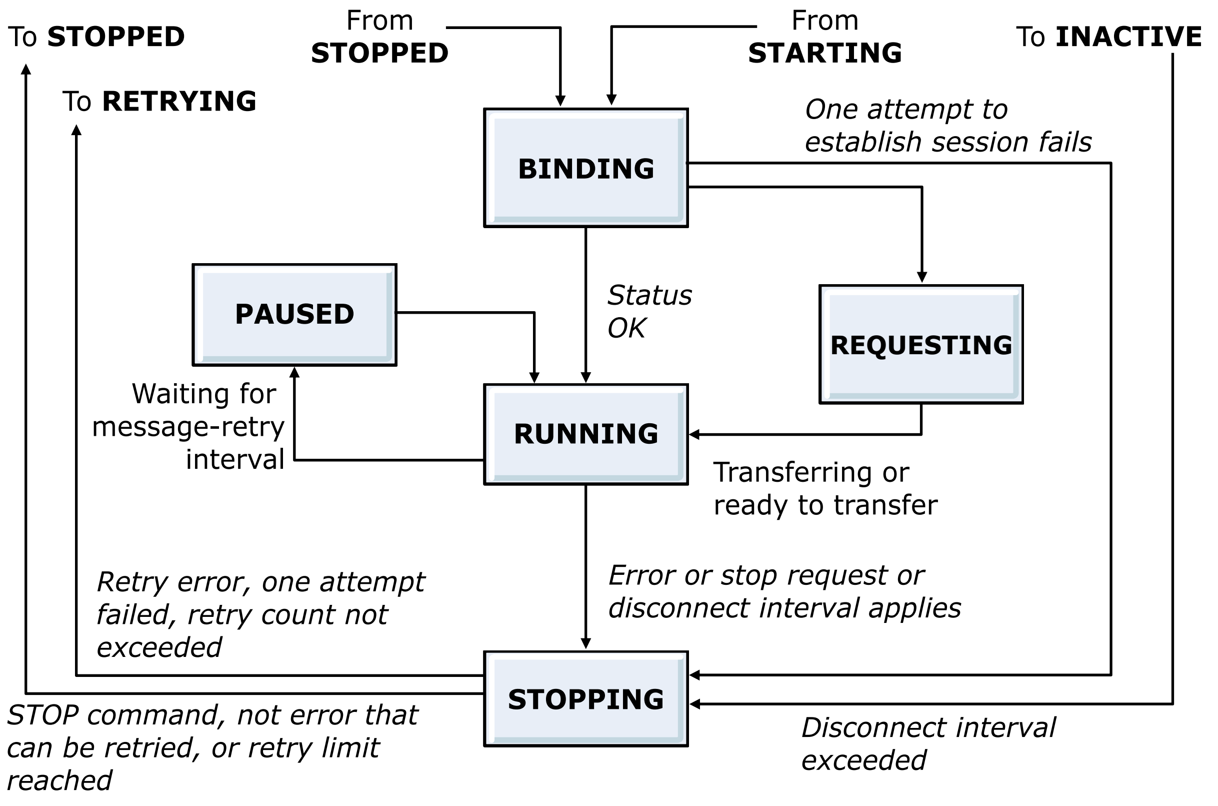
A **STOP CHANNEL** command also puts a channel into the STOPPED state. The **STOP CHANNEL** command can be entered against any type of channel except a client-connection channel. You can

STOP a channel by specifying the channel name and the remote connection name or the remote queue manager name.

If you specify either the queue manager name or connection name, the status of the channel must be INACTIVE.

IBM Training

## Channel states (2 of 2)


To STOPPED
From STOPPED
From STARTING
To INACTIVE
To RETRYING
BINDING
One attempt to establish session fails
PAUSED
Status OK
REQUESTING
Waiting for message-retry interval
RUNNING
Transferring or ready to transfer
Retry error, one attempt failed, retry count not exceeded
Error or stop request or disconnect interval applies
STOPPING
STOP command, not error that can be retried, or retry limit reached
Disconnect interval exceeded

Implementing distributed queuing

© Copyright IBM Corporation 2017

*Figure 6-28. Channel states (2 of 2)*

During the BINDING state, the channel establishes a communications connection and completes the initial data exchange.

In the REQUESTING state, a requester is waiting for callback from a sender.

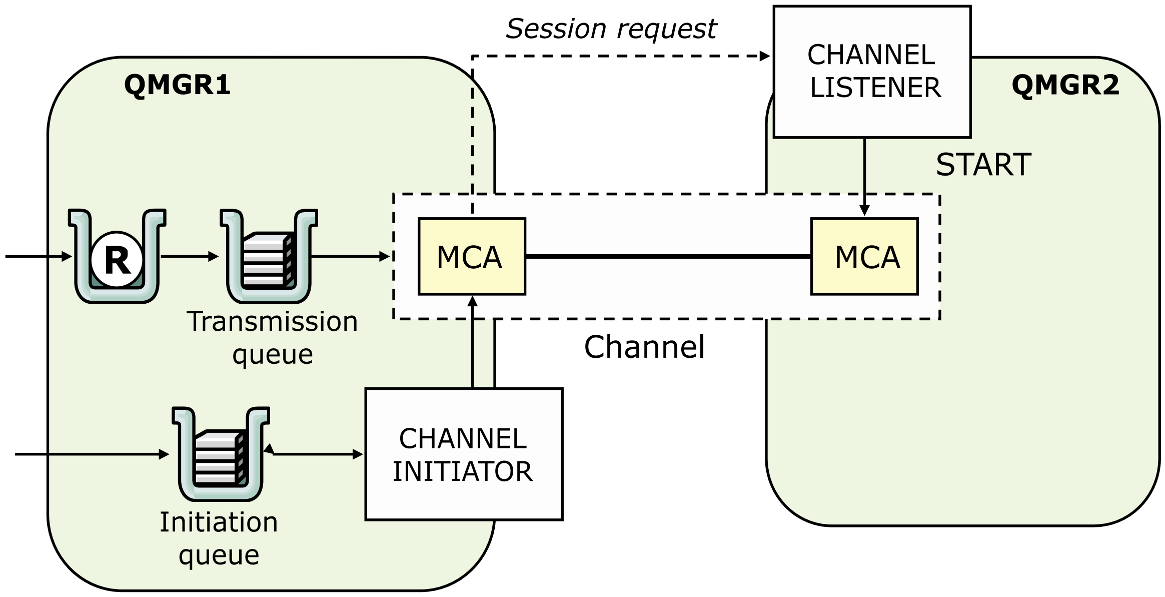
In the RUNNING state, messages are being transferred or the channel is waiting for messages to arrive on the transmission queue.

A PAUSED channel is waiting for the message-retry interval to complete before MQ attempts to put a message on its destination queue.

The channel enters the STOPPING state if an error occurs, the **`STOP CHANNEL`** command is entered, or the disconnect interval expires.

IBM Training

## Channel initiators and listeners



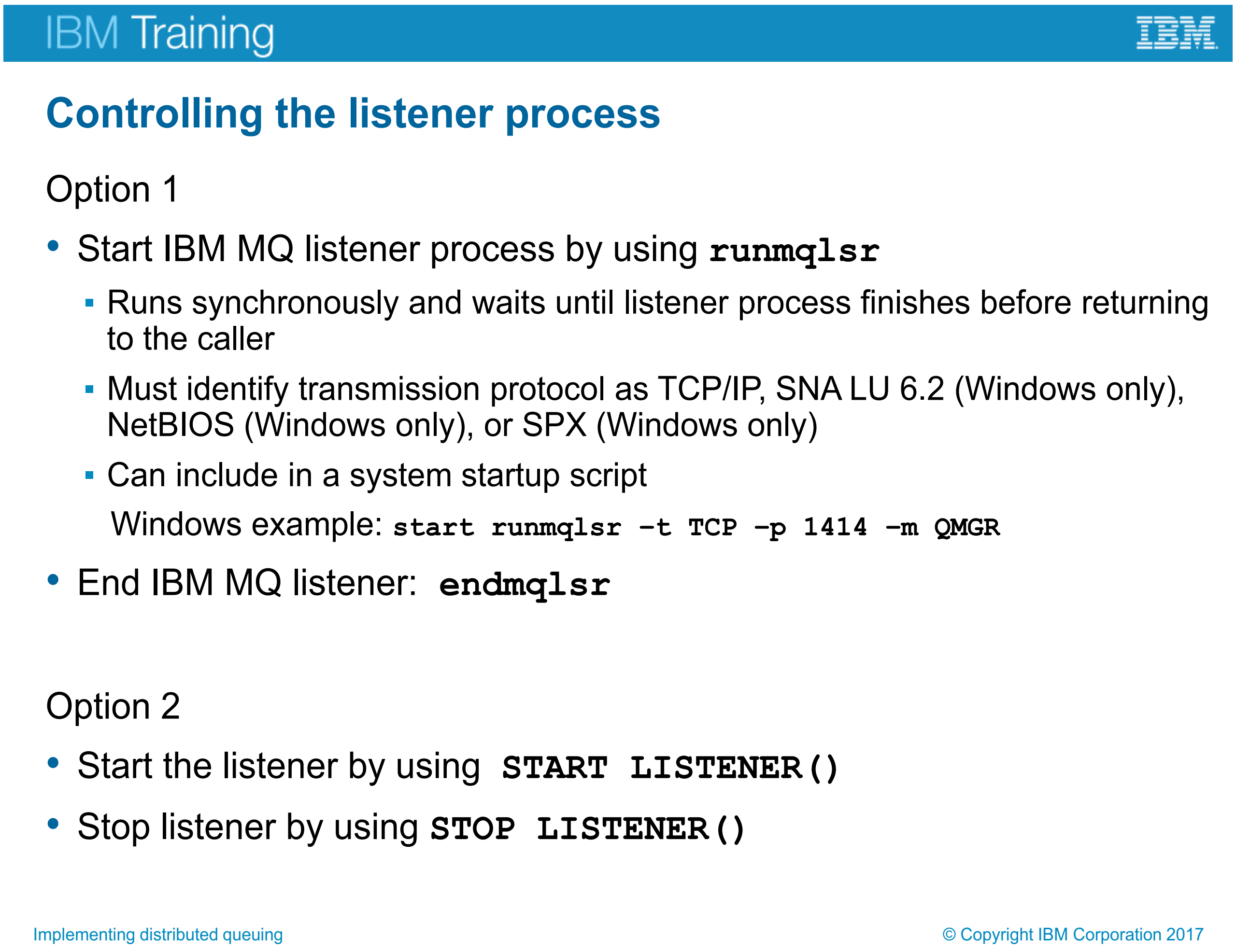
- Channel listener waits to start the other end of a channel to receive the message
- Channel initiator is needed to start a communication channel when a message is to be delivered

*Figure 6-29. Channel initiators and listeners*

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, which triggers the channel initiator to start the appropriate sender channel. Server channels can also be started in this way when the connection name of the server is specified in the channel definition. Channels can be started automatically, based on messages that arrive on the appropriate transmission queue.

You need a *channel listener* 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.

The figure shows a message flow that uses a channel initiator and a channel listener. A message is put on the initiation queue of QMGR1. The message triggers the channel initiator for that queue manager, which starts the sender channel. The channel listener at QMGR2 receives the session request and starts the receiving MCA. Messages can then be transmitted from QMGR1 to QMGR2.



IBM Training

## Controlling the listener process

Option 1

- Start IBM MQ listener process by using **`runmqlsr`**
  - Runs synchronously and waits until listener process finishes before returning to the caller
  - Must identify transmission protocol as TCP/IP, SNA LU 6.2 (Windows only), NetBIOS (Windows only), or SPX (Windows only)
  - Can include in a system startup script

    Windows example: **`start runmqlsr -t TCP -p 1414 -m QMGR`**
- End IBM MQ listener: **`endmqlsr`**

Option 2

- Start the listener by using **`START LISTENER()`**
- Stop listener by using **`STOP LISTENER()`**

Implementing distributed queuing © Copyright IBM Corporation 2017

*Figure 6-30. Controlling the listener process*

You can use the listener for all the supported communications protocols.

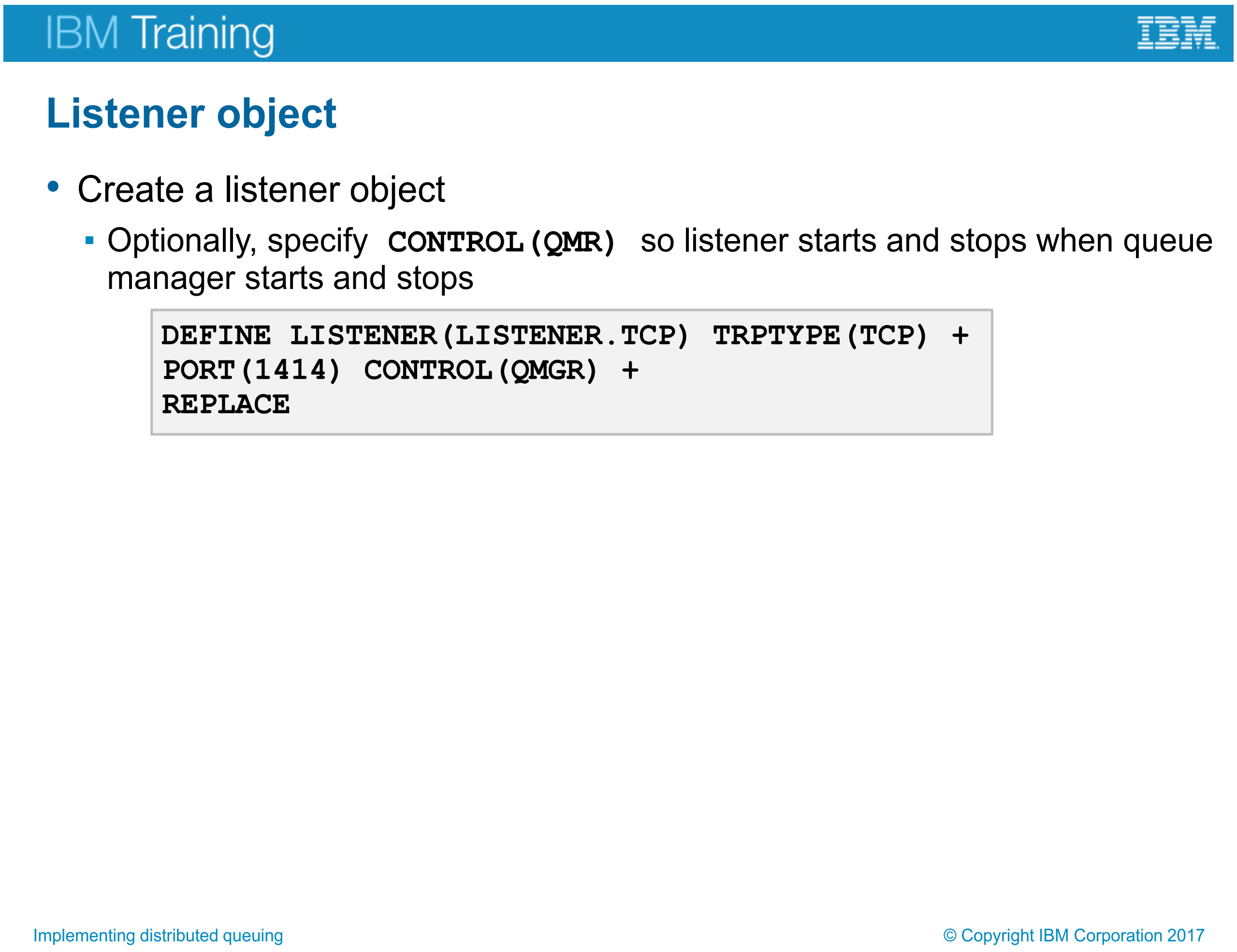
The Internet Assigned Numbers Authority assigns port number 1414 to MQ. By default, MQ uses this port number for the queue manager listener port. If you are running multiple queue managers on the same computer, ensure that each queue manager uses a unique listener port.

When MQ uses TCP/IP to start a message channel at one end of the channel, a listener process must be running at the other end.

To run the listener that is supplied with MQ, use the **`runmqlsr`** command. One advantage of using **`runmqlsr`** is that a channel runs as a thread within the listener process. This command is run synchronously and waits until the listener process finishes before it returns to the caller.

The **`endmqlsr`** command ends all listener processes for the specified queue manager.

The implementation of channel listeners is operating system specific; however, some common features exist. On all MQ operating systems, the listener can be started by using the MQSC command **`START LISTENER`**.



IBM Training

## Listener object

- Create a listener object
  - Optionally, specify **CONTROL(QMR)** so listener starts and stops when queue manager starts and stops

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

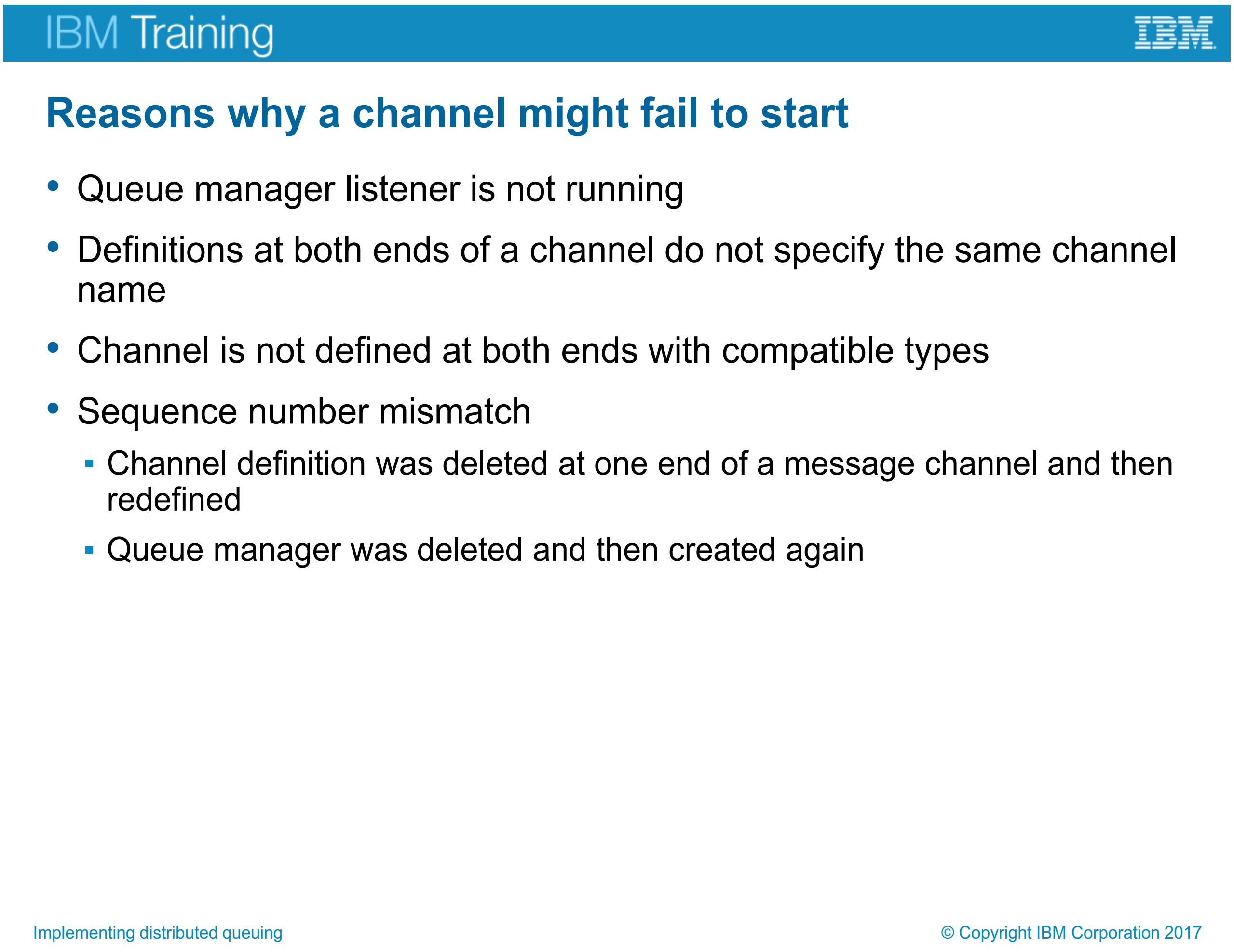
Implementing distributed queuing © Copyright IBM Corporation 2017

*Figure 6-31. Listener object*

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.

Use the MQSC command **DEFINE LISTENER** to define a new MQ listener definition, and set its properties. The **CONTROL** attribute specifies how the listener is started and stopped:

- If you want the listener to start and stop when the queue manager starts and stops, specify the **CONTROL(QMGR)** property.
- If you want the listener to start when the queue manager starts but not stop when the queue manager stops, specify the **CONTROL(STARTONLY)** property.
- If you want to manually control when the listener starts and stops, specify the **CONTROL(MANUAL)** property.



IBM Training

## Reasons why a channel might fail to start

- Queue manager listener is not running
- Definitions at both ends of a channel do not specify the same channel name
- Channel is not defined at both ends with compatible types
- Sequence number mismatch
  - Channel definition was deleted at one end of a message channel and then redefined
  - Queue manager was deleted and then created again

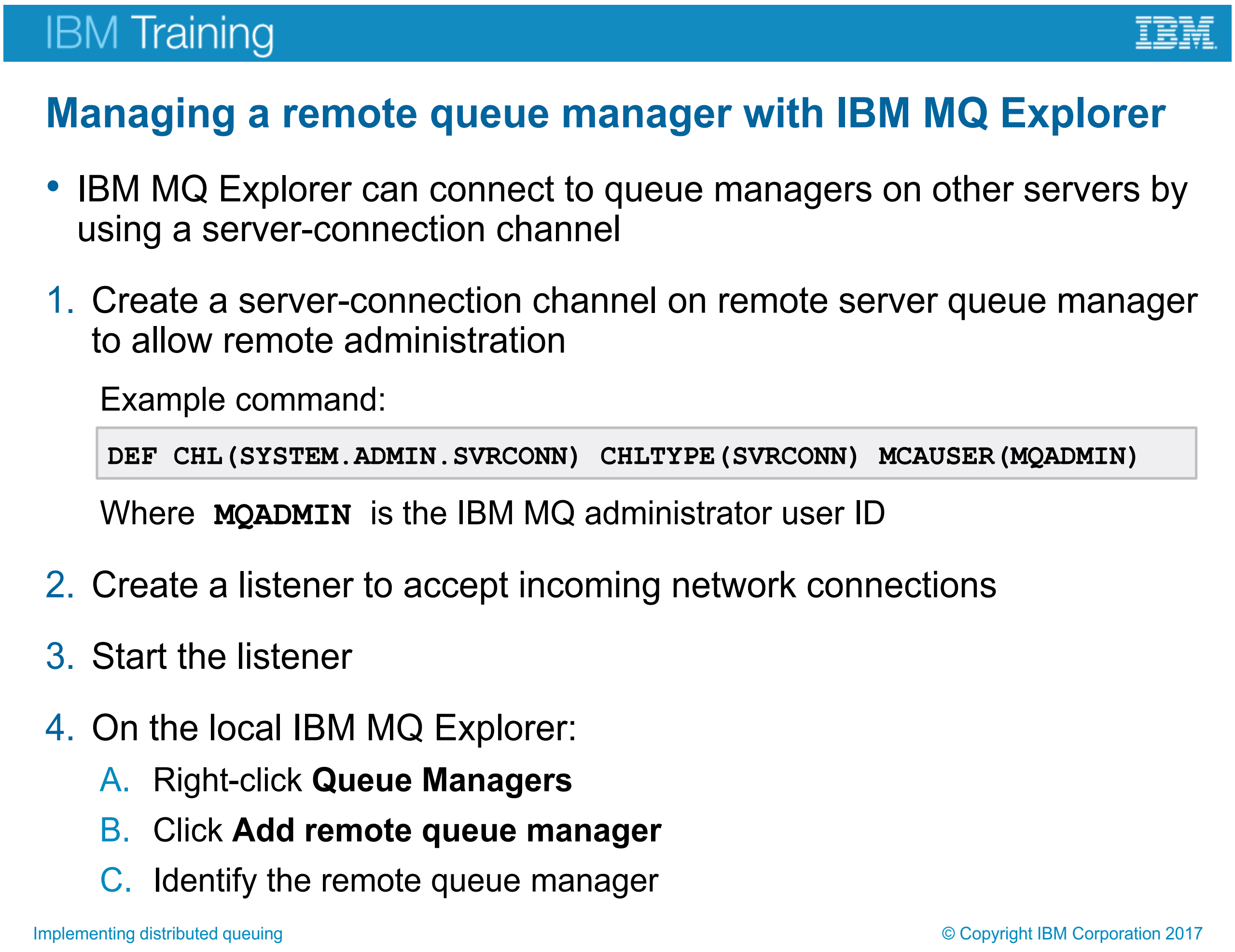
Implementing distributed queuing

© Copyright IBM Corporation 2017

*Figure 6-32. Reasons why a channel might fail to start*

Several reasons exist for why a channel might fail to start.

- The queue manager channel listener is not running.
- 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.
- A sequence number mismatch exists, 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.



IBM Training

## Managing a remote queue manager with IBM MQ Explorer

- IBM MQ Explorer can connect to queue managers on other servers by using a server-connection channel

1. Create a server-connection channel on remote server queue manager to allow remote administration

   Example command:

   ```
   DEF CHL(SYSTEM.ADMIN.SVRCONN) CHLTYPE(SVRCONN) MCAUSER(MQADMIN)
   ```

   Where **`MQADMIN`** is the IBM MQ administrator user ID

2. Create a listener to accept incoming network connections

3. Start the listener

4. On the local IBM MQ Explorer:

   A. Right-click **Queue Managers**

   B. Click **Add remote queue manager**

   C. Identify the remote queue manager

Implementing distributed queuing © Copyright IBM Corporation 2017

*Figure 6-33. Managing a remote queue manager with IBM MQ Explorer*

You can use MQ Explorer to manage a queue manager on a remote server. However, the queue manager on the remote server must first be configured to allow remote management. The first three steps in this figure list the configuration actions that must be made on the remote queue manager.

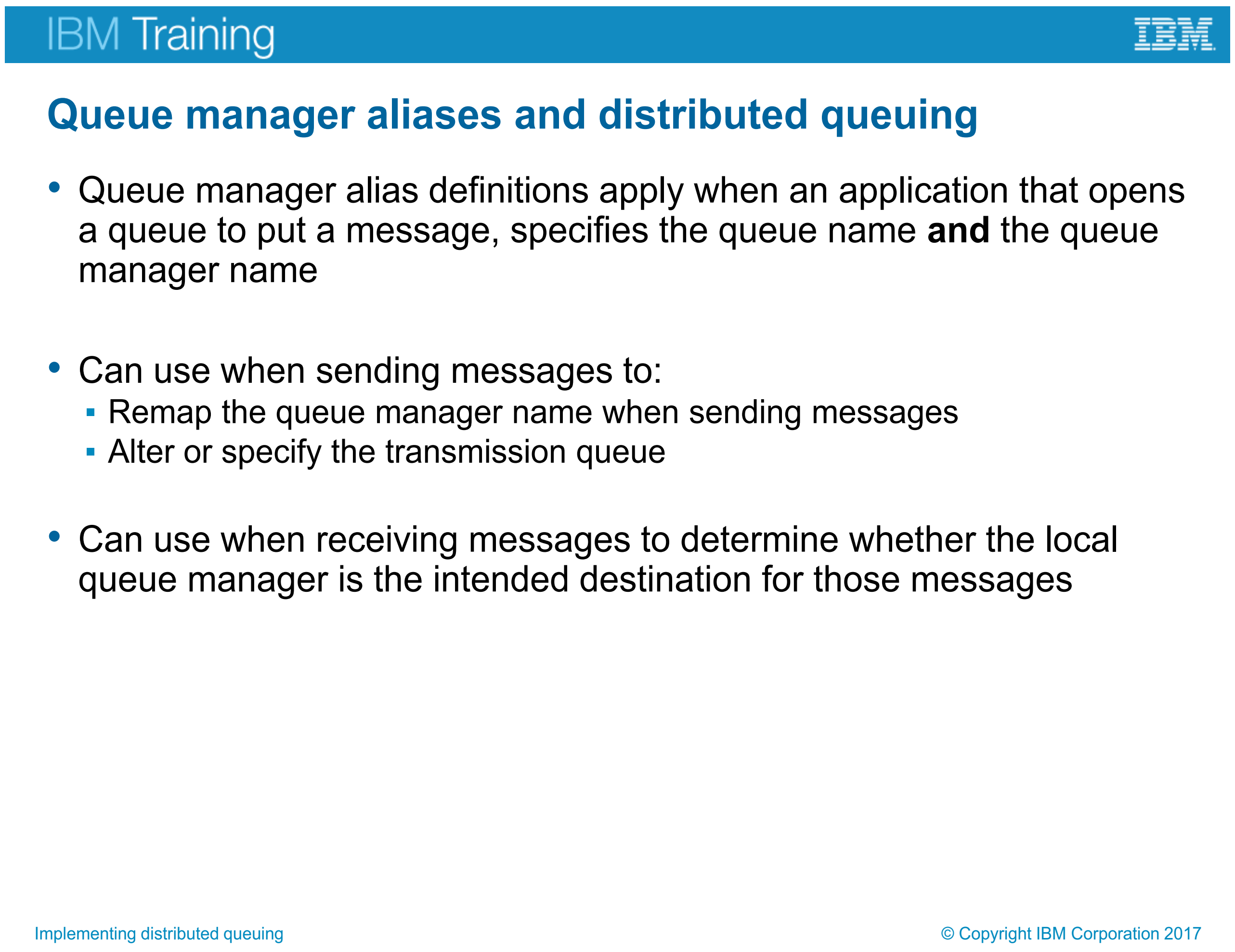
To run the listener in background with no hang-up, type:

```
nohup runmqlsr -t tcp -p port -m QMgrName 2>&1 &
```

To verify that the listener is running on Linux, type:

```
ps -ef | grep runmqlsr
```

After you complete the first three steps, you can add the remote queue manager to MQ Explorer for management.



IBM Training

## Queue manager aliases and distributed queuing

- 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
- Can use when sending messages to:
  - Remap the queue manager name when sending messages
  - Alter or specify the transmission queue
- Can use when receiving messages to determine whether the local queue manager is the intended destination for those messages

Implementing distributed queuing

© Copyright IBM Corporation 2017

*Figure 6-34. Queue manager aliases and distributed queuing*

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.

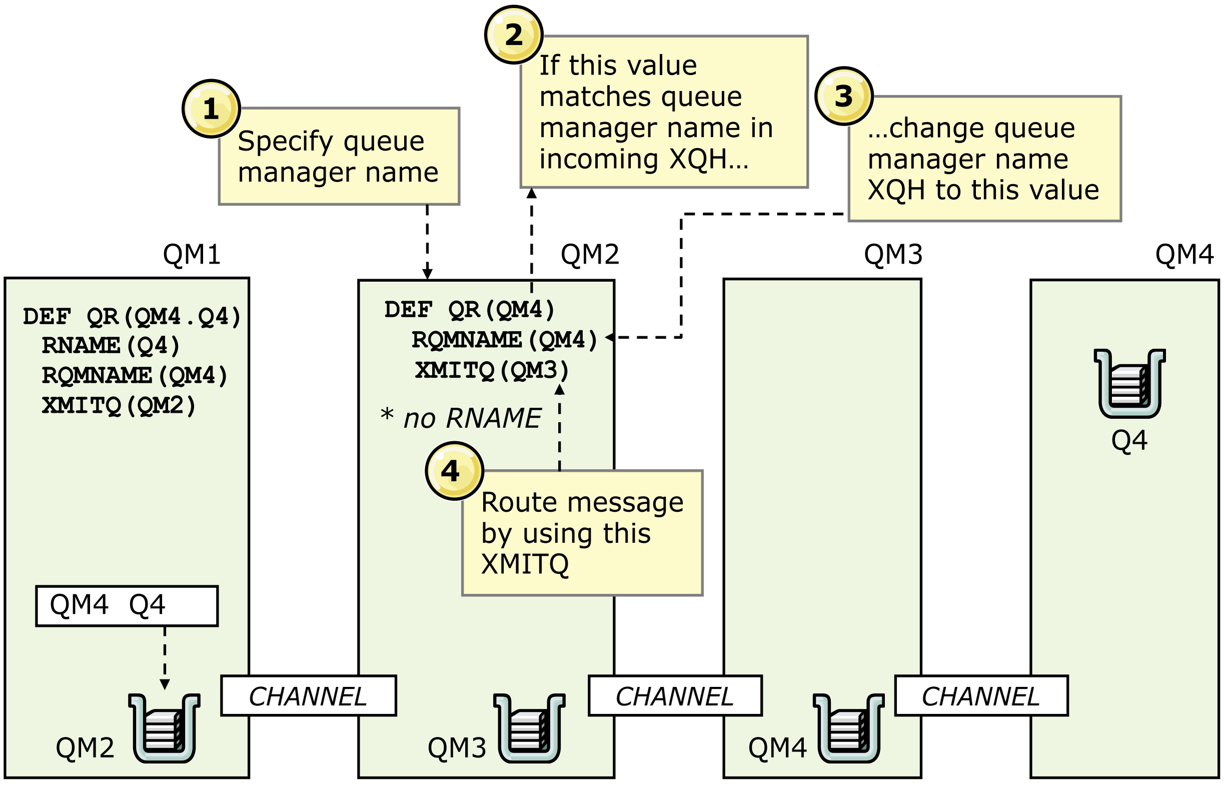
Queue manager alias definitions have three uses:

- 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

Aliases are used to provide a quality of service for messages. By using a queue manager alias, a system administrator can alter the name of a target queue manager without changing the applications. The system administrator can alter the route to a destination queue manager, or define a route that passes the message through a number of other queue managers (multi-hopping). The reply-to queue alias provides a quality of service for replies.

IBM Training

## Using a queue manager alias


1 Specify queue manager name
2 If this value matches queue manager name in incoming XQH…
3 …change queue manager name XQH to this value
QM1
DEF QR(QM4.Q4) RNAME(Q4) RQMNAME(QM4) XMITQ(QM2)
QM4 Q4
QM2
CHANNEL
QM2
DEF QR(QM4) RQMNAME(QM4) XMITQ(QM3)
* no RNAME
4 Route message by using this XMITQ
QM3
CHANNEL
QM3
QM4
CHANNEL
QM4
Q4

Implementing distributed queuing © Copyright IBM Corporation 2017

*Figure 6-35. Using a queue manager alias*

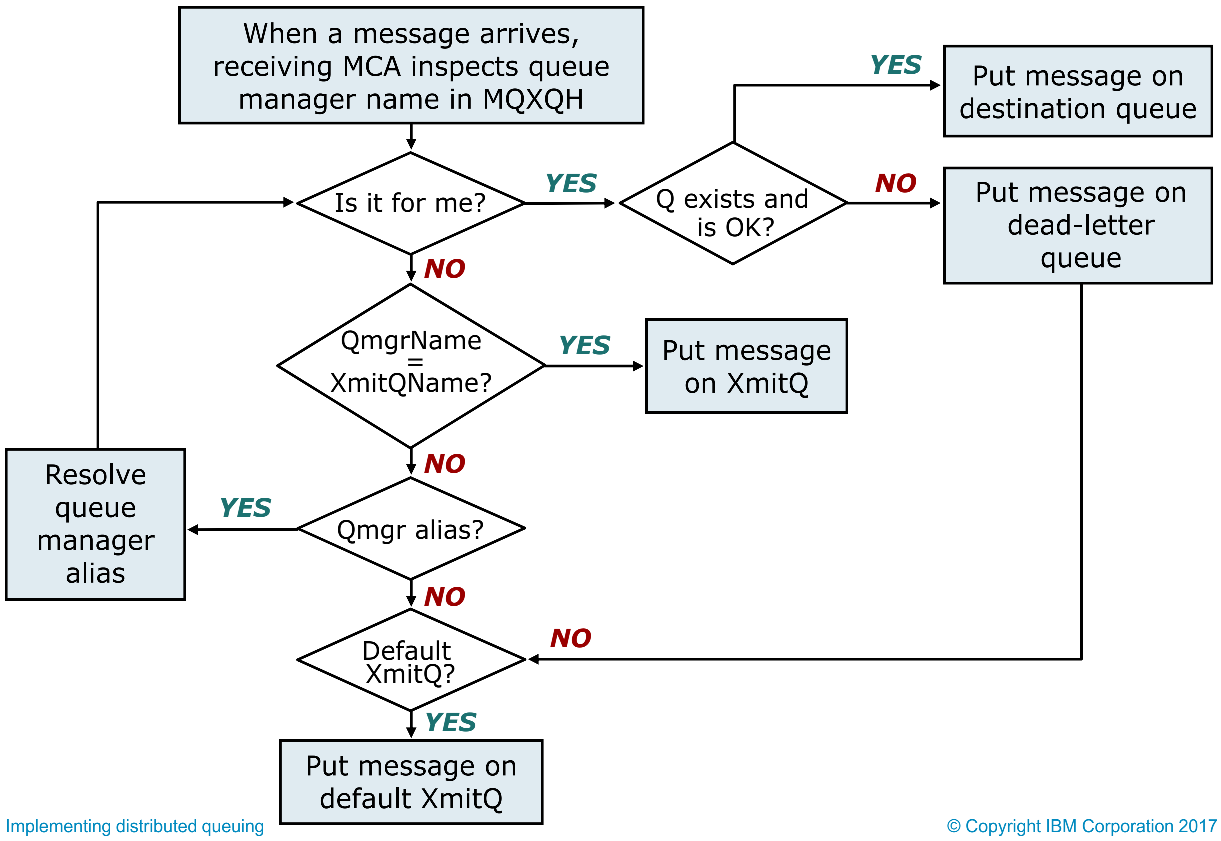
Queue manager aliases and reply-to queue aliases are created by using the **DEFINE QREMOTE** without an **RNAME** attribute, as shown in the figure. These definitions do not define real queues; the queue manager uses them to resolve physical queue names, queue manager names, and transmission queues.

By using default transmission queues and queue manager aliases, a message can be delivered through successive queue managers in a larger network. For example, to enable a message that originates at queue manager QM1 to be delivered to queue manager QM4, do the following steps:

- On QM2, create a transmission queue that is called QM3 and define QM4 as a queue manager alias that specifies QM3 as the transmission queue to use.
- On QM3, create a transmission queue called QM4.

IBM Training

## When a message arrives at a queue manager


When a message arrives, receiving MCA inspects queue manager name in MQXQH
Is it for me?
YES
Q exists and is OK?
YES
Put message on destination queue
NO
Put message on dead-letter queue
NO
QmgrName = XmitQName?
YES
Put message on XmitQ
NO
Qmgr alias?
YES
Resolve queue manager alias
NO
Default XmitQ?
NO
YES
Put message on default XmitQ

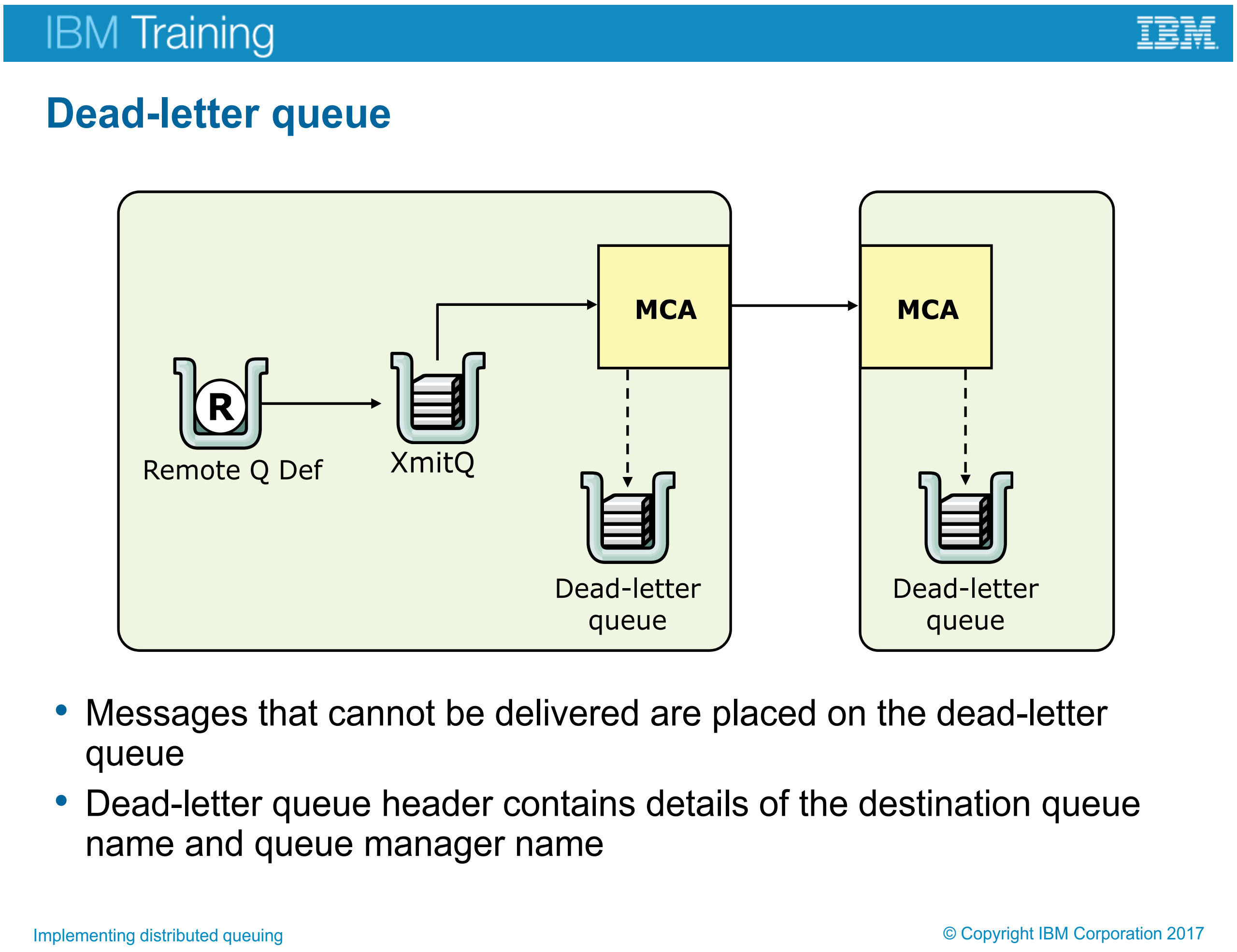
Implementing distributed queuing © Copyright IBM Corporation 2017

*Figure 6-36. When a message arrives at a queue manager*

During message processing, it is expected that the message can reach its target. As shown in this process flow, undeliverable messages are put on the queue manager's dead-letter queue.

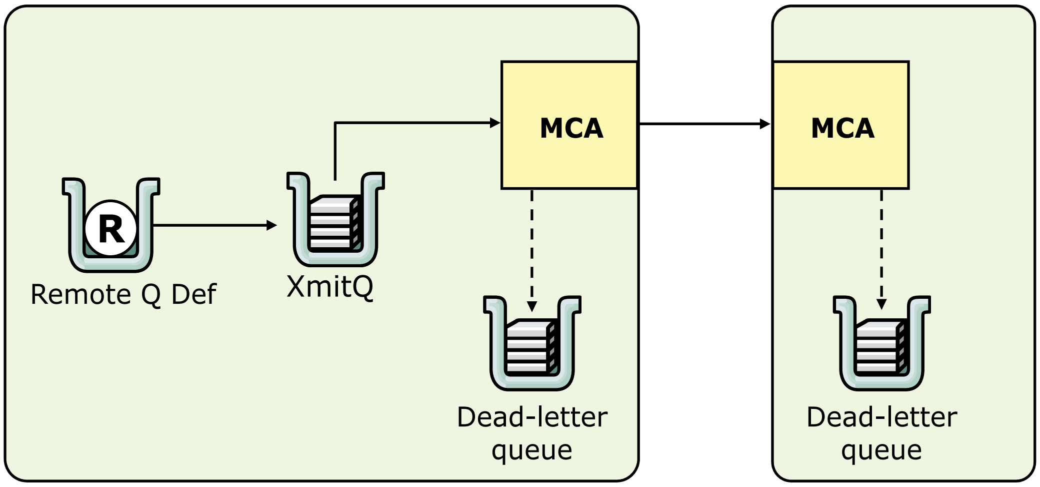
When a message arrives, the receiving MCA inspects the queue manager name in the transmission queue header (MQXQH) to determine whether the message matches its name. If it does, the queue manager checks whether the destination queue exists and whether it can accept messages. If it exists and it is running, the queue manager puts the message on the queue. If the destination queue does not exist or is not running, the message is put on the dead-letter queue.

If the destination queue manager name does not match, the destination queue manager name is compared to the transmission queue. If a match exists, the message is put on the transmission queue. If no match exists, the message is put on the dead-letter queue.



IBM Training

## Dead-letter queue



- Messages that cannot be delivered are placed on the dead-letter queue
- Dead-letter queue header contains details of the destination queue name and queue manager name

Implementing distributed queuing © Copyright IBM Corporation 2017

*Figure 6-37. Dead-letter queue*

If a message cannot be delivered, it is put on the dead-letter queue at the receiving end of a message channel. A message-retry at the receiving end of a channel might be useful if the problem is only temporary.

When a problem that relates to a message is detected asynchronously, an exception report is generated if one is requested, and the report is sent to the specified reply-to queue. The *Feedback* field in the message descriptor of the report message indicates the reason for the report. The original message is put on the dead-letter queue unless MQRO_DISCARD_MSG is requested as a report option.

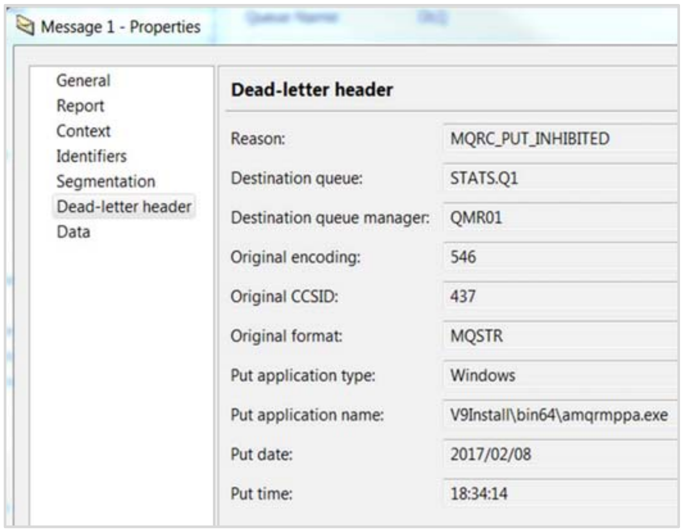
If the message is persistent and it cannot be put on the dead-letter queue, the channel is stopped and the message remains on the transmission queue. If the message is nonpersistent, it is discarded and the channel remains open.

Always create a dead-letter queue, and specify a dead-letter queue when you create a queue manager.

IBM Training

## Dead-letter header (MQDLH)

- Prefixes application message data of messages on dead-letter queue
- Applications that put messages directly on dead-letter queue must prefix message data with an MQDLH structure, and initialize fields with appropriate values
- Queue manager does not require that an MQDLH structure is present, or that valid values are specified for the fields
- Fields include:
  - Reason why message was put on dead-letter queue
  - Name of original destination queue and queue manager
  - Date and time message was put on queue
- Can examine the dead-letter header in IBM MQ Explorer by viewing the **Dead-letter header** message properties



**Dead-letter header**

| | |
|---|---|
| Reason: | MQRC_PUT_INHIBITED |
| Destination queue: | STATS.Q1 |
| Destination queue manager: | QMR01 |
| Original encoding: | 546 |
| Original CCSID: | 437 |
| Original format: | MQSTR |
| Put application type: | Windows |
| Put application name: | V9Install\bin64\amqrmppa.exe |
| Put date: | 2017/02/08 |
| Put time: | 18:34:14 |

Implementing distributed queuing

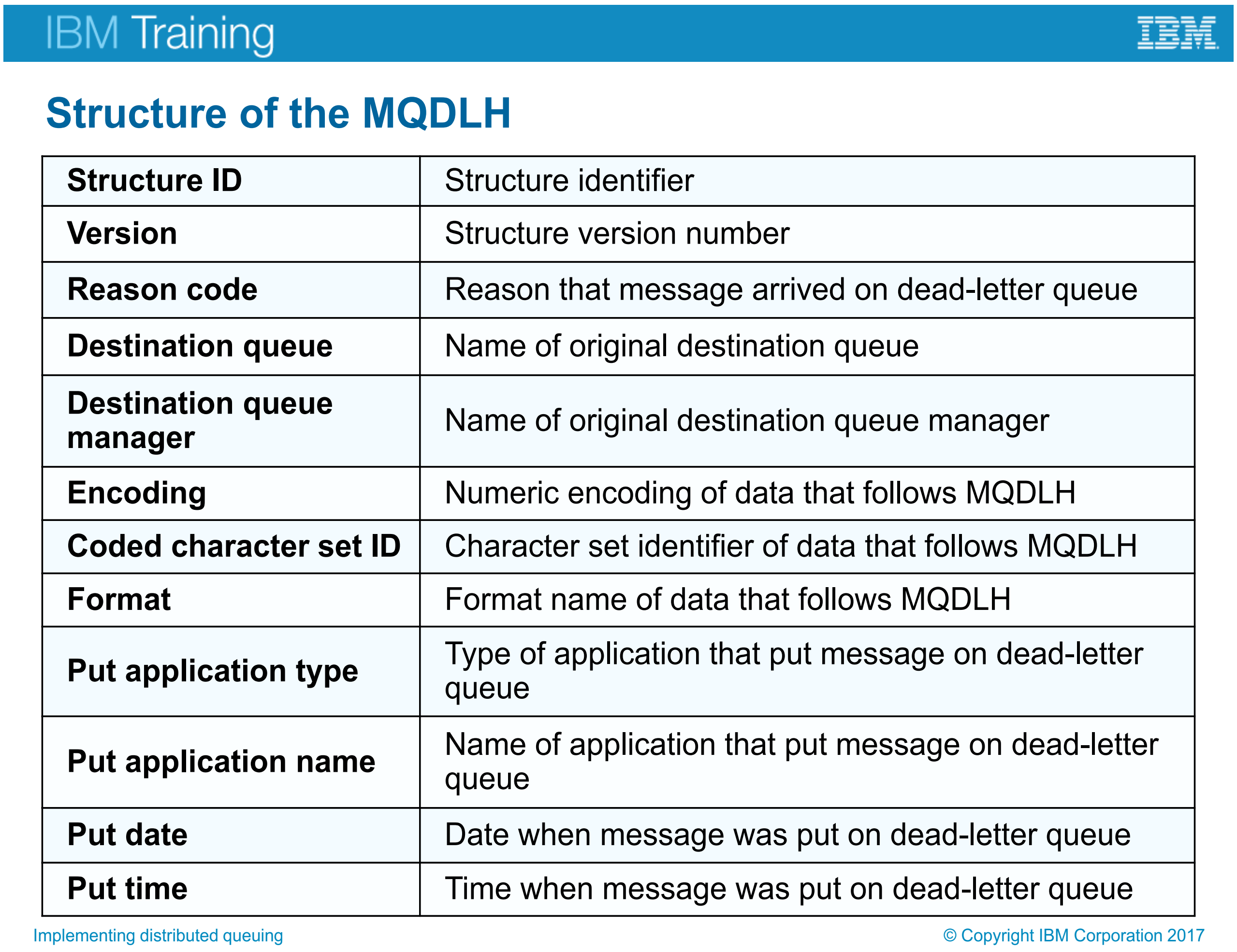
*Figure 6-38. Dead-letter header (MQDLH)*

The MQ dead-letter queue header (MQDLH) is the information that prefixes the application message data of the messages on the dead-letter queue.

Applications that put messages directly on the dead-letter queue must prefix message data with an MQDLH structure, and initialize fields with appropriate values. However, the queue manager does not require an MQDLH structure, or that valid values are specified for the fields.

Fields in the MQDLH structure include:

- The reason why the message was put on the dead-letter queue
- The name of the original destination queue and queue manager
- The date and time the message was put on the dead-letter queue



IBM Training

## Structure of the MQDLH

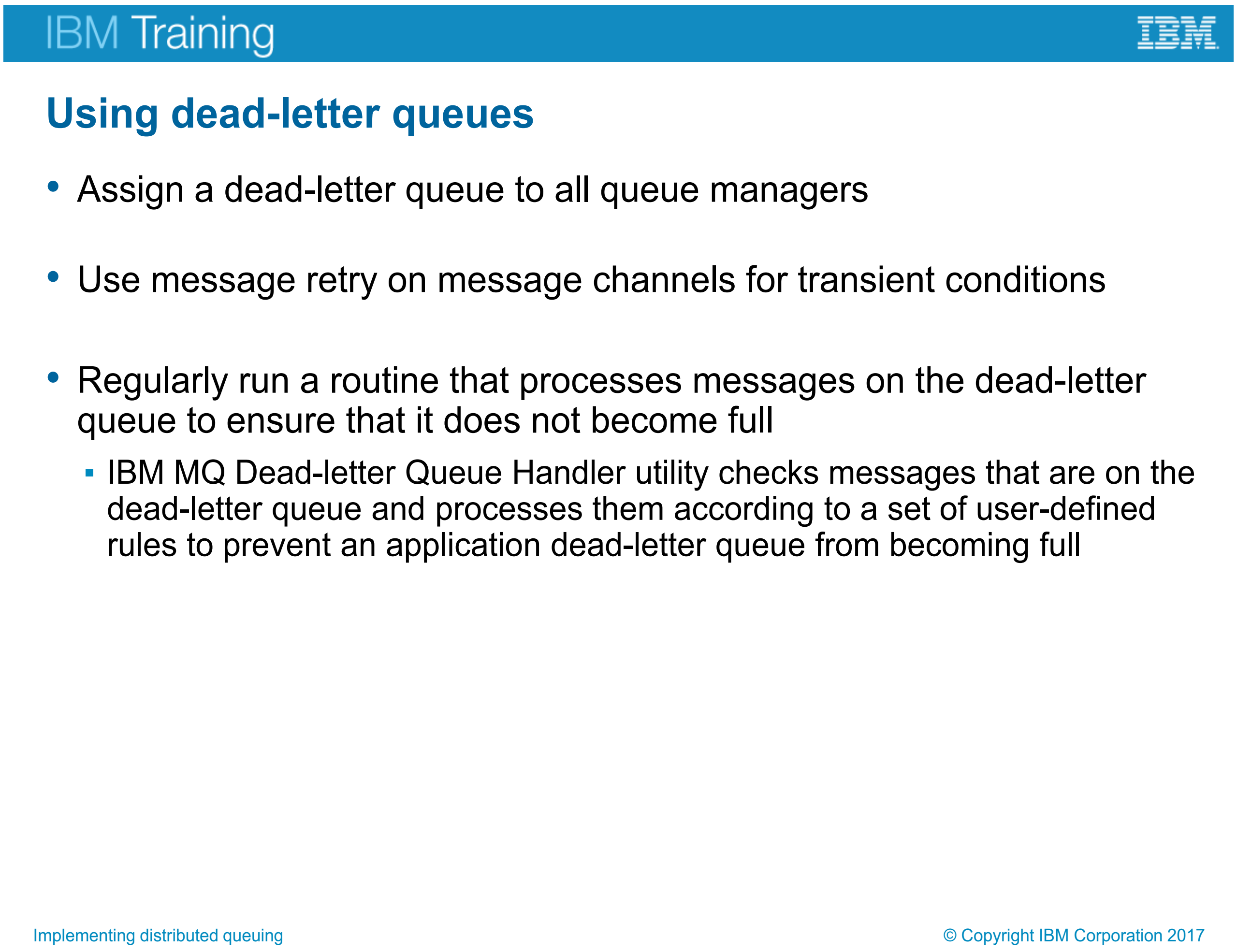
| | |
|---|---|
| **Structure ID** | Structure identifier |
| **Version** | Structure version number |
| **Reason code** | Reason that message arrived on dead-letter queue |
| **Destination queue** | Name of original destination queue |
| **Destination queue manager** | Name of original destination queue manager |
| **Encoding** | Numeric encoding of data that follows MQDLH |
| **Coded character set ID** | Character set identifier of data that follows MQDLH |
| **Format** | Format name of data that follows MQDLH |
| **Put application type** | Type of application that put message on dead-letter queue |
| **Put application name** | Name of application that put message on dead-letter queue |
| **Put date** | Date when message was put on dead-letter queue |
| **Put time** | Time when message was put on dead-letter queue |

Implementing distributed queuing © Copyright IBM Corporation 2017

*Figure 6-39. Structure of the MQDLH*

This figure summarizes the fields in the MQDLH.

See the MQ product documentation for a detailed description of all fields in the MQDLH.



IBM Training

## Using dead-letter queues

- Assign a dead-letter queue to all queue managers
- Use message retry on message channels for transient conditions
- Regularly run a routine that processes messages on the dead-letter queue to ensure that it does not become full
  - IBM MQ Dead-letter Queue Handler utility checks messages that are on the dead-letter queue and processes them according to a set of user-defined rules to prevent an application dead-letter queue from becoming full

Implementing distributed queuing

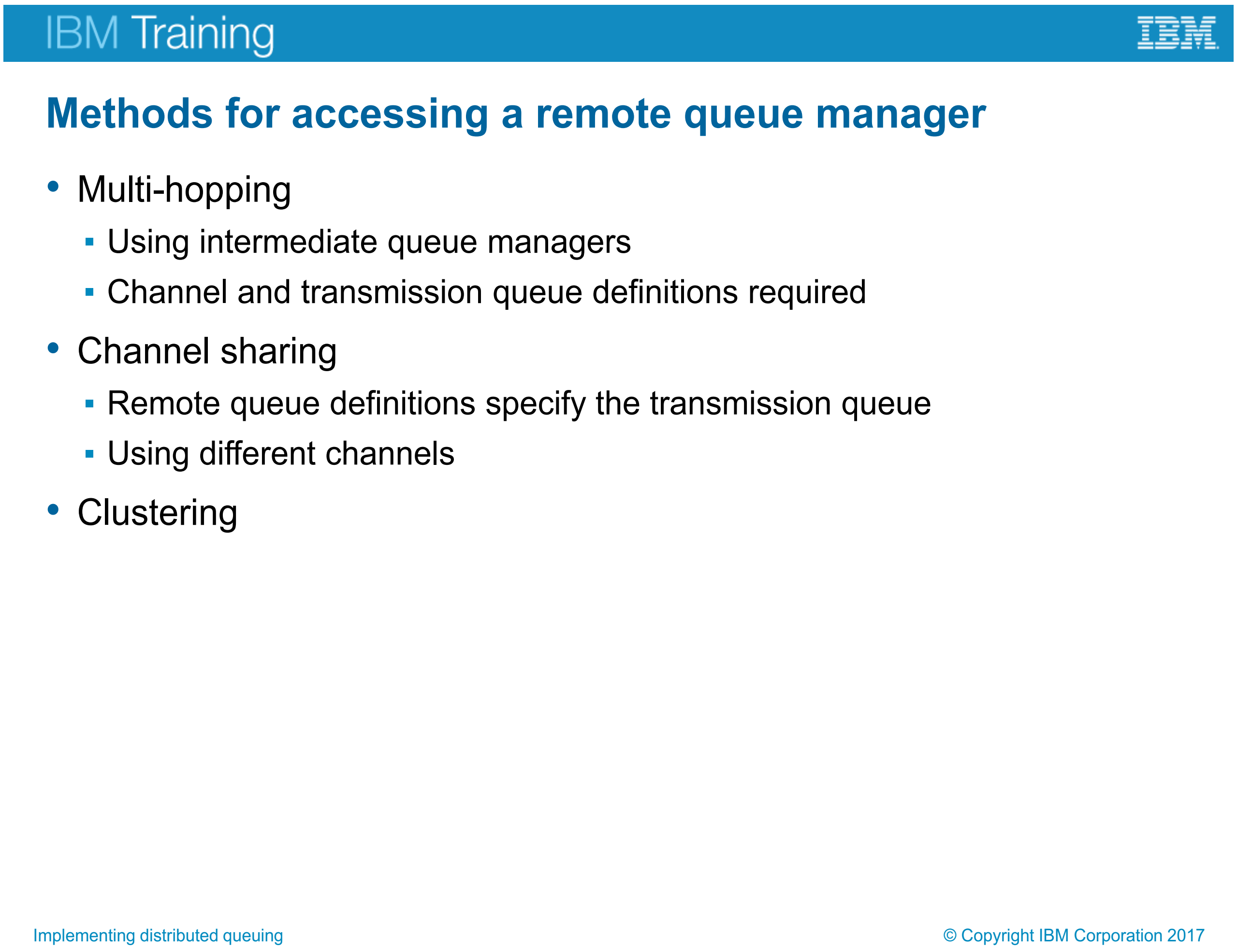
© Copyright IBM Corporation 2017

*Figure 6-40. Using dead-letter queues*

This figure lists some guidelines for using dead-letter queues.

- Create a dead-letter queue on all queue managers.
- Use message-retry on message channels to allow for transient conditions.
- Consider the combination of report options that implements “return to sender” as an alternative to putting a message on the dead-letter queue. The use of the “return to sender” function might mean that some messages that are targeted for the dead-letter queue might be returned.
- Do not allow an application dead-letter queue to become full.
- You can use the MQ Dead-letter Queue Handler to prevent the dead-letter queue from becoming full. The dead-letter queue handler is a stand-alone utility that checks the messages on the dead-letter queue and processes them according to a set of rules. The arrival of a message on the dead-letter queue can trigger the dead-letter queue handler.

  The MQ Dead-letter Queue Handler is covered in detail in course WM213/ZM213, *IBM MQ V9 Advanced System Administration*.
- If no reasonable retry option exists, forward the message to an application dead-letter queue.



IBM Training

## Methods for accessing a remote queue manager

- Multi-hopping
  - Using intermediate queue managers
  - Channel and transmission queue definitions required
- Channel sharing
  - Remote queue definitions specify the transmission queue
  - Using different channels
- Clustering

Implementing distributed queuing

© Copyright IBM Corporation 2017

*Figure 6-41. Methods for accessing a remote queue manager*

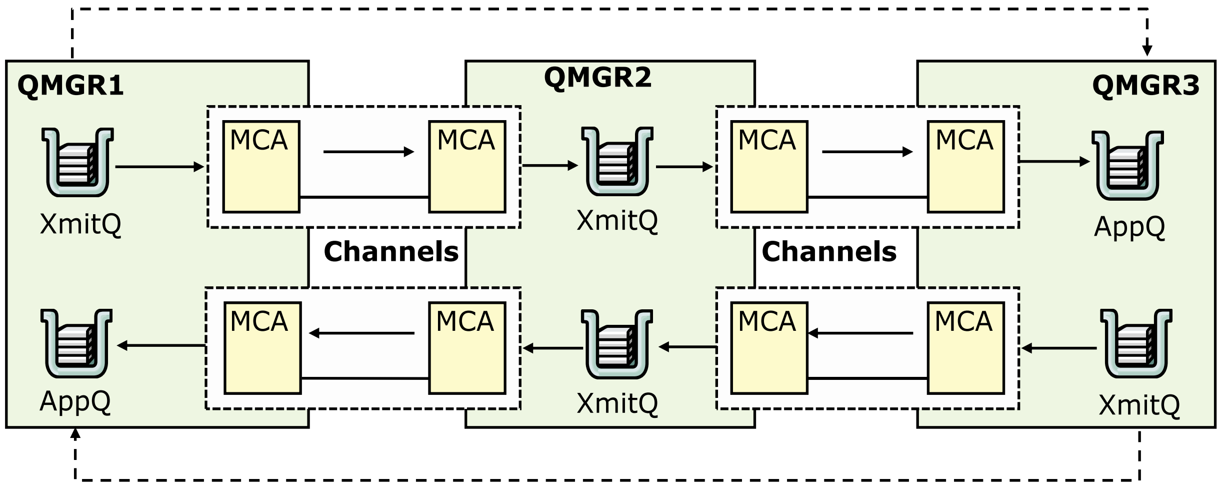
You might not always have one channel between each source and target queue manager. The other methods for accessing a remote queue manager are multi-hopping, channel sharing, and clustering.

Multi-hopping, channel sharing, and channel sharing by using different channels are described in detail on the next pages.

Clustering is described later in this course.

IBM Training

## Multi-hopping



Pass through one or more intermediate queue managers when no direct communication link exists between the source queue manager and the target queue manager

Implementing distributed queuing

© Copyright IBM Corporation 2017

*Figure 6-42. Multi-hopping*

When no direct communication link exists between the source queue manager and the target queue manager, it is possible to pass through one or more intermediate queue managers. This configuration is known as a *multi-hopping*.

In this configuration, you must define channels between all the queue managers and transmission queues on the intermediate queue managers.

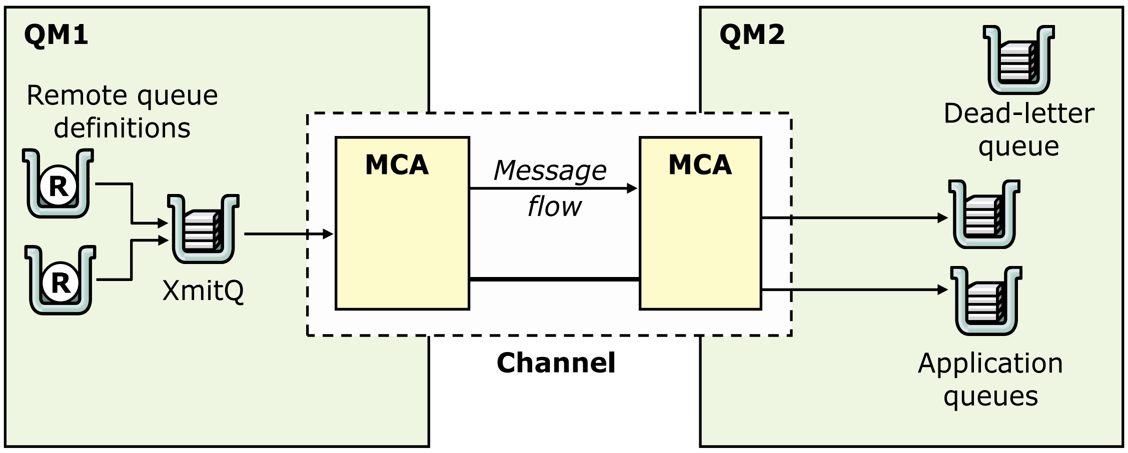
The example shows three queue managers. To send messages to the destination queue on QMGR3 from QMGR1, an intermediate queue manager, QMGR2, is used.

Messages are sent from QMGR1 to the transmission queue on QMGR2, and then on to the destination queue on QMGR3.

Channel definitions exist between QMGR1 and QMGR2, and between QMGR2 and QMGR3.

IBM Training

## Channel sharing



All messages from all applications that address queues at the same remote location send their messages through the same transmission queue

Implementing distributed queuing

*Figure 6-43. Channel sharing*

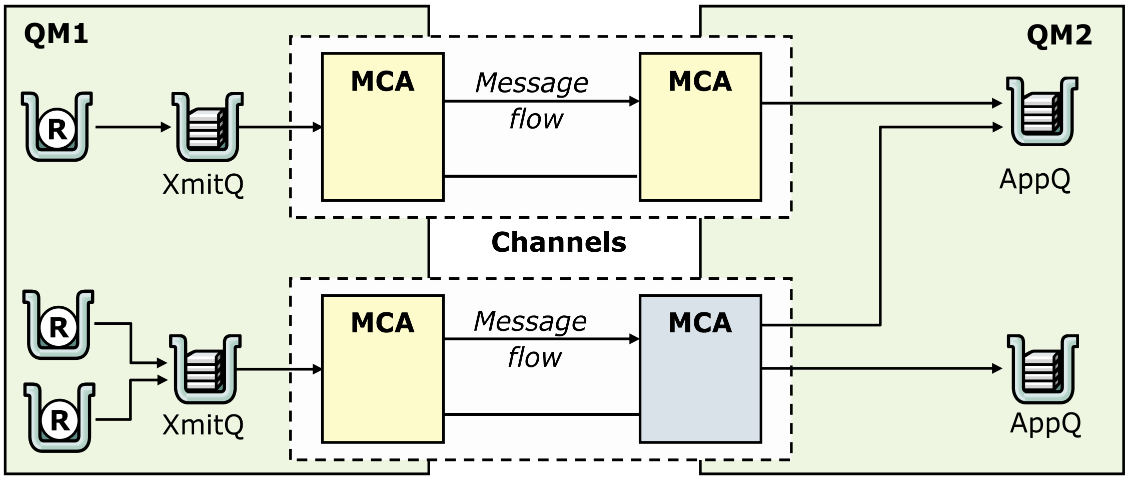
An application designer has the choice of forcing applications to specify the remote queue manager name and the queue name, or creating a *remote queue definition* for each remote queue. This definition holds the remote queue manager name, the queue name, and the name of the transmission queue.

Either way, all messages from all applications that address queues at the same remote location are sent through the same transmission queue. This configuration is known as *channel-sharing*.

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

IBM Training

## 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 queue managers

Implementing distributed queuing

© Copyright IBM Corporation 2017

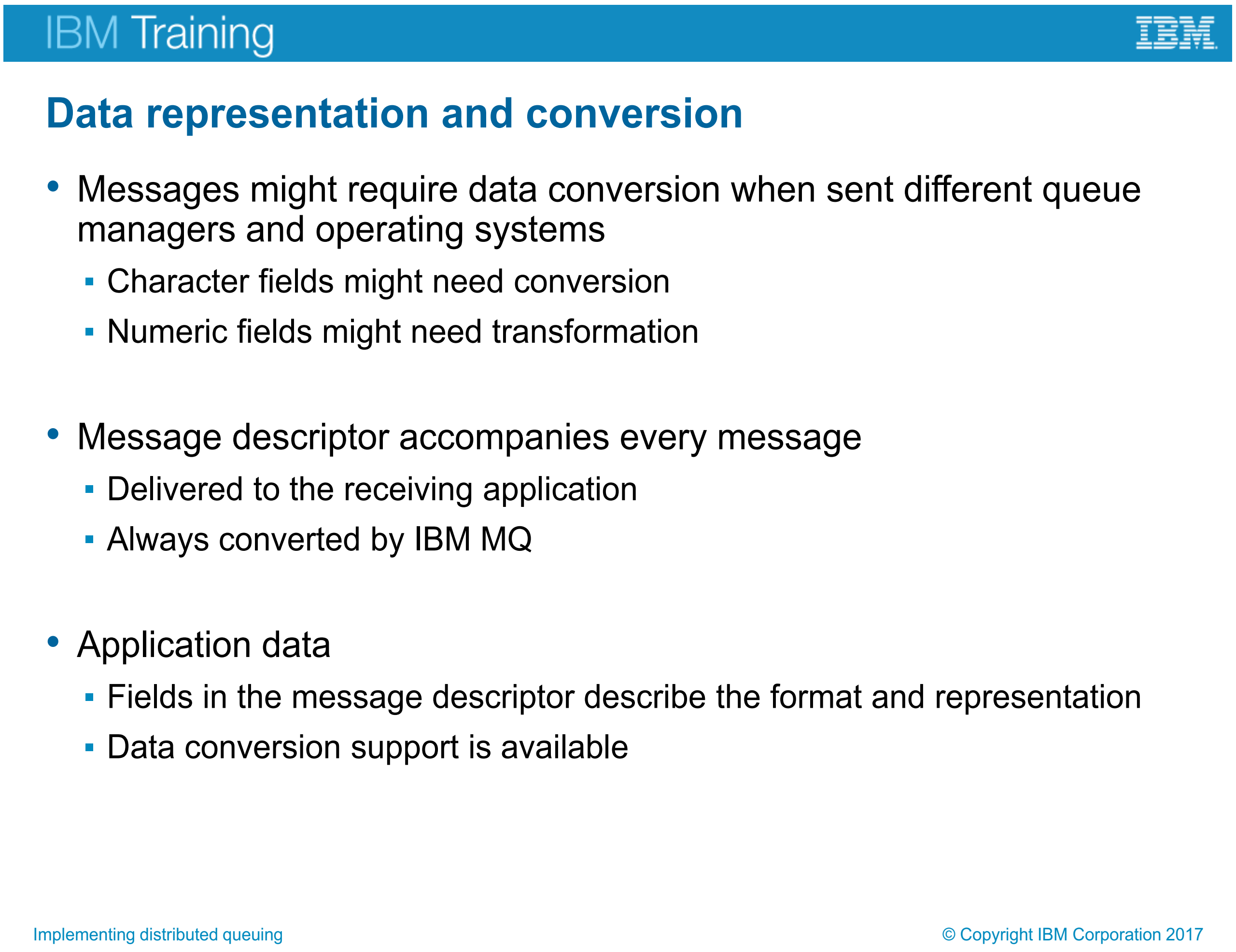
*Figure 6-44. Using different channels*

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

To set up a second channel, you must first define a second channel and transmission queue. Then, create a remote queue definition that specifies the location and the new transmission queue name. Your applications can then use either channel, but the messages are still delivered to the same target queues.

The figure shows multiple channels between two queue managers. The sending queue manager, QM1, has two channels with a transmission queue for each channel. Messages are sent to the receiving queue manager, QM2, by using both 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 does not use the remote queue definitions. Remote queue definitions make the location of queues and the transmission queue not apparent 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 changing your applications.



IBM Training

## Data representation and conversion

- Messages might require data conversion when sent different queue managers and operating systems
  - Character fields might need conversion
  - Numeric fields might need transformation
- Message descriptor accompanies every message
  - Delivered to the receiving application
  - Always converted by IBM MQ
- Application data
  - Fields in the message descriptor describe the format and representation
  - Data conversion support is available

Implementing distributed queuing © Copyright IBM Corporation 2017

*Figure 6-45. Data representation and conversion*

Application data might need to be converted to the character set and the encoding that another application requires when the queue managers or applications are on different operating systems and hardware.

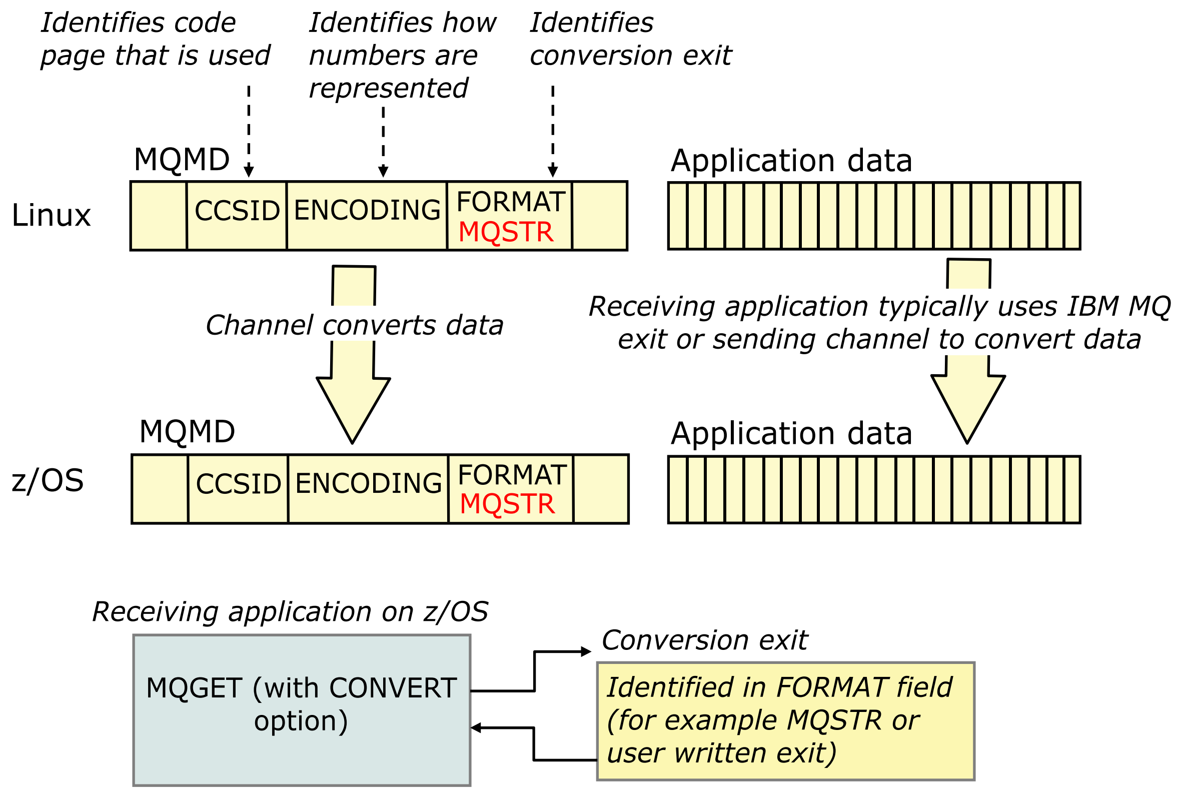
- Some character fields might require conversion from one character set to another.
- Some numeric fields might require transformation, such as byte reversal for integers.

An MQMD accompanies every message and is delivered to the receiving application with the application data. The MQMD is always converted to the representation of the destination system by all queue managers.

Application data might need to be converted to the character set and the encoding that another application requires when the queue managers or applications are on different operating systems and hardware.

IBM Training

## Data conversion example


Identifies code page that is used
Identifies how numbers are represented
Identifies conversion exit
MQMD
Linux
CCSID
ENCODING
FORMAT
MQSTR
Application data
Channel converts data
Receiving application typically uses IBM MQ exit or sending channel to convert data
MQMD
z/OS
CCSID
ENCODING
FORMAT
MQSTR
Application data
Receiving application on z/OS
MQGET (with CONVERT option)
Conversion exit
Identified in FORMAT field (for example MQSTR or user written exit)

Implementing distributed queuing © Copyright IBM Corporation 2017

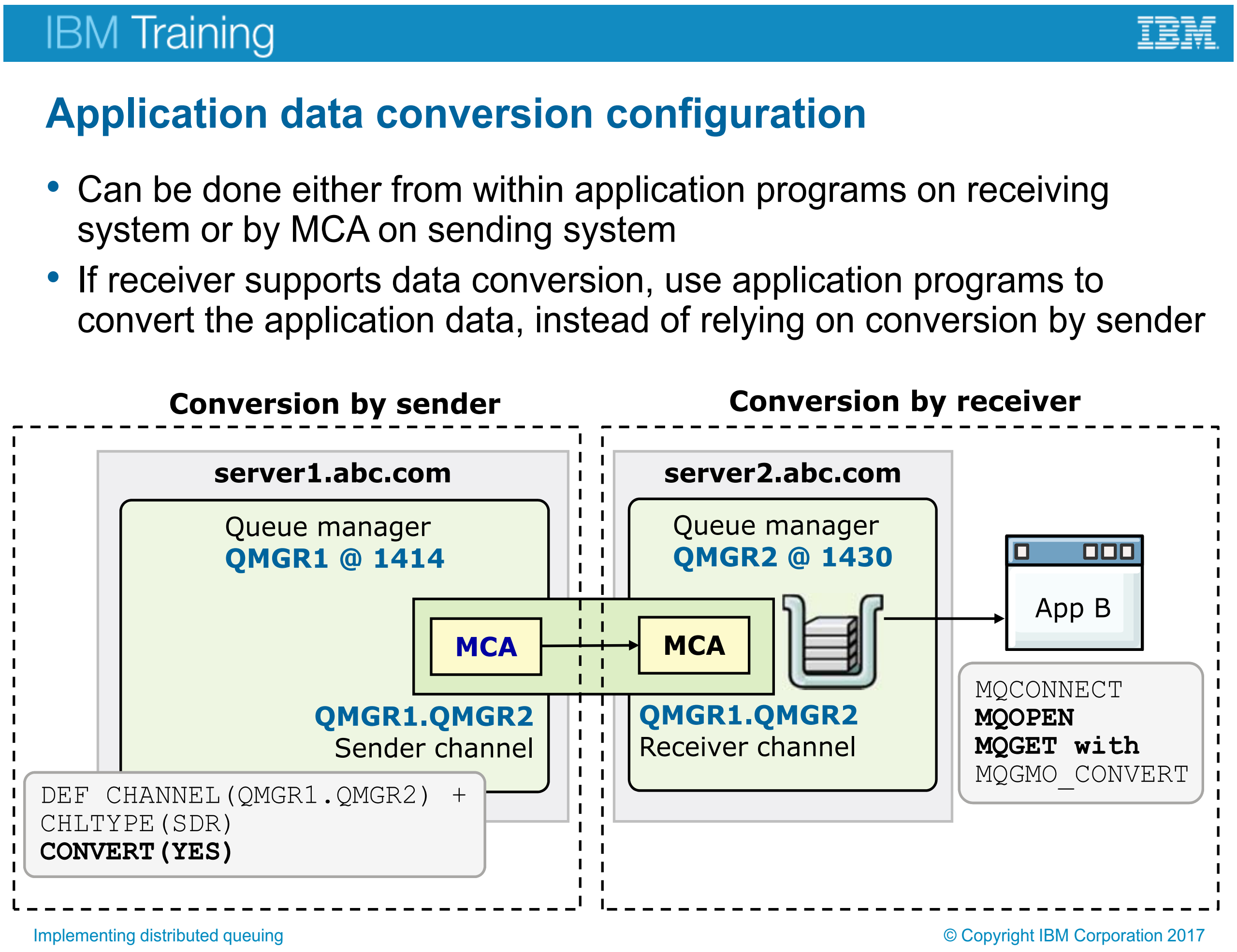
*Figure 6-46. Data conversion example*

IBM MQ products support the coded character sets that are provided by the underlying operating system. When you create a queue manager, the queue manager coded character set ID (CCSID) used is based on the underlying environment.

When MQ sends messages between queue managers, you must consider the message code page and encoding, especially when MQ sends data between countries and operating systems.

The MQMD is always converted to the representation of the destination system by all queue managers.

The queue manager cannot convert messages in user-defined formats from one coded character set to another. If you need to convert data in a user-defined format, you must supply a data-conversion exit for each such format.



IBM Training

## Application data conversion configuration

- Can be done either from within application programs on receiving system or by MCA on sending system
- If receiver supports data conversion, use application programs to convert the application data, instead of relying on conversion by sender

```
DEF CHANNEL(QMGR1.QMGR2) +
CHLTYPE(SDR)
CONVERT(YES)
```

```
MQCONNECT
MQOPEN
MQGET with
MQGMO_CONVERT
```

Implementing distributed queuing © Copyright IBM Corporation 2017

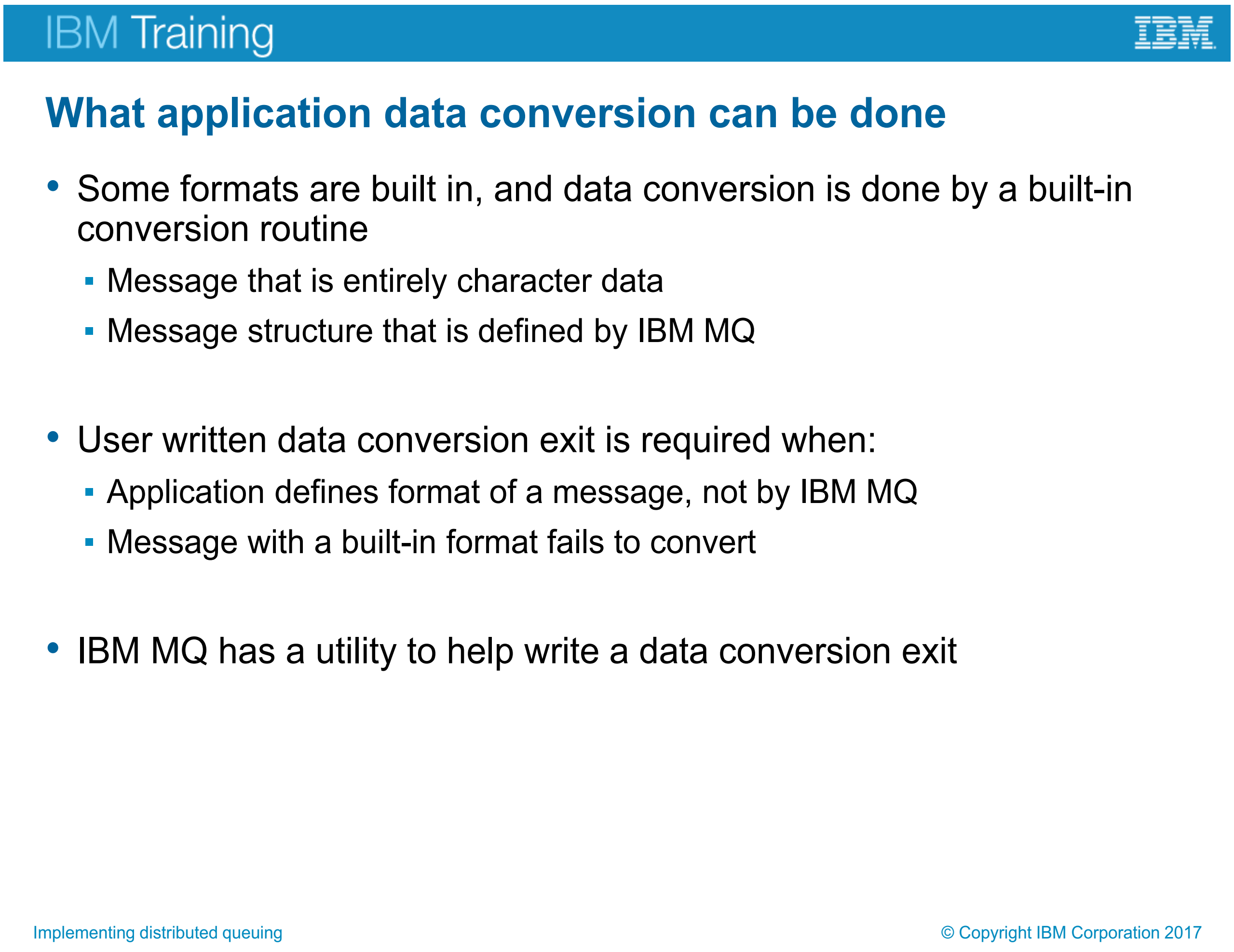
*Figure 6-47. Application data conversion configuration*

When a message channel is started between two queue managers, the two MCAs determine what conversion is required and decide which of them does it.

Application data can be converted at the sending queue manager or at the receiving queue manager. The type of conversion depends on the message format that is specified in the format field of the MQMD.

If you need the sending MCA to convert the application data, set the **CONVERT** channel attribute to **YES**. The conversion is done at the sending queue manager for certain built-in formats and for user-defined formats if a user exit is supplied.

The receiving queue manager can convert application message data for both built-in and user-defined formats. If the reading application specifies the MQGMO_CONVERT get message option, the conversion is done during the processing of an MQGET call.



IBM Training

## What application data conversion can be done

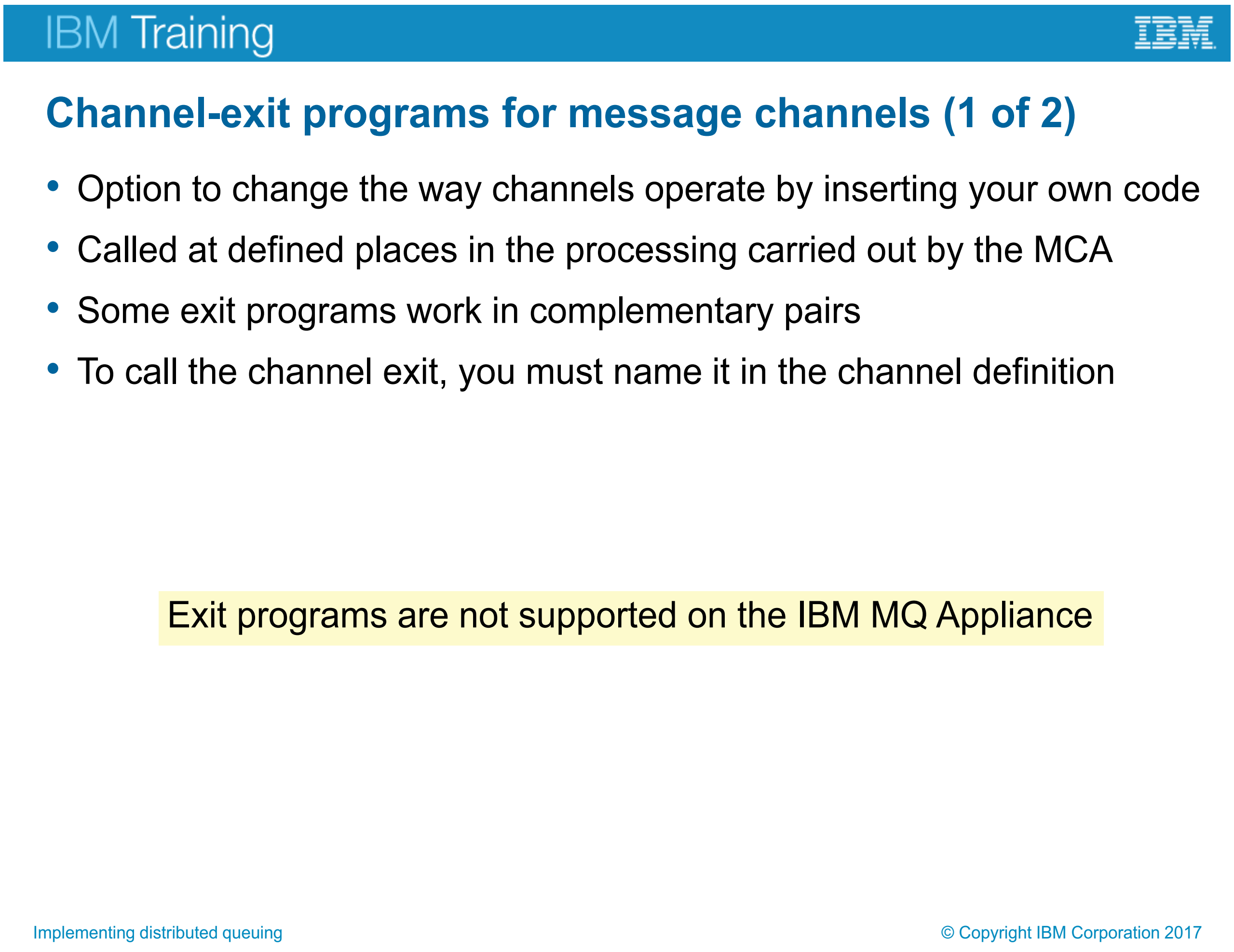
- Some formats are built in, and data conversion is done by a built-in conversion routine
  - Message that is entirely character data
  - Message structure that is defined by IBM MQ
- User written data conversion exit is required when:
  - Application defines format of a message, not by IBM MQ
  - Message with a built-in format fails to convert
- IBM MQ has a utility to help write a data conversion exit

Implementing distributed queuing © Copyright IBM Corporation 2017

*Figure 6-48. What application data conversion can be done*

A built-in conversion routine can convert a message that consists entirely of characters or a message with a structure that is defined by MQ.

If the application defines the format of a message, a user-written data conversion exit must convert the data. The **`crtmqcvx`** utility is supplied with MQ for help in developing a data conversion exit.



IBM Training

## Channel-exit programs for message channels (1 of 2)

- Option to change the way channels operate by inserting your own code
- Called at defined places in the processing carried out by the MCA
- Some exit programs work in complementary pairs
- To call the channel exit, you must name it in the channel definition

Exit programs are not supported on the IBM MQ Appliance

Implementing distributed queuing © Copyright IBM Corporation 2017

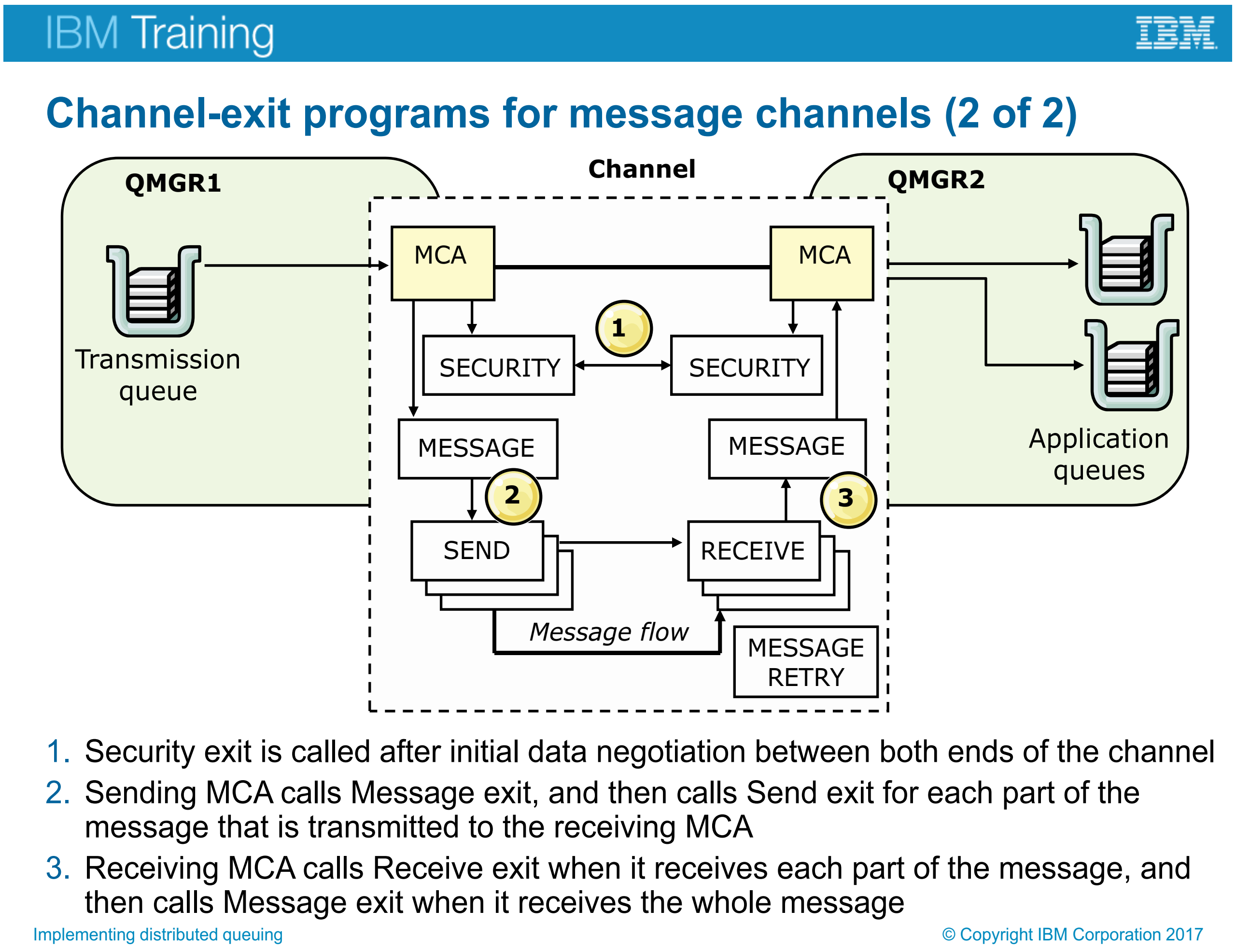
*Figure 6-49. Channel-exit programs for message channels (1 of 2)*

MQ calls channel-exit programs at defined places in the processing carried out by the MCA. Four types of channel exits exist:

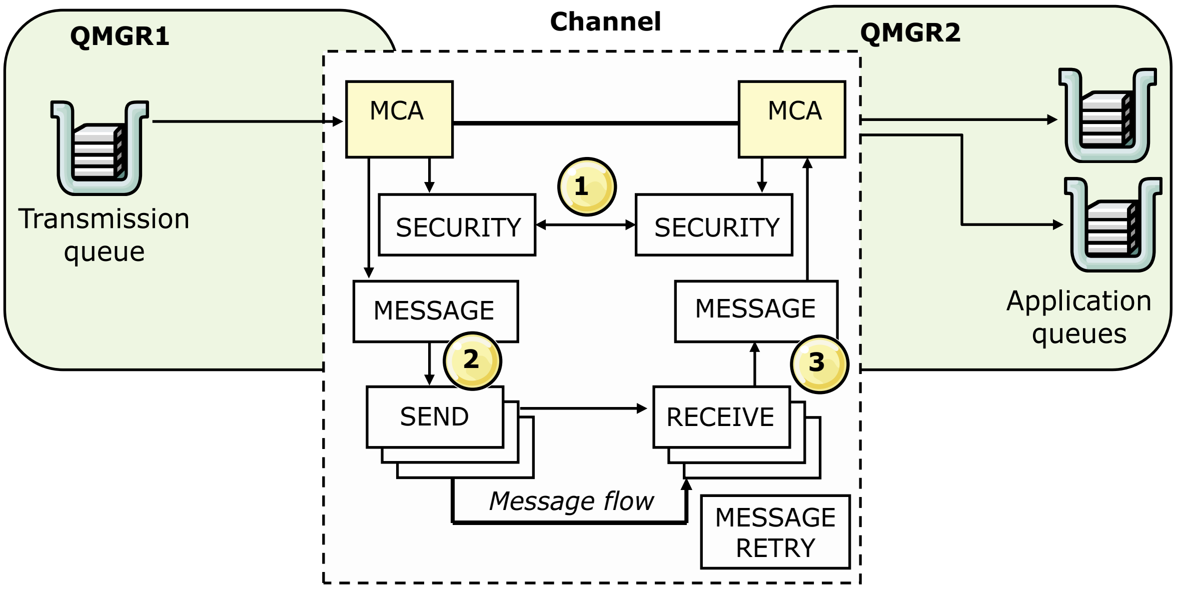
- Security exit for security checking, such as authentication of the remote server
- Message exit for operations on the message, for example, encryption before transmission
- Send and Receive exits for operations on split messages, for example, data compression and decompression
- Message-retry exit that is used when a problem arises in putting the message to the destination

Before you write a channel exit, consider other MQ capabilities such as SSL for encryption, or channel authorization and connection authorization for authentication.

Channel exits are described in detail in WM213, *IBM MQ V9 Advanced System Administration*.


IBM Training

## Channel-exit programs for message channels (2 of 2)



1. Security exit is called after initial data negotiation between both ends of the channel
2. Sending MCA calls Message exit, and then calls Send exit for each part of the message that is transmitted to the receiving MCA
3. Receiving MCA calls Receive exit when it receives each part of the message, and then calls Message exit when it receives the whole message

Implementing distributed queuing © Copyright IBM Corporation 2017

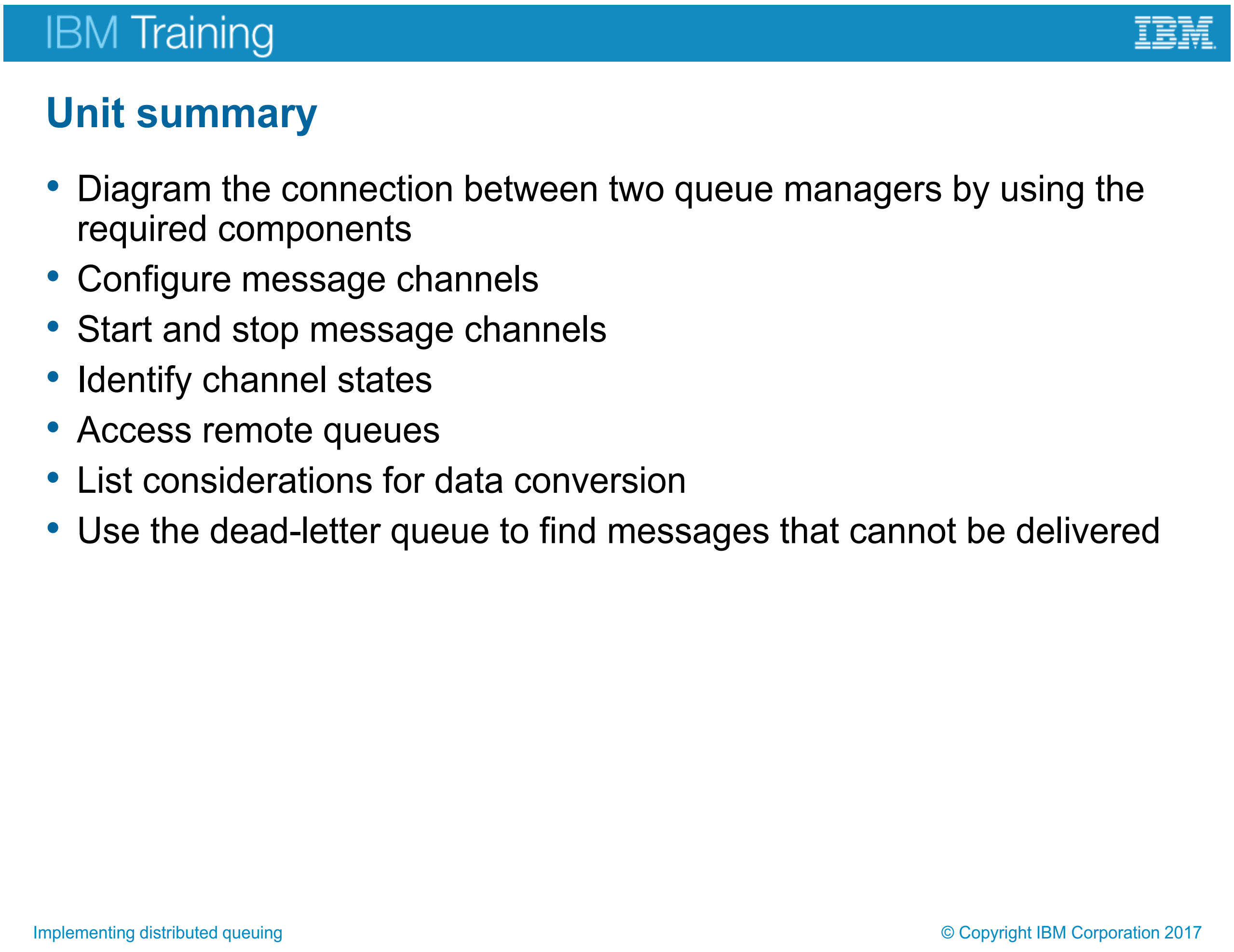
*Figure 6-50. Channel-exit programs for message channels (2 of 2)*

The sequence of processing for channel-exit programs is shown in the figure:

1. A security exit is 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 sending MCA calls the Message exit, 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 is received.



IBM Training

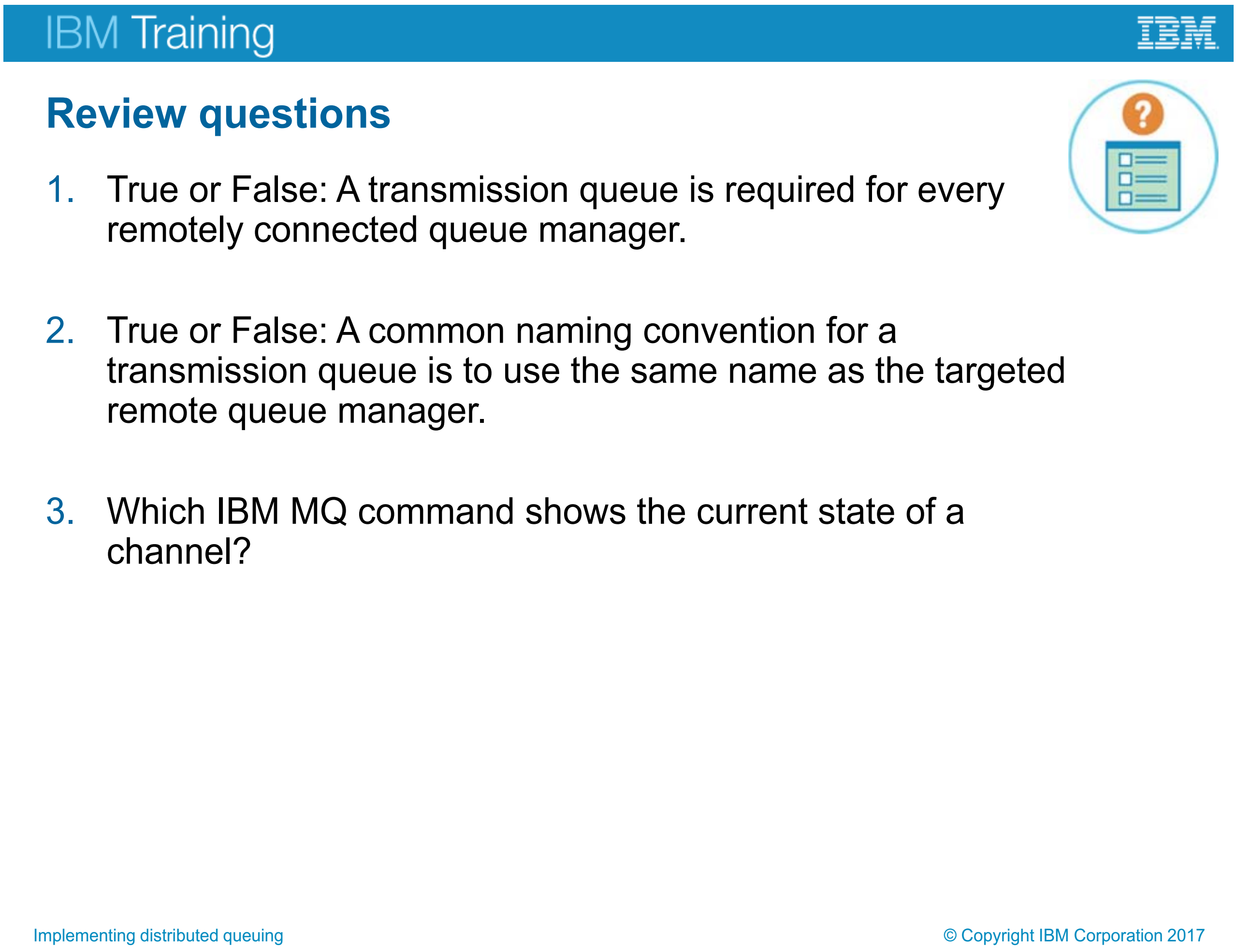
## Unit summary

- Diagram the connection between two queue managers by using the required components
- Configure message channels
- Start and stop message channels
- Identify channel states
- Access remote queues
- List considerations for data conversion
- Use the dead-letter queue to find messages that cannot be delivered

Implementing distributed queuing

© Copyright IBM Corporation 2017

*Figure 6-51. Unit summary*



IBM Training

## Review questions

1. True or False: A transmission queue is required for every remotely connected queue manager.

2. True or False: A common naming convention for a transmission queue is to use the same name as the targeted remote queue manager.

3. Which IBM MQ command shows the current state of a channel?

Implementing distributed queuing © Copyright IBM Corporation 2017

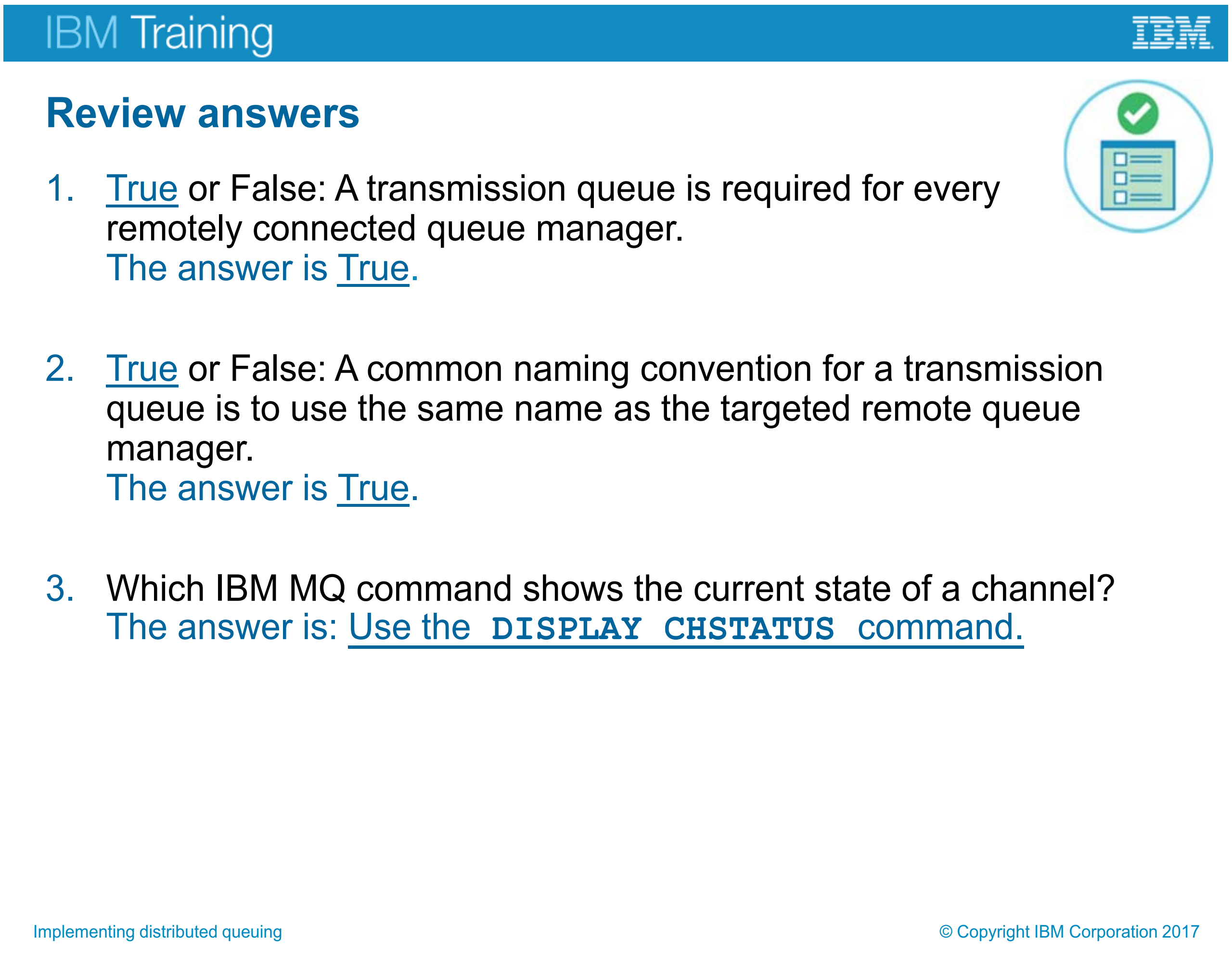
*Figure 6-52. Review questions*

Write you answers down here:

1.

2.

3.



IBM Training

IBM

## Review answers

1. True or False: A transmission queue is required for every remotely connected queue manager.
   The answer is True.

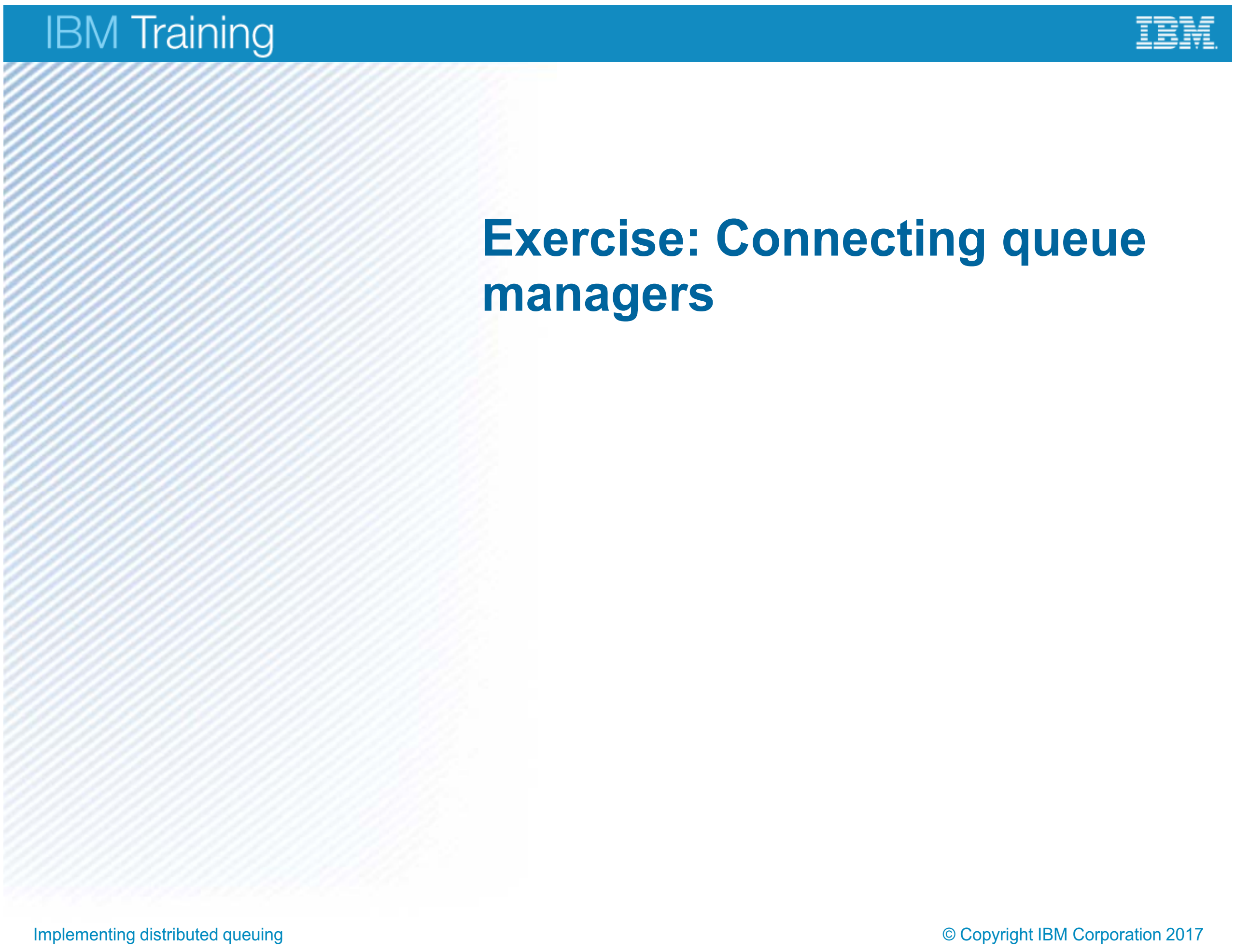
2. True or False: A common naming convention for a transmission queue is to use the same name as the targeted remote queue manager.
   The answer is True.

3. Which IBM MQ command shows the current state of a channel?
   The answer is: Use the **`DISPLAY CHSTATUS`** command.

Implementing distributed queuing

© Copyright IBM Corporation 2017

*Figure 6-53. Review answers*


IBM Training

IBM

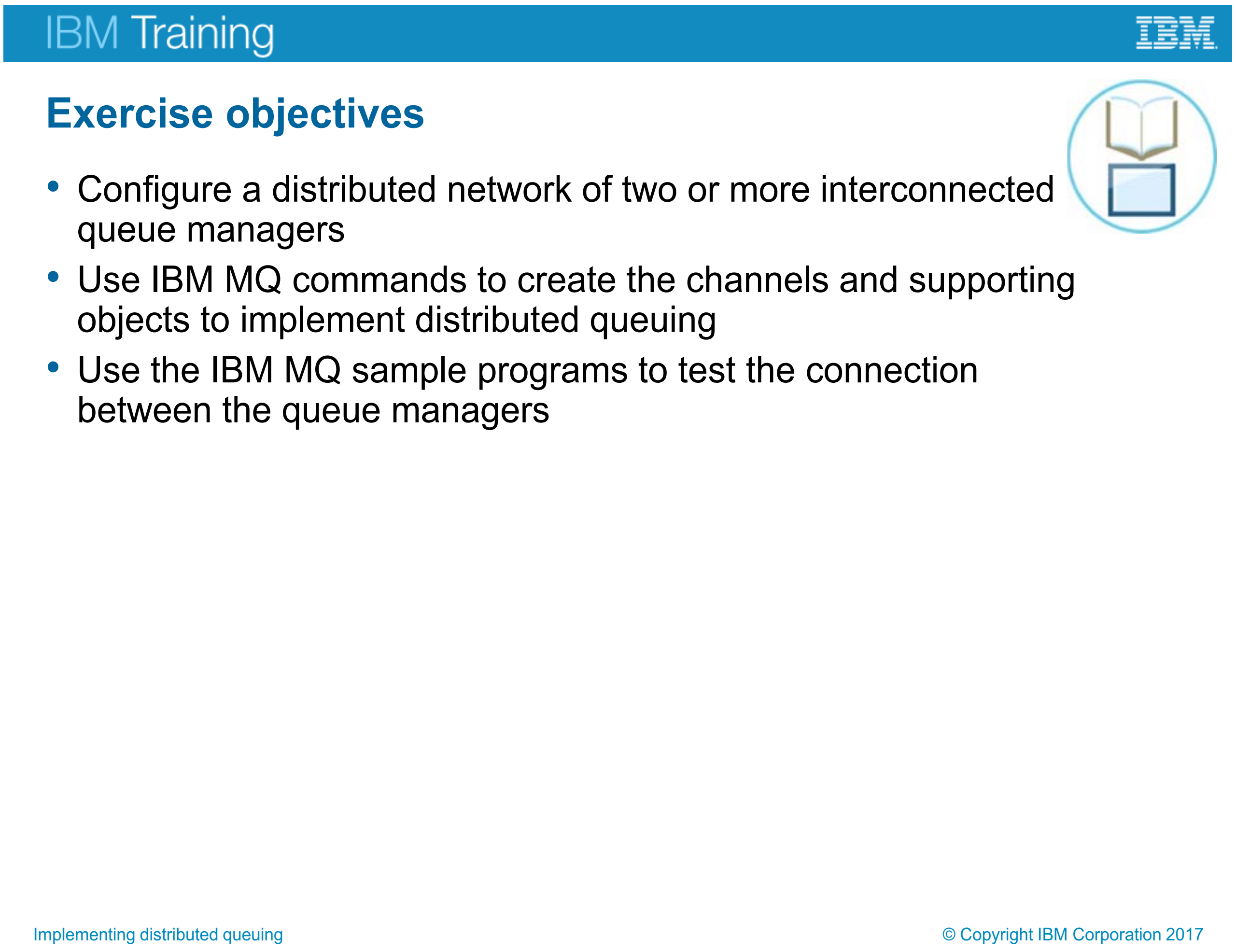
## Exercise: Connecting queue managers

Implementing distributed queuing

© Copyright IBM Corporation 2017

*Figure 6-54. Exercise: Connecting queue managers*

In this exercise, you create channels between two queue managers, and one of the queue managers simulates a remote queue manager on another system. You use the IBM MQ sample programs to test the connection between the queue managers.



IBM Training

## Exercise objectives

- Configure a distributed network of two or more interconnected queue managers
- Use IBM MQ commands to create the channels and supporting objects to implement distributed queuing
- Use the IBM MQ sample programs to test the connection between the queue managers

Implementing distributed queuing

© Copyright IBM Corporation 2017

*Figure 6-55. Exercise objectives*

See the *Course Exercise Guide* for detailed instructions.