

Course Guide

IBM MQ V9 System Administration (using Windows for labs)

Course code WM153 / ZM153 ERC 1.0



April 2017 edition

Notices

This information was developed for products and services offered in the US.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

*IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
United States of America*

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some jurisdictions do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you provide in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to actual people or business enterprises is entirely coincidental.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the web at "Copyright and trademark information" at www.ibm.com/legal/copytrade.shtml.

© Copyright International Business Machines Corporation 2017.

This document may not be reproduced in whole or in part without the prior written permission of IBM.

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

Contents

Trademarks	xiii
Course description	xiv
Agenda	xvi
Unit 1. IBM MQ review	1-1
How to check online for course material updates	1-2
Unit objectives	1-3
What is IBM MQ?	1-4
IBM MQ features	1-5
The power of IBM MQ	1-6
IBM MQ components	1-7
IBM MQ functional overview	1-8
Messages	1-10
Message persistence	1-11
Queues	1-12
Queue manager	1-13
Queue manager terminology	1-14
Channels	1-15
Parallel processing	1-16
Asynchronous processing	1-17
Application programming interfaces	1-18
IBM MQ client	1-19
IBM MQ administration interfaces	1-20
Advantages of messaging with IBM MQ	1-21
Deployment options	1-23
IBM MQ Advanced	1-24
IBM MQ product information	1-25
IBM Knowledge Center for IBM MQ V9	1-26
Unit summary	1-27
Review questions	1-28
Review answers	1-29
Unit 2. IBM MQ installation and deployment options	2-1
Unit objectives	2-2
Deployment options	2-3
IBM MQ Appliance M2001	2-4
IBM MQ in the Cloud	2-5
IBM MQ Hypervisor editions	2-6
Supported Hypervisors (1 of 2)	2-7
Supported Hypervisors (2 of 2)	2-8
IBM MQ Advanced for PureApplication	2-9
IBM Message Hub on IBM Bluemix	2-10
IBM Message Connect on IBM Bluemix	2-11
Other Cloud options for IBM MQ	2-12
IBM MQ on-premises installation	2-13
Multiple version installations	2-14
Multiple version installation commands	2-16
Multiple version installation commands examples	2-17

Choosing an installation name	2-18
Choosing an installation path	2-20
IBM MQ release and support versions	2-21
Supported operating systems and hardware (1 of 2)	2-23
Supported operating systems and hardware (2 of 2)	2-24
Locating IBM MQ Fix Packs	2-25
IBM MQ SupportPacs	2-26
IBM MQ SupportPac examples	2-28
Choosing what to install	2-29
Preparing the system	2-30
IBM MQ on-premises installation	2-31
Finding more information	2-32
Uninstalling IBM MQ	2-33
Unit summary	2-34
Review questions	2-35
Review answers	2-36
Unit 3. Creating a queue manager and queues	3-1
Unit objectives	3-2
Administration interfaces	3-3
IBM MQ command modes	3-4
Creating a queue manager	3-6
Queue manager default and system objects	3-8
Queue manager logs	3-9
Naming IBM MQ objects	3-11
Basic queue manager control commands	3-12
Stopping a queue manager	3-13
Stopping and starting the IBM MQ service on Windows	3-15
Queue manager configuration file	3-16
Example queue manager configuration file stanzas	3-17
When do you need to edit the queue manager configuration file?	3-18
Using MQSC to configure IBM MQ objects	3-19
Starting the MQSC command interface	3-20
Stopping the MQSC command interface	3-22
MQSC command rules	3-23
MQSC command summary	3-24
Using filters in MQSC commands	3-25
MQSC filtering example	3-26
MQSC script files	3-27
Running MQSC script files	3-29
Defining a local queue	3-30
Defining a local queue examples	3-31
Defining other queue types	3-32
Clearing and deleting queues	3-34
Displaying queue manager and queue attributes	3-35
Display queue manager example	3-36
Displaying queue attributes example	3-37
Modifying attributes	3-38
Special queues	3-39
Defining a dead-letter queue	3-41
Unit summary	3-42
Review questions (1 of 2)	3-43
Review questions (2 of 2)	3-44
Review answers (1 of 2)	3-45
Review answers (2 of 2)	3-46
Exercise: Using commands to create queue managers and queues	3-47

Exercise objectives	3-48
Unit 4. Introduction to IBM MQ Explorer	4-1
Unit objectives	4-2
IBM MQ Explorer	4-3
Starting IBM MQ Explorer	4-4
Welcome page	4-5
Viewing installations of IBM MQ	4-6
IBM MQ Explorer default views	4-7
Creating a local queue manager (1 of 3)	4-8
Creating a local queue manager (2 of 3)	4-9
Creating a local queue manager (3 of 3)	4-10
Queue Manager content view	4-11
Changing queue manager properties	4-12
Queue manager property categories	4-13
Grouping queue managers	4-14
Creating a queue manager set (1 of 7)	4-15
Creating a queue manager set (2 of 7)	4-16
Creating a queue manager set (3 of 7)	4-17
Creating a queue manager set (4 of 7)	4-18
Creating a queue manager set (5 of 7)	4-19
Creating a queue manager set (6 of 7)	4-20
Creating a queue manager set (7 of 7)	4-21
Creating a local queue (1 of 3)	4-22
Creating a local queue (2 of 3)	4-23
Creating a local queue (3 of 3)	4-24
Queues content view	4-25
Queue status information	4-26
Shortcut icons	4-27
Viewing queue filters	4-28
Managing queue filters	4-29
Adding queue filters	4-30
Queue filter attributes	4-31
Comparing queues (1 of 2)	4-32
Comparing queues (2 of 2)	4-33
IBM MQ object definition tests	4-34
IBM MQ general tests	4-35
Running the IBM MQ tests	4-36
IBM MQ test results example	4-37
IBM MQ Explorer preferences	4-38
Context-sensitive help	4-39
Unit summary	4-40
Review questions	4-41
Review answers	4-42
Exercise: Using IBM MQ Explorer to create queue managers and queues	4-43
Exercise objectives	4-44
Unit 5. Testing the IBM MQ implementation	5-1
Unit objectives	5-2
IBM MQ Message Queue Interface (MQI)	5-3
Basic MQI calls	5-4
Basic MQI calls in use	5-5
Order of retrieving messages	5-6
Browsing messages in IBM MQ Explorer	5-7
Message properties	5-8
General message properties	5-9

Report message properties	5-10
Context message properties	5-11
Identifiers message properties	5-12
Segmentation message properties	5-13
Data message properties	5-14
Message property preferences in IBM MQ Explorer	5-15
Message property preferences: Examples	5-16
Distribution lists	5-17
Testing IBM MQ configuration	5-19
Testing with IBM MQ sample programs	5-20
Sample program example	5-22
Testing with IBM MQ Explorer	5-23
Testing with RFHUtil (SupportPac IH03)	5-24
Unit summary	5-25
Review questions	5-26
Review answers	5-27
Exercise: Using IBM MQ sample programs to test the configuration	5-28
Exercise objectives	5-29
Unit 6. Implementing distributed queuing	6-1
Unit objectives	6-2
Message-queuing styles	6-3
Point-to-point messaging example	6-4
Distributed queuing components overview (1 of 2)	6-5
Distributed queuing components overview (2 of 2)	6-7
Queue definitions for distributed queuing	6-9
Using IBM MQ Explorer to define a remote queue	6-11
Using IBM MQ Explorer to define a transmission queue	6-12
Transmission queue header	6-13
Choosing a transmission queue	6-14
Example of using a default transmission queue	6-16
Message channel	6-17
Message channel types	6-18
Sender to receiver message channel	6-19
Requester to server message channel	6-20
Requester to sender message channel	6-21
DEFINE CHANNEL command	6-22
Defining channels example	6-23
Defining channels in IBM MQ Explorer (1 of 2)	6-24
Defining channels in IBM MQ Explorer (2 of 2)	6-25
Channel control parameters	6-26
Minimum channel definitions for remote communication	6-27
Distributed queuing configuration example	6-28
Starting a message channel	6-29
Displaying channel status	6-31
DISPLAY CHSTATUS examples	6-32
Channel states (1 of 2)	6-33
Channel states (2 of 2)	6-35
Channel initiators and listeners	6-36
Controlling the listener process	6-37
Listener object	6-38
Reasons why a channel might fail to start	6-39
Managing a remote queue manager with IBM MQ Explorer	6-40
Queue manager aliases and distributed queuing	6-41
Using a queue manager alias	6-42
When a message arrives at a queue manager	6-43

Dead-letter queue	6-44
Dead-letter header (MQDLH)	6-45
Structure of the MQDLH	6-46
Using dead-letter queues	6-47
Methods for accessing a remote queue manager	6-48
Multi-hopping	6-49
Channel sharing	6-50
Using different channels	6-51
Data representation and conversion	6-52
Data conversion example	6-53
Application data conversion configuration	6-54
What application data conversion can be done	6-55
Channel-exit programs for message channels (1 of 2)	6-56
Channel-exit programs for message channels (2 of 2)	6-57
Unit summary	6-58
Review questions	6-59
Review answers	6-60
Exercise: Connecting queue managers	6-61
Exercise objectives	6-62
Unit 7. IBM MQ clients	7-1
Unit objectives	7-2
IBM MQ client	7-3
IBM MQ client communication	7-4
Units of work and sync point	7-5
Units of work and sync point control	7-7
Sync point control on a base client	7-8
Extended transactional client	7-9
Configuring connections between the server and client	7-10
Defining an MQI channel	7-11
Defining server-connection channels in IBM MQ Explorer	7-12
Client configuration methods	7-13
Method 1: Using MQSERVER	7-14
Method 2: Client configuration file	7-15
Method 3: Using a CCDT	7-16
Using a CCDT example	7-17
Defining client-connection channels in IBM MQ Explorer	7-18
Accessing a CCDT	7-19
URL support for CCDT files	7-20
Precedence order for finding the CCDT	7-21
Weighted selection on CLNTCONN channels	7-22
MQI channel instances	7-24
Instance limits on SVRCONN channels	7-25
Auto-definition of channels	7-26
Testing client connectivity	7-27
Testing client-to-server connectivity with MQCONN	7-28
IBM MQ client limitations	7-29
IBM MQ client security	7-30
IBM MQ client security checks	7-31
IBM MQ redistributable client	7-32
Unit summary	7-33
Review questions	7-34
Review answers	7-35
Exercise: Connecting an IBM MQ client	7-36
Exercise objectives	7-37

Unit 8. Implementing trigger messages and monitors	8-1
Unit objectives	8-2
Using triggering to start IBM MQ applications	8-3
Application triggering	8-4
Components of triggering	8-5
Queue attributes that control triggering	8-6
‘Defining triggering properties in IBM MQ Explorer’	8-8
Initiation queues	8-9
Defining an IBM MQ process	8-10
Defining a process in IBM MQ Explorer	8-11
Application triggering conditions	8-12
Information in the trigger message	8-14
Trigger monitor	8-15
Defining a trigger monitor in IBM MQ Explorer	8-16
Trigger monitor service properties in IBM MQ Explorer	8-17
Starting the trigger monitor in IBM MQ Explorer	8-18
Testing a trigger monitor with an IBM MQ sample program	8-19
Monitoring trigger monitor status	8-20
Triggering and the dead-letter queue	8-21
Application triggering program problems	8-22
Unit summary	8-23
Review questions	8-24
Review answers	8-25
Exercise: Implementing a trigger monitor	8-26
Exercise objectives	8-27
Unit 9. Diagnosing problems	9-1
Unit objectives	9-2
Where is the message?	9-3
Missing message checklist	9-4
Finding the dead-letter reason (1 of 2)	9-5
Finding the dead-letter reason (2 of 2)	9-6
Checking IBM MQ reason codes	9-7
Checking for IBM MQ network problems	9-8
Checking message persistence	9-10
Checking message expiration	9-11
Uncommitted messages	9-12
Displaying the queue status	9-13
First Failure Support Technology (FFST)	9-14
FFST files	9-15
FFST report example	9-16
IBM MQ error logs	9-17
Example error log contents	9-18
IBM MQ AMQ messages	9-19
AMQ message example	9-20
Application activity trace	9-21
The mqat.ini file	9-22
The mqat.ini file	9-23
Subscribing to activity trace data	9-24
Activity trace topics and subscriptions	9-25
Example activity trace topic strings	9-27
Activity trace sample program	9-28
Tracing IBM MQ components	9-29
Trace commands	9-30
Enabling trace in IBM MQ Explorer	9-31
Example trace on Windows	9-32

Configuration files and problem determination	9-33
Stopping a queue manager manually	9-34
Stopping a queue manager manually on Windows	9-35
Stopping a queue manager manually on UNIX and Linux	9-36
Removing a queue manager manually	9-37
Channel problem determination	9-38
Remote administration failures	9-39
Channel triggering problems	9-40
Unit summary	9-41
Review questions	9-42
Review answers	9-43
Exercise: Running an IBM MQ trace	9-44
Exercise objectives	9-45
Unit 10. Implementing basic security in IBM MQ	10-1
Unit objectives	10-2
Security mechanisms	10-3
IBM MQ security implementations	10-4
Planning for security	10-5
IBM MQ access control overview	10-6
OAM installable service	10-7
Principals and groups	10-8
Access control with the OAM	10-9
OAM access control lists	10-10
Set or reset authorization	10-12
Display authorization	10-14
Create authorization report	10-15
Using MQSC to manage authorization	10-16
Using IBM MQ Explorer to manage authorization	10-17
Queue manager authorization in IBM MQ Explorer	10-18
Queue authorization in IBM MQ Explorer	10-19
Channel authorization in IBM MQ Explorer	10-20
Access control for IBM MQ control commands	10-21
Security and distributed queuing	10-22
Security authorization for remote queues	10-23
Authorization checking in the MQI	10-24
Authority events (1 of 2)	10-25
Authority events (2 of 2)	10-26
Channel authentication control	10-27
Channel authentication commands	10-28
Channel authentication example	10-29
IBM MQ security exits	10-30
Connection authentication	10-31
Enabling connection authentication on a queue manager	10-32
Enabling connection authentication examples	10-33
Secure Sockets Layer (SSL)	10-34
IBM MQ SSL support	10-35
Enabling SSL on the queue manager	10-36
Enabling SSL by using IBM MQ Explorer	10-37
Channel attributes for SSL	10-38
Default security configuration in IBM MQ Explorer	10-39
Storing IBM MQ Explorer passwords	10-40
Unit summary	10-41
Review questions	10-42
Checkpoint answers	10-43
Exercise: Controlling access to IBM MQ	10-44

Exercise objectives	10-45
Unit 11. Backing up and restoring IBM MQ messages and object definitions	11-1
Unit objectives	11-2
The need to back up	11-3
Backing up the IBM MQ file system	11-4
Overview of IBM MQ objects	11-5
Why back up IBM MQ object definitions	11-6
Capturing IBM MQ object definitions	11-7
Capturing resource definitions with the DISPLAY command	11-8
Save IBM MQ configuration: <code>dmpmqcfg</code>	11-9
Backing up queue manager configuration	11-10
Syntax examples of <code>dmpmqcfg</code>	11-11
Using <code>dmpmqcfg</code> : Output MQSC	11-12
Using <code>dmpmqcfg</code> : Output <code>setmqaut</code>	11-13
Overview of security definitions	11-14
Need to back up security definitions	11-15
Backing up security definitions	11-16
<code>dmpmqaut</code> command	11-17
Backing up security definitions in IBM MQ Explorer	11-18
Recovering persistent messages	11-19
Types of logs	11-20
Circular logs	11-21
Linear logs	11-22
Dumping the log	11-23
Damaged objects and media recovery	11-25
Recording media image to a log	11-26
Re-creating an object from an image	11-27
Valid object types for recording and re-creating images	11-28
Unit summary	11-29
Review questions	11-30
Review answers	11-31
Exercise: Using a media image to restore a queue	11-32
Exercise objectives	11-33
Exercise: Backing up and restoring IBM MQ object definitions	11-34
Exercise objectives	11-35
Unit 12. Introduction to queue manager clusters	12-1
Unit objectives	12-2
What is a cluster?	12-3
Benefits of clustering	12-4
Overview of cluster components	12-5
Distributed queuing versus clustering (1 of 3)	12-7
Distributed queuing versus clustering (2 of 3)	12-9
Distributed queuing versus clustering (3 of 3)	12-10
Steps to set up a basic cluster	12-12
Considerations for a full repository	12-13
Defining full repository queue managers	12-14
Cluster channels	12-16
Define cluster-receiver channels	12-17
Define cluster-sender channel	12-18
Display information about cluster queue manager	12-19
Display channel status	12-20
Define local queues to the cluster	12-21
Controlling clusters with MQSC commands	12-22
Cluster administration considerations	12-24

Defining a cluster by using IBM MQ Explorer	12-25
Controlling clusters with IBM MQ Explorer	12-26
Monitoring clusters with IBM MQ Explorer	12-27
Cluster workload management options	12-28
Use-queue basics	12-29
Application binding options	12-30
Using IBM Explorer to specify cluster queue properties	12-31
Verifying and testing the cluster	12-32
Cluster tests in IBM MQ Explorer	12-33
Implementation	12-34
Cluster troubleshooting	12-35
Administering the SYSTEM.CLUSTER queues	12-36
Unit summary	12-37
Review questions	12-38
Review answers	12-39
Exercise: Implementing a basic cluster	12-40
Exercise objectives	12-41
Unit 13. Monitoring and configuring IBM MQ for performance	13-1
Unit objectives	13-2
Collecting statistics and accounting data	13-3
Subscribing to statistics data (1 of 2)	13-4
Subscribing to statistics data (2 of 2)	13-5
Benefits of subscribing to statistics topics	13-6
IBM MQ statistics that are published to topics only	13-7
Queue manager statistics monitoring properties	13-9
MQI statistics	13-10
Queue statistics	13-11
Controlling queue statistics	13-12
Channel statistics	13-13
Controlling channel statistics	13-14
IBM MQ accounting data collection	13-15
Controlling queue manager accounting	13-17
Controlling queue accounting	13-18
Displaying statistics and accounting data	13-19
Examples of accounting output	13-20
Example of statistics data	13-21
Subscribing to statistics with the amqsrua program	13-22
Example: Using the amqsrua sample program	13-23
Real-time monitoring	13-24
Controlling queue manager real-time monitoring	13-25
Controlling channel and queue real-time monitoring	13-26
Displaying queue and channel monitoring data	13-27
Configuring and tuning IBM MQ for performance	13-28
Queue manager logs	13-29
Default log settings in IBM MQ Explorer	13-30
Log type	13-31
Log path	13-32
Log file pages	13-33
Log primary and secondary files	13-34
Log buffer pages	13-35
Log write integrity level	13-36
Queue manager Log settings in qm.ini	13-37
Queue manager channels performance tuning	13-38
Queue manager listeners performance tuning	13-39
Unit summary	13-40

Review questions	13-41
Checkpoint answers	13-42
Exercise: Monitoring IBM MQ for performance	13-43
Exercise objectives	13-44
Unit 14. Course summary	14-1
Unit objectives	14-2
Course objectives	14-3
Course objectives	14-4
To learn more on the subject	14-5
Enhance your learning with IBM resources	14-6
Unit summary	14-7
Appendix A. List of abbreviations	A-1

Trademarks

The reader should recognize that the following terms, which appear in the content of this training document, are official trademarks of IBM or other companies:

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide.

The following are trademarks of International Business Machines Corporation, registered in many jurisdictions worldwide:

AIX®	Bluemix®	developerWorks®
FFST™	First Failure Support Technology™	GPFS™
Notes®	PowerVM®	PureApplication®
WebSphere®	z/OS®	

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft and Windows are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java™ and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

UNIX is a registered trademark of The Open Group in the United States and other countries.

VMware and the VMware “boxes” logo and design, Virtual SMP and VMotion are registered trademarks or trademarks (the “Marks”) of VMware, Inc. in the United States and/or other jurisdictions.

SoftLayer® is a trademark or registered trademark of SoftLayer, Inc., an IBM Company.

Other product and service names might be trademarks of IBM or other companies.

Course description

IBM MQ V9 System Administration (using Windows for labs)

Duration: 4 days

Purpose

This course provides technical professionals with the skills that are needed to administer IBM MQ queue managers on distributed operating systems and in the Cloud. In addition to the instructor-led lectures, you participate in hands-on lab exercises that are designed to reinforce lecture content. The lab exercises use IBM MQ V9.0, giving you practical experience with tasks such as handling queue recovery, implementing security, and problem determination.

This course does not cover any of the features of MQ for z/OS or MQ for IBM i.

Audience

This course is designed for technical professionals who require the skills to administer IBM MQ queue managers on distributed operating systems, in the Cloud, or on the IBM MQ Appliance.

Prerequisites

- Basic knowledge of IBM MQ V9 concepts and features, obtained either through experience or by successfully completing *Technical Introduction to IBM MQ* (WM103G) or *Technical Introduction to IBM MQ* (ZM103G)
- Ability to invoke standard functions within the operating system that is used in the lab exercises
- Some knowledge of TCP/IP configuration

Objectives

- Describe the IBM MQ deployment options
- Plan for the implementation of IBM MQ on-premises or in the Cloud
- Use IBM MQ commands and the IBM MQ Explorer to create and manage queue managers, queues, and channels
- Use the IBM MQ sample programs and utilities to test the IBM MQ network
- Enable a queue manager to exchange messages with another queue manager
- Configure client connections to a queue manager
- Use a trigger message and a trigger monitor to start an application to process messages
- Implement basic queue manager restart and recovery procedures
- Use IBM MQ troubleshooting tools to identify the cause of a problem in the IBM MQ network

- Plan for and implement basic IBM MQ security features
- Use accounting and statistics messages to monitor the activities of an IBM MQ system
- Define and administer a simple queue manager cluster

Contents

- Administration by using commands and IBM MQ Explorer
- Implementing distributed queuing

Curriculum relationship

This course is a prerequisite for course WM213, *IBM MQ V9 Advanced System Administration*, and course WM253, *Designing, Implementing, and Managing IBM MQ V9 Clusters*.

Agenda



Note

The following unit and exercise durations are estimates, and might not reflect every class experience.

Day 1

- (00:15) Course introduction
- (00:30) Unit 1. IBM MQ review
- (00:30) Unit 2. IBM MQ installation and deployment options
- (01:00) Unit 3. Creating a queue manager and queues
- (00:30) Exercise 1. Using commands to create a queue manager and queues
- (00:30) Unit 4. Introduction to IBM MQ Explorer
- (00:30) Exercise 2. Using IBM MQ Explorer to create queue managers and queues
- (00:30) Unit 5. Testing the IBM MQ implementation
- (00:30) Exercise 3. Using IBM MQ sample programs to test the configuration

Day 2

- (01:30) Unit 6. Implementing distributed queuing
- (01:00) Exercise 4. Connecting queue managers
- (01:00) Unit 7. IBM MQ clients
- (01:00) Exercise 5. Connecting an IBM MQ client
- (00:30) Unit 8. Implementing trigger messages and monitors
- (01:00) Exercise 6. Implementing a trigger monitor

Day 3

- (01:00) Unit 9. Diagnosing problems
- (00:30) Exercise 7. Running an IBM MQ trace
- (01:00) Unit 10. Implementing basic security in IBM MQ
- (01:00) Exercise 8. Controlling access to IBM MQ
- (01:00) Unit 11. Backing up and restoring IBM MQ messages and object definitions
- (00:30) Exercise 9. Using a media image to restore a queue
- (00:30) Exercise 10. Backing up and restoring IBM MQ object definitions

Day 4

- (01:00) Unit 12. Introduction to queue manager clusters
- (01:00) Exercise 11. Implementing a basic cluster
- (01:30) Unit 13. Monitoring and configuring IBM MQ for performance
- (01:30) Exercise 12. Monitoring IBM MQ for performance
- (00:30) Unit 14. Course summary

Unit 1. IBM MQ review

Estimated time

00:30

Overview

This unit reviews IBM MQ basic concepts and components. It also reviews the runtime options for IBM MQ on-premises and in the Cloud.

How you will check your progress

- Review questions

References

IBM Knowledge Center for IBM MQ V9

How to check online for course material updates



Note: If your classroom does not have internet access, ask your instructor for more information.

Instructions

1. Enter this URL in your browser:
ibm.biz/CloudEduCourses
2. Find the product category for your course, and click the link to view all products and courses.
3. Find your course in the course list and then click the link.
4. The wiki page displays information for the course. If there is a course corrections document, this page is where it is found.
5. If you want to download an attachment, such as a course corrections document, click the **Attachments** tab at the bottom of the page.
6. To save the file to your computer, click the document link and follow the prompts.

[Comments \(0\)](#) [Versions \(1\)](#) **Attachments (1)** [About](#)

Figure 1-1. How to check online for course material updates

Unit objectives

- Summarize the features and benefits of IBM MQ
- Identify the IBM MQ components and their functions
- Describe the IBM MQ runtime options for on-premises and Cloud
- Use the IBM Knowledge Center for IBM MQ V9

IBM MQ review

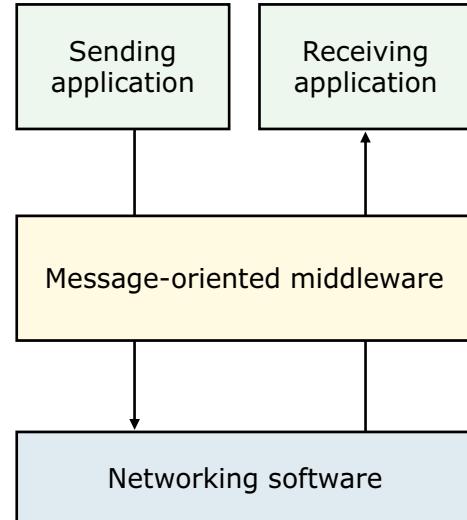
© Copyright IBM Corporation 2017

Figure 1-2. Unit objectives

Course WM103 or ZM103, *A Technical Introduction to IBM MQ V9*, is a prerequisite for this course; this unit is a review of previously acquired knowledge.

What is IBM MQ?

- Message-oriented middleware that supports sending and receiving data in messages between distributed systems
 - Messages are placed on queues in storage
 - Programs can run independently of each other, at different speeds and times, in different locations
 - Source and target applications do not require a logical connection
- Supports asynchronous calls between the client and server applications



IBM MQ review

© Copyright IBM Corporation 2017

Figure 1-3. What is IBM MQ?

A key impediment to a flexible enterprise is traditional application connectivity, which combines business applications with one of many network application programming interfaces (API). This architecture requires knowledge of the networking conditions to adequately handle any communication problems.

With messaging-oriented middleware, such as IBM MQ, all the work of connecting, polling, handling failure, and implementing change is removed from the application. Instead, message-oriented middleware provides a dedicated middleware layer for handling connectivity. It is decoupled from the sending and receiving application, which provides the flexibility to react to business conditions.

IBM MQ creates a distributed communications layer that insulates the application developer from the details of the various operating systems and network interfaces. So, application modules can be distributed over heterogeneous operating systems.

In IBM MQ, data is packaged in *messages*, which are stored on *queues*. Queues allow programs to put and get messages as required so that programs can run independently of each other, at different speeds and times, and in different locations.

IBM MQ supports asynchronous calls between the client and server applications.

IBM MQ features

- Transports any type of data as messages, which enable businesses to build flexible, reusable architectures such as service-oriented architecture (SOA) environments
- Runs on a broad range of operating systems and hardware
- Connects to applications, web services, and communications protocols for security-rich message delivery
- Provides versatile messaging integration, from mainframe to mobile, in a single robust messaging backbone
- Shields application developers from networking complexities, enabling them to develop and deploy new applications faster
- Includes administrative features that simplify messaging management and reduce time that is spent on using or developing complex tools
- Offers a range of qualities of service (QoS)
- Provides a common application programming interface that is consistent across all the supported operating systems

IBM MQ review

© Copyright IBM Corporation 2017

Figure 1-4. IBM MQ features

What is IBM MQ?

IBM MQ is enterprise messaging software that is used in the market for over 25 years.

IBM MQ provides asynchronous messaging for a wide range of systems. Applications that are indecipherable because they were written in an obsolete language can communicate with those applications that were written in the popular language last week.

IBM MQ communication is fast, and because the messaging is asynchronous, the business has the assurance that data is not lost. With IBM MQ, data arrives on time, in order, and once only.

IBM MQ includes administrative features and tools that simplify the management of the MQ components and the messages that it processes. It also offers a range of qualities of service (QoS).

The power of IBM MQ

- **Reliability**

Assured delivery of business-critical events when and where needed

- **Universal connectivity**

Freedom to use the right technology for the business

- **Flexibility**

Day-to-day business needs and tomorrow's growth opportunities

- **Scalability**

Grow applications and capacity incrementally, scale elastically

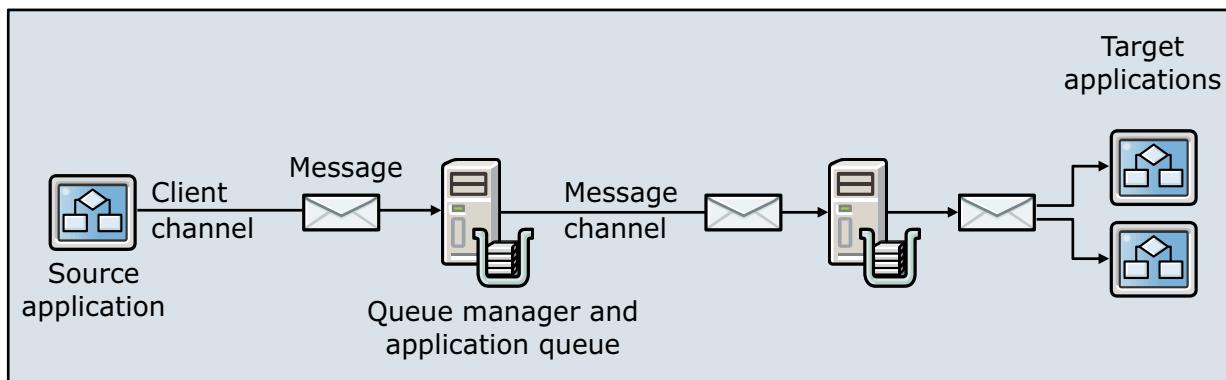
Effective messaging provides reliability, universal connectivity, flexibility, and scalability.

Effective messaging solves the following common business problems:

- If one application fails, many other applications fail. It costs the business a lot in downtime and recovery.
- An organization's applications run on different hardware and operating systems, and are written in different programming languages.
- A process was originally designed for one purpose, but now it must change to meet new requirements without breaking other applications.
- The business expands, and the capacity of the system can no longer cope with the workload demand.

IBM MQ components

- *Messages* that contain application data
- *Queues* that hold messages
- *Channels* that connect queue managers and client applications
- *Queue manager* that manages messages, queues, channels, and other IBM MQ services and resources



IBM MQ review

© Copyright IBM Corporation 2017

Figure 1-6. IBM MQ components

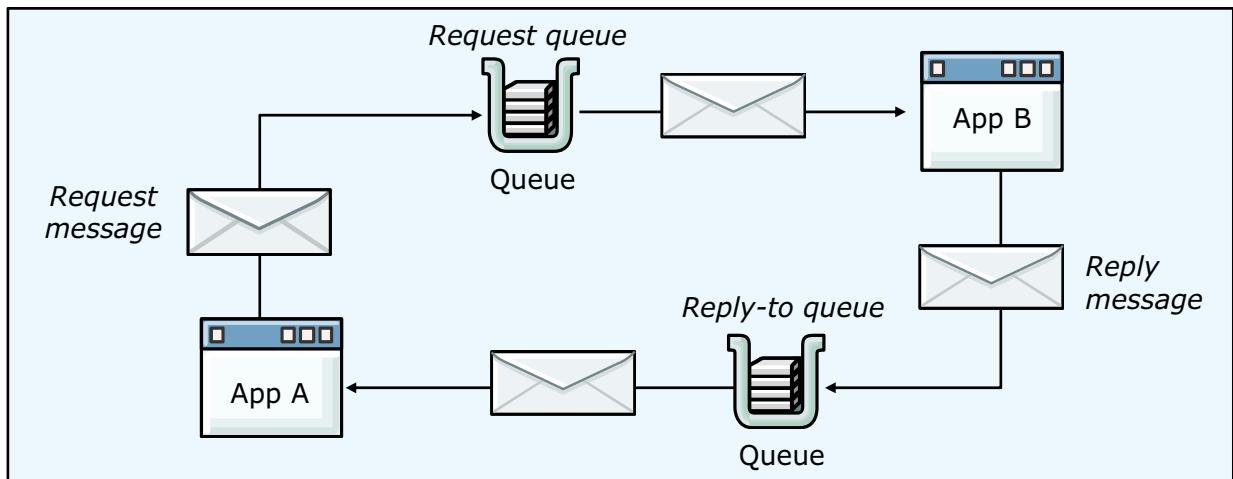
The main components of an IBM MQ configuration are queue managers, messages, queues, and channels. Each of these components is described in more detail in this unit.

An *MQ server* controls the computer that contains the IBM MQ configuration. Computers that contain client applications that must connect to the MQ server must also have an *MQ client* installation. It is possible to have both an MQ server and an MQ client installation on the same computer.

The difference between an MQ server and an MQ client is discussed at the end of this unit. Unless the term MQ client is specified, the content in this unit applies to MQ server installations.

IBM MQ functional overview

- IBM MQ uses messages and queues to provide program-to-program communication
- IBM MQ provides:
 - Assured message delivery
 - Time-independent processing
 - Application parallelism



IBM MQ review

© Copyright IBM Corporation 2017

Figure 1-7. IBM MQ functional overview

IBM MQ uses messages and queues to provide program-to-program communication. The communicating applications can be on the same system, or distributed across a network of IBM and non-IBM systems.

In general, multiple applications can put messages on the same queue to request the same service. The application that serves the queue gets each message and responds to it.

The example in the figure shows a simple request-reply scenario that uses messages and queues. In this example, a client application (App A) puts a message onto a request queue. Another application (App B) gets the message from the request queue and processes the request. The reply message is put on the reply-to queue. The original requester (App A) can then get the reply message from the reply-to queue.

In this example, the original request contains a message descriptor that provides the name of the reply-to queue for the reply message. Each requesting application can have its own reply-to queue so that each client application can receive its replies separately from other client applications.

The message descriptor also contains a message identifier that the requesting application can use to correlate a reply with a request.

IBM MQ can transfer data with assured delivery; messages do not get lost, even when a system failure occurs. MQ does not duplicate message delivery.

With IBM MQ, communicating applications do not have to be active at the same time. An application is divided into discrete functional modules that communicate with each other by using messages. In this way, the modules can run on different systems, be scheduled at different times, or they can act in parallel.

Messages



- Distinct string of bytes exchanged between applications
 - Contains MQ message descriptor (MQMD) with message control information
 - Contains the application data or payload
 - Can contain optional headers that contain information about message structure, intended consumers, and message properties
- Are placed on queues, which allow programs to run independently of each other, at different speeds and times, in different locations, and without a logical connection between them

IBM MQ message descriptor (MQMD)	Additional headers (optional)	Application data (Payload)
----------------------------------	-------------------------------	----------------------------

IBM MQ review

© Copyright IBM Corporation 2017

Figure 1-8. Messages

The application data that MQ exchanges is sent in the form of messages. A *message* is any information that one application uses to communicate to another. A message can convey a request for a service, or it can be a reply to a request. A message might also report on the progress of another message: to confirm its arrival or report on an error, for example. A message can also carry information for which no reply is expected.

Messages consist of the MQ message descriptor (MQMD) and the payload. Optionally the application can also provide message properties.

The MQMD contains information about the message. Objects are identified by the MQMD when addressed from a program. The MQMD contains information about the sending application, for example.

The message data portion of an MQ message contains the actual payload, such as a bank transaction or healthcare record.

The sending application supplies both the message descriptor and the application data when it puts a message on a queue. The application that puts the messages on the queue sets some of the fields in the message descriptor; the queue manager sets others on behalf of the application. Both the message descriptor and the application data are returned to the application that gets the message from the queue.

Message persistence

- Persistent messages
 - Queue manager keeps a failure-tolerant recovery log of all actions on messages
 - Recovered from logged data after a queue manager restart
- Non-persistent messages
 - Stored in system memory only
 - Discarded when a queue manager stops for any reason
- Application can specify message persistence in MQMD
 - Use persistent messages for critical business data that must be reliably maintained and not lost in a failure
 - Use non-persistent messages where loss of data is not crucial and in cases where performance is considered more important than data integrity

IBM MQ review

© Copyright IBM Corporation 2017

Figure 1-9. Message persistence

The MQMD contains a persistence attribute that controls whether a message survives restarts of the queue manager.

If the application identifies the message as persistent, the queue manager keeps a failure-tolerant recovery log of all actions on that message. If a queue manager restarts before the message is delivered to its target, the message is recovered from the logged data. If an application contains critical business data that must be reliably maintained, it should identify a message as persistent. Examples of persistent messages are stock trades or banking transactions.

If the application identifies the message as nonpersistent, the message is stored in system memory only. If a queue manager restarts before the message is delivered to its target, the message is discarded. An application can identify a message as nonpersistent when the loss of data is not crucial.

Persistence is a critical attribute that controls whether a message survives restarts of the queue manager. What if it carries time-sensitive information? What if it contains an important time-independent notification? Before you rush to make a message persistent, it is important that you understand the use for this data.

Queues

- Defined endpoint destination for messages
 - Local queue* is owned by queue manager to which the application is connected
 - Remote queue* is owned by a different queue manager from the one to which the application is connected
 - Alias queue* is a pointer to a local queue or a locally owned remote queue
 - Model queue* is a template to create a dynamic local queue



Only queues that are defined as local queues (QLOCAL) hold messages

Figure 1-10. Queues

An MQ queue is a named object on which applications can put messages, and from which applications can get messages. Messages are stored on a queue, so that if the putting application is expecting a reply to its message, it can do other work while it waits for the reply. Applications access a queue by using the Message Queue Interface (MQI).

MQ supports different types of queues for storing, identifying, and moving messages in the MQ network. This figure summarizes the types of queues that the MQ supports.

Messages are stored on local queues, which are owned by the queue manager to which the application connects.

Remote queues and alias queues are pointers to other queues but do not hold messages. Messages can be directed to remote or alias queues, but the final target is a local queue.

It is not possible to retrieve messages from a remote queue or an alias queue; messages are always in the target local queue that is identified in the definition.

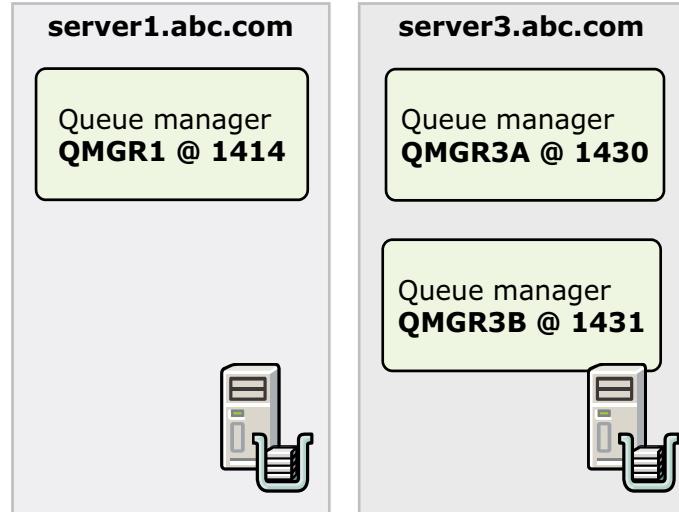
A model queue is a template for a dynamic local queue.

A queue is owned by a queue manager, and that queue manager can own many queues. However, each queue must have a name that is unique within that queue manager.

The graphics on this figure are used throughout this course to identify the different types of queues.

Queue manager

- Provides services for accepting and delivering messages
- Maintains queues of all messages that are waiting to be processed or routed
- System program that owns the resources that it services
- Requires a name and TCP/IP listener port
- A server can host more than one queue manager
 - Queue managers that share a server require different names and listener ports



IBM MQ review

© Copyright IBM Corporation 2017

Figure 1-11. Queue manager

Queue managers are the main components in an MQ messaging network. Queue managers host the other objects in the network, such as the queues and the channels that connect the queue managers together.

Queue managers define the properties of MQ objects. The values of these properties affect how MQ processes these objects. You create and manage objects by using MQ commands and interfaces.

A server can contain more than one queue manager but each queue manager on the server must have a unique name.

The queue manager communicates by using a TCP connection. In a distributed environment, each queue manager is assigned a unique TCP/IP port. The default TCP port for a queue manager is 1414. The port number 1414 is assigned to MQ by the Internet Assigned Numbers Authority. When the server contains more than one queue manager, each queue manager must have a different port number.

In the example, the server that is named server1.abc.com contains one queue manager that is named QMGR1, which is listening on TCP port 1414. The server that is named server3.abc.com contains two queue managers. The queue manager that is named QMGR3A is listening on TCP port 1430. The queue manager that is named QMGR3B is listening on TCP port 1431.

Queue manager terminology

- *Local queue manager* is the currently referenced queue manager
- All other queue managers are *remote queue managers*

Example:

- If working on QMGR3B, then QMGR3B is the local queue manager, and QMGR1 and QMGR3A are remote queue managers
- Objects and resources that are defined in QMGR3B are said to be “locally owned”

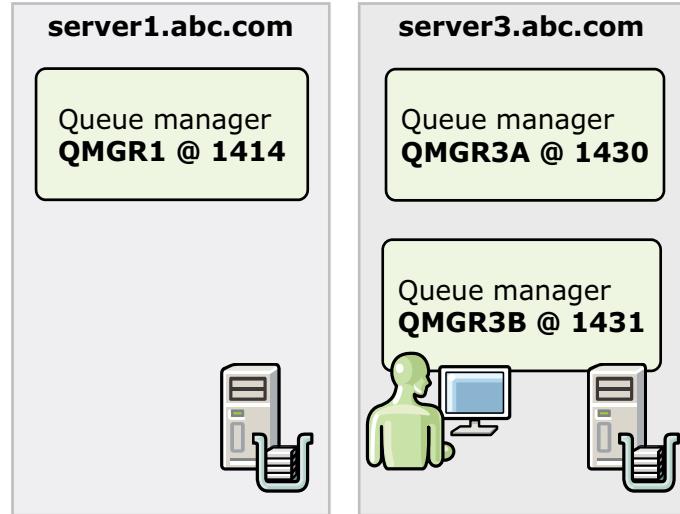


Figure 1-12. Queue manager terminology

To avoid confusion, you must understand what is meant by *remote* when you reference a queue manager in the network.

Whether a queue manager is local or remote depends on whether the administrator or application is directly connected to a queue manager. A local queue manager is the currently referenced queue manager; a remote queue manager is any queue manager that is not currently referenced.

The local queue manager designation changes when work moves to another queue manager. For example, if an administrator is connected to queue manager QMGR3B, then that queue manager is the local queue manager and the other queue managers (QMGR3A and QMGR1) are remote queue managers. If the administrator is connected to the queue manager that is named QMGR3A, the other queue managers (QMGR3B and QMGR1) are the remote queue managers.

Channels

- Configurable processes that send or receive messages to queue managers
- Message channel
 - One-way communications link between two queue managers
 - Defined in pairs
 - Message channel agent (MCA) controls sending and receiving of messages between queue managers
- Client (MQI) channel
 - Two-way communications link
 - Connects an application (MQI client) to a queue manager

IBM MQ review

© Copyright IBM Corporation 2017

Figure 1-13. Channels

Channels are the processes that move messages to and from queue managers.

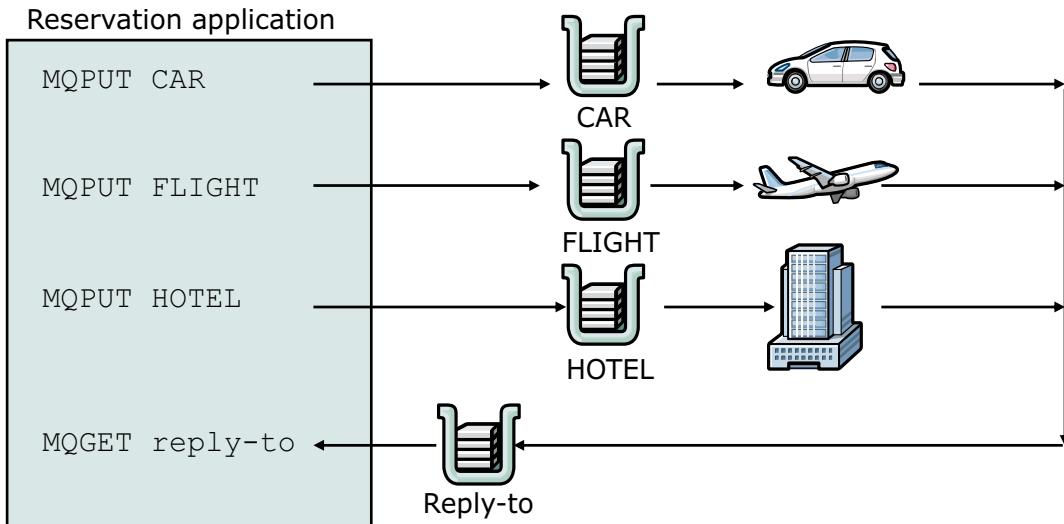
A *message channel* provides a one-way communications link between queue managers. Message channels are used in distributed queuing to move messages from one queue manager to another.

When you define message channels, you must define one channel object at the queue manager that is to send messages. You must also define another, complementary channel, at the queue manager that is to receive the messages. Sender-receiver channels are the most commonly used distributed message channels.

A message channel connects two queue managers by using message channel agents (MCAs). The MCA on one end takes messages from local transmission queue and puts them on communication link. The MCA on other end receives the messages and puts them on target queue.

A client channel, also referred to as an *MQI channel*, provides a two-way communications link between a client application and a queue manager.

Parallel processing



- Message descriptor provides name of reply-to queue
- Requests are not serialized
- Replies are consolidated
- Transactions have shorter elapsed time

IBM MQ review

© Copyright IBM Corporation 2017

Figure 1-14. Parallel processing

In general, multiple applications can put messages on the same queue to request the same service. The application that serves the queue gets each message and responds to it.

Parallel processing allows an application to send several requests without waiting for a reply to one request before sending the next. All the requests can then be processed in parallel.

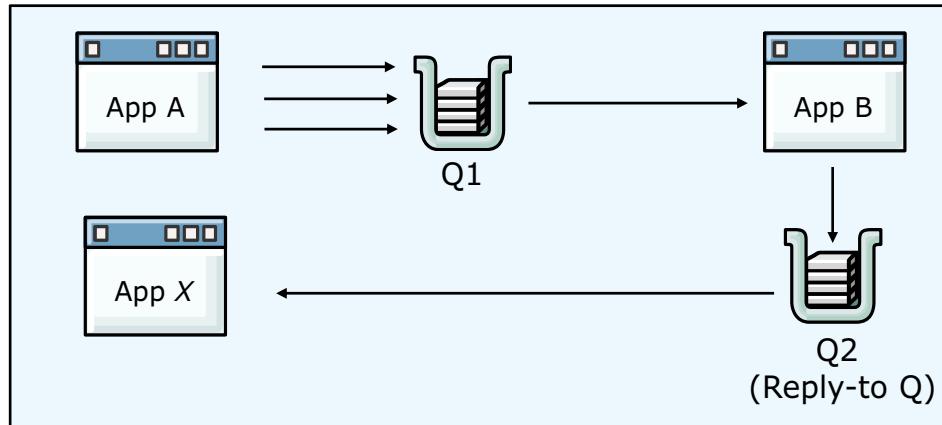
The application can process the replies when they are all received, and produce a consolidated answer. The program logic might also specify what to do when only a partial set of replies is received within a specified period.

A more complex application might involve sending a number of requests to different servers. In the example in the figure, a travel agent uses an application to arrange a trip for a customer. The application must make a number of requests, such as making a rental car, airplane, or hotel reservation. By using queues, these steps do not have to be completed serially.

In this example, the message descriptor provides the name of the reply-to queue. The reply-to queue informs the server application where to put the reply message. In this way, the client application can receive its replies separately from other client applications.

The application can process the replies when they are all received, and produce a consolidated answer. The program logic might also specify what to do when only a partial set of replies is received within a specified period.

Asynchronous processing



- Separate process for replies
- No need for communicating programs to be active at the same time
- Time independence

Figure 1-15. Asynchronous processing

In the asynchronous message processing model, instead of waiting for a reply to its first message, App A continues to send further requests to App B. In a separate process, App X receives the replies when they arrive on the reply-to queue (Q2).

In the asynchronous message processing model, App A is not dependent on App B to be running when the requests are sent. It can continue to do work even when App B is stopped.

In this model, it is expected that App X gets the reply messages but not necessarily at the same time that App B puts them on the reply-to queue. This scenario illustrates another of the major benefits of IBM MQ, which is *time independence*.

Application programming interfaces

- Message Queue Interface (MQI)
 - IBM MQ application programming interface
 - Supports many programming languages and styles, depending on the operating system and hardware, such as C, COBOL, and Visual Basic
- Support for the following object-oriented programming languages and frameworks:
 - .NET
 - ActiveX
 - C++
 - Java
 - JMS

IBM MQ review

© Copyright IBM Corporation 2017

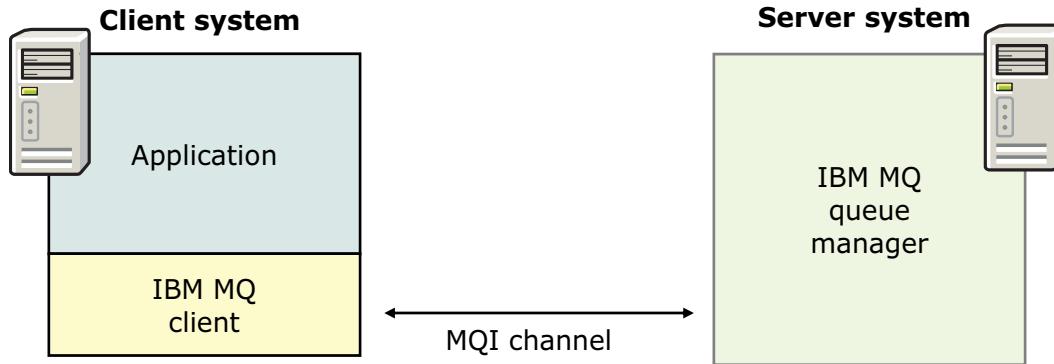
Figure 1-16. Application programming interfaces

You can develop applications to send and receive messages, and to manage your queue managers and related resources. MQ supports applications that are written in procedural languages, and object-oriented languages and frameworks.

MQ supports an application programming interface. On distributed operating systems, MQ supports C, Visual Basic (on Windows), and COBOL. These procedural languages use the MQI to access MQ services.

MQ also supports the following object-oriented programming languages and frameworks: .NET, ActiveX, C++, Java, and JMS. These languages and frameworks use the IBM MQ Object Model. The IBM MQ Object Model provides classes that provide the same functions as the MQI calls and structures, but that are a more natural way of programming in an object-oriented environment. Some of the languages and frameworks that use the IBM MQ Object Model provide functions that are not available to the procedural languages that use the MQI.

IBM MQ client



- IBM MQ component on a system without a queue manager
 - Communicates with queue manager by an MQI channel
 - Allows an application that runs on the client system to connect to a queue manager that is running on another system

IBM MQ review

© Copyright IBM Corporation 2017

Figure 1-17. IBM MQ client

An MQ client is a component of MQ that allows an application that is running on one system to send MQI calls to a queue manager that is running on another system.

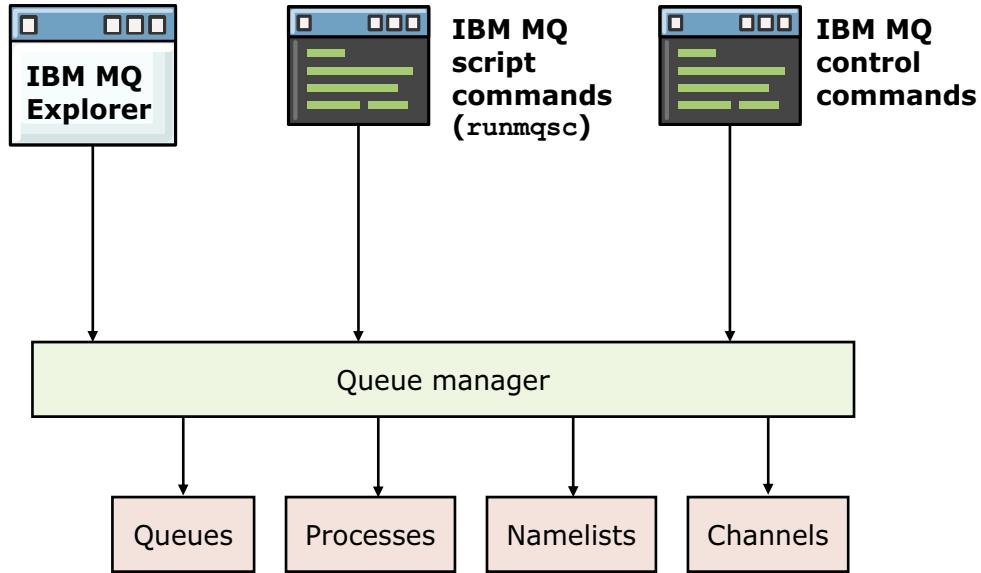
The *client connection* receives the input parameters of an MQI call from the application. It then sends the input parameters over a communications connection to the *server connection* on the same system as the queue manager. The server connection then sends the MQI call to the queue manager on behalf of the application. After the queue manager completes the MQI call, the server connection sends the output parameters of the callback to the client connection, which then passes them onto the application.

The combination of a client connection, a server connection, and a communications connection between them (for example, a TCP connection) is called an *MQI channel*.

The full range of MQI calls and options are available to a client application.

You learn more about MQ clients in Unit 7, “IBM MQ clients”.

IBM MQ administration interfaces



IBM MQ review

© Copyright IBM Corporation 2017

Figure 1-18. IBM MQ administration interfaces

IBM MQ provides three primary interfaces for managing MQ queue managers and queue manager resources such as queues, processes, namelists, and channels.

IBM MQ Explorer is a graphical user interface that runs on Linux and Windows. It can connect to, control, and configure MQ on all operating systems. MQ Explorer is described in detail later in this course.

MQ script commands (MQSC) and MQ control commands are command-line interfaces that used to create and modify queue managers and queue manager resources. MQSC and MQ control commands are described in more detail throughout this course.

Advantages of messaging with IBM MQ

- IBM MQ clients enable an application to connect remotely or locally to an IBM MQ queue manager
- Publish/subscribe support increases messaging capability from point-to-point messaging to a less coupled style of messaging
- Clusters of IBM MQ queue managers allow multiple instances of the same service to be hosted through multiple queue managers for load balancing, failover, and to simplify administration
- Secure Sockets Layer (SSL) and Transport Layer Security (TLS) can be used to secure communication between queue managers and IBM MQ clients
- IBM MQ supports a wide range of hardware and operating systems

IBM MQ review

© Copyright IBM Corporation 2017

Figure 1-19. Advantages of messaging with IBM MQ

This figure lists some of the IBM MQ features that make communication and configuration easier.

A separate MQ client component enables an application to connect remotely or locally to an MQ queue manager.

IBM MQ supports two primary communication methods: point-to-point and publish/subscribe. In publish/subscribe communication, a publishing application publishes a copy of each message and delivers it to every application that subscribes to the publication. 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. Publish/subscribe is described in more detail in course WM213, *IBM MQ V9 Advanced System Administration on Distributed*.

An MQ cluster is a collection of queue managers that can be on different servers and operating systems, and typically serve a common application. Every queue manager in the cluster can make the queues that they host available to every other queue manager in the cluster, without the need for remote queue definitions. Queue manager clusters are described in more detail in Unit 12, “Introduction to queue manager clusters”.

MQ supports SSL and TLS for security channels between queue managers and between queue managers and MQ clients. Security is described in more detail in Unit 10, “Implementing basic security in IBM MQ”.

MQ runs on wide variety of operating systems and hardware. MQ installation options are described in more detail in Unit 2, “IBM MQ installation and deployment options”.

Deployment options

- IBM MQ and IBM MQ Advanced
 - Implement a single messaging backbone that connects applications all parts of your business, from mainframe to mobile, and on premises to Cloud
 - Reduce development and administration time with simple tools
- IBM MQ Appliance
 - IBM MQ on a physical appliance, with premium hardware, including SSDs
 - Pre-optimized for simple setup
 - Easy maintenance, with firmware updates
 - Deploy to partner and remote locations
- Cloud options
 - Plug into a range of services on IBM's PaaS, IBM Bluemix with IBM Message Hub
 - Access repeatable solutions, with IBM MQ on PureApplication and SoftLayer and other Cloud and virtualized environments
 - Develop in an ever-expanding network of tools that work with containers

[IBM MQ review](#)

© Copyright IBM Corporation 2017

Figure 1-20. Deployment options

This figure summarizes the development options for IBM MQ.

IBM MQ is messaging middleware that simplifies and accelerates the integration of diverse applications and business data across multiple platforms. It uses message queues to facilitate the exchange of information between applications, systems, services, and files and simplifies the creation and maintenance of business applications.

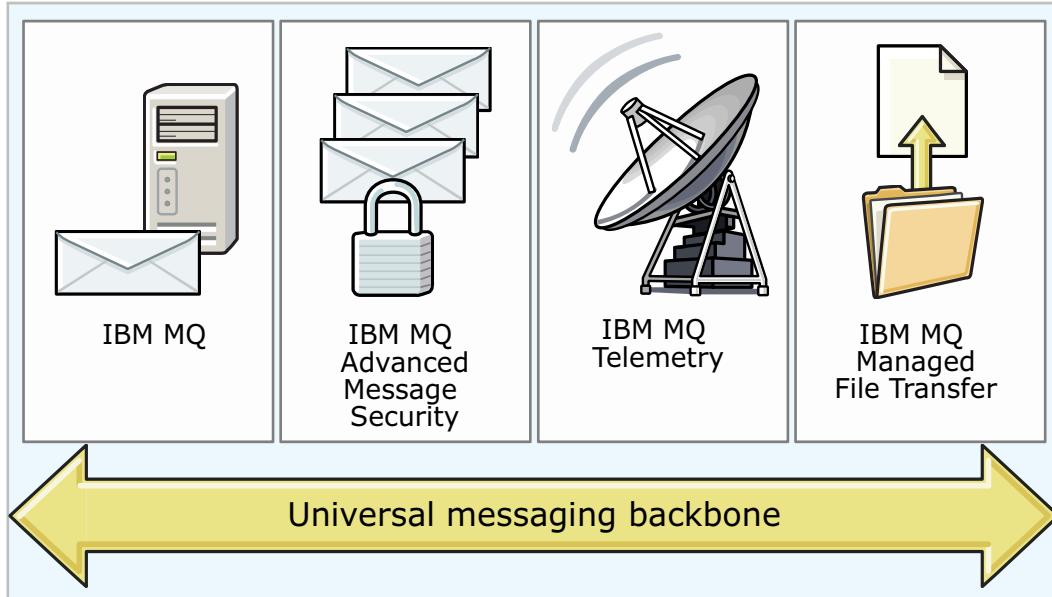
IBM MQ Advanced is IBM MQ plus managed file transfer, advanced message security, and telemetry. It provides lightweight messaging between the mainframes and mobile, and machine-to-machine connectivity, helping to reduce business costs and complexities, while it speeds time to value.

IBM MQ and IBM MQ Advanced can be installed on a server network in your enterprise.

The IBM MQ Appliance provides an optimized version of MQ that runs in a hardware appliance. It offers MQ capability in a hardware appliance and provides more features and some appliance benefits.

You can extend IBM MQ to the Cloud by using IBM's PaaS (platform as a service) such as IBM PureApplication System, IBM Bluemix, IBM SoftLayer, and other Cloud and virtualized environments.

IBM MQ Advanced



- Strengths of IBM MQ
- End-to-end encryption of data with IBM MQ Advanced Message Security
- Transfer of files as messages with IBM MQ Message File Transfer
- Real-time connection to sensors and mobile with IBM MQ Telemetry

IBM MQ review

© Copyright IBM Corporation 2017

Figure 1-21. IBM MQ Advanced

You can license the IBM MQ base software only. Optionally, you can license the IBM MQ Advanced package, which includes IBM MQ and three MQ extensions:

- IBM MQ Advanced Message Security provides advanced security features such as data encryption from the point it leaves the sending application to the point it enters the receiving application.
- IBM MQ Telemetry extends the universal messaging backbone with the MQTT protocol to a wide range of remote devices. It provides real-time access for enterprise applications to connect to a range of mobile devices, remote sensors, actuators, and other telemetry devices.
- IBM MQ Managed File Transfer facilitates the secure, reliable, and real-time transfer of files within the MQ network, and across a range of other platforms and networks. This manageable and auditable file transfer solution provides greater visibility and control of file transfer activity through a single system. IBM MQ Managed File Transfer is optimized to eliminate costly redundancies and lower maintenance costs, while it maximizes existing IT investments, to enable customers to move their business data stored in files over the IBM MQ infrastructure.

IBM MQ Advanced helps to meet your current and emerging connectivity needs reliably with a complete, integrated, and universal messaging solution.

IBM MQ product information

- IBM Knowledge Center for IBM MQ V9:
http://www.ibm.com/support/knowledgecenter/SSFKSJ_9.0.0/com.ibm.mq.helphome.v90.doc/WelcomePagev9r0.htm
 - Contains detailed instructions on how to complete the tasks to create and maintain the IBM MQ environment
- IBM DeveloperWorks: <https://www.ibm.com/developerworks/community/>
 - Contains links to forums, blogs, podcasts, and communities where you can discover and share information with other product users
- IBM MQ product site: <http://www.ibm.com/software/products/en/ibm-mq>
 - Latest news
 - References
 - Software downloads

IBM MQ review

© Copyright IBM Corporation 2017

Figure 1-22. IBM MQ product information

The IBM Knowledge Center for IBM MQ V9 contains documentation for the MQ components that you install. In addition to reference documentation, the IBM MQ product documentation also contains links to tutorials and product tours.

Other resources, such as IBM DeveloperWorks and the IBM MQ product site, provide information about how to use new features, tips and techniques, and connections to user communities.



IBM Knowledge Center for IBM MQ V9

IBM Knowledge Center

WebSphere MQ > WebSphere MQ 9.0.0 > Welcome

Getting started

- IBM MQ information roadmap
- What's new in Version 9.0
- FAQ for Long Term Support and Continuous Delivery releases
- Notices
- Accessibility features for IBM MQ
- Downloading IBM MQ Version 9.0 from the Passport Advantage Web site
- Scenario: Getting started with IBM MQ
- Planning
- Migrating and upgrading
- Installing IBM MQ
- Other learning resources
- Reference
- Mobile Messaging and M2M Client Pack Version 1.0.0
- IBM MQ Appliance Version 8.0
- IBM MQ Version 9.0 PDFs

Common tasks

- Availability, recovery and restart
- Planning security
- Creating and managing queue managers on distributed platforms
- Configuring distributed queuing
- Administering
- Developing applications
- Application development concepts
- Message monitoring
- Dealing with problems
- Diagnostic messages
- Troubleshooting and support
- Product support
- System requirements
- Support technotes
- IBM Support Portal
- IBM Support Assistant

More information

- IBM MQ Version 9.0 library page
- IBM MQ Version 9.0 downloadable documentation
- IBM MQ Program Directory PDFs
- IBM Redbooks domain for WebSphere software platform
- IBM MQ Migration Guide for distributed systems
- IBM MQ for z/OS® Migration Guide
- developerWorks: WebSphere MQ resources
- developerWorks: WebSphere application integration zone
- developerWorks: New to WebSphere
- Global WebSphere Community
- MQSeries®.net
- Follow the IBM Messaging Twitter community

IBM MQ review

© Copyright IBM Corporation 2017

Figure 1-23. IBM Knowledge Center for IBM MQ V9

The IBM Knowledge Center for IBM MQ v9 is the primary source of information about IBM MQ. It contains information to help you understand the product and describes implementation scenarios. The IBM Knowledge Center is available online and in download versions for Windows and Linux.

- Primary source of information about IBM MQ
- Contains information to help you understand the product, and the ways in which you can use it to solve your business problems
- Downloadable on Windows and Linux

Unit summary

- Summarize the features and benefits of IBM MQ
- Identify the IBM MQ components and their functions
- Describe the IBM MQ runtime options for on-premises and Cloud
- Use the IBM Knowledge Center for IBM MQ V9

IBM MQ review

© Copyright IBM Corporation 2017

Figure 1-24. Unit summary

Review questions

1. True or False: IBM MQ supports asynchronous messaging only.
2. IBM MQ assured delivery means that:
 - A. A report of delivery can always be sent back.
 - B. Unless the entire system goes down, no messages are lost.
 - C. Messages can be duplicated but never lost.
 - D. Messages are delivered with no loss or duplication.
3. Applications place messages on queues by using the IBM MQ:
 - A. Program-to-program interface
 - B. Message Queue Interface
 - C. Command processor



IBM MQ review

© Copyright IBM Corporation 2017

Figure 1-25. Review questions

Write your answers here:

- 1.
- 2.
- 3.

Review answers

1. True or False: IBM MQ supports asynchronous messaging only.

The answer is False. IBM MQ supports both synchronous and asynchronous messaging.



2. IBM MQ assured delivery means that:

- A. A report of delivery can always be sent back.
- B. Unless the entire system goes down, no messages are lost.
- C. Messages can be duplicated but never lost.
- D. Messages are delivered with no loss or duplication.

The answer is D.

3. Applications place messages on queues by using the IBM MQ:

- A. Program-to-program interface
- B. Message Queue Interface
- C. Command processor

The answer is B.

Unit 2. IBM MQ installation and deployment options

Estimated time

00:30

Overview

This unit describes the installation and deployment options for IBM MQ on-premises and in the Cloud. It also describes the release options for IBM MQ V9 and the process for locating and installing IBM MQ Fix Packs.

How you will check your progress

- Review questions

References

IBM Knowledge Center for IBM MQ V9

Unit objectives

- Summarize the IBM MQ installation options for on-premises and Cloud implementations
- Find the hardware and software prerequisites for an IBM MQ on-premises installation
- List the steps that are required to install IBM MQ on distributed operating systems
- Locate and install IBM MQ Fix Packs
- Locate IBM MQ SupportPacs

Deployment options

- IBM MQ and IBM MQ Advanced on-premises
- IBM MQ Appliance
- IBM MQ in the Cloud

IBM MQ installation and deployment options

© Copyright IBM Corporation 2017

Figure 2-2. Deployment options

You have many options for deploying your IBM MQ solutions.

IBM MQ can be deployed in a traditional server network, on the IBM MQ Appliance, and in the Cloud.

IBM MQ and IBM MQ Advanced can be installed on a server network in your enterprise.

Optionally, you can use the IBM MQ Appliance, which provides an optimized version of MQ that runs in a hardware appliance. Offering MQ capability in hardware appliances provides more features and some appliance benefits.

You can extend IBM MQ to the Cloud by using IBM's PaaS (Platform as a Service) such as IBM PureApplication System, IBM Bluemix, IBM SoftLayer, and other Cloud and virtualized environments.

IBM MQ Appliance M2001

- Provides scalability and security of IBM MQ in the convenience, fast time-to-value, and low total cost of ownership of an appliance
 - Integrates seamlessly into IBM MQ networks
 - Familiar administration model for administrators with IBM MQ skills
 - Supports IBM MQ Light API
- Ideal for use as a messaging hub that runs queue managers that clients access, or to extend IBM MQ connectivity to a remote location
- Familiar feel for existing IBM MQ users such as application interfaces, administration, networking, clustering, and security
- Appliance-specific features such as built-in high availability and disaster recovery
- Helps extend applications to the Cloud as part of a hybrid infrastructure



IBM MQ installation and deployment options

© Copyright IBM Corporation 2017

Figure 2-3. IBM MQ Appliance M2001

The IBM MQ Appliance provides the application connectivity performance of MQ software in a physical messaging appliance. It offers rapid deployment of enterprise messaging with easier administration. Performance and message throughput are optimized for the appliance's capability and configuration.

You can also use the MQ Appliance as the runtime messaging provider for applications that are written by using the IBM MQ Light API. It also supports connectivity from other programming interfaces such as the MQI and JMS.

IBM MQ in the Cloud

- Plug into a range of services on IBM's PaaS, IBM Bluemix with IBM Message Hub
- Access repeatable solutions, with IBM MQ on PureApplication and SoftLayer and other Cloud and virtualized environments
- Reliable, secure, and robust messaging solution for deployments in the Cloud, on-premises or in hybrid environments
- Easier to configure and manage in Cloud and hybrid environments, while also delivering enhanced encryption configurations, updates to managed file transfer capabilities, and more flexible delivery and support options
- Centrally managed client connectivity for continuously available applications
- Ability to bridge from core business applications to Cloud applications

IBM MQ installation and deployment options

© Copyright IBM Corporation 2017

Figure 2-4. IBM MQ in the Cloud

You can extend IBM MQ to the Cloud by using IBM PureApp, SoftLayer, and IBM Bluemix.

IBM MQ Hypervisor editions

- Contain the operating system and an IBM MQ installation
- Three different ways to deploy IBM MQ:
 - Run IBM MQ Hypervisor Edition for Red Hat Enterprise Linux Server with VMware ESX hypervisor
 - Deploy IBM MQ Hypervisor Editions with IBM Workload Deployer
 - Run IBM MQ Hypervisor Editions with IBM PureApplication System

IBM MQ installation and deployment options

© Copyright IBM Corporation 2017

Figure 2-5. IBM MQ Hypervisor editions

Hypervisors are software or firmware components that can virtualize system resources. Examples of hypervisors are PowerVM for AIX and VMware ESXServer.

The MQ Hypervisor Editions are self-contained virtual machine images. The images contain the operating system and MQ. You can deploy the virtual machine images into a Cloud with IBM Workload Deployer or IBM PureApplication System.

The other resources include an MQ basic part, script packages, and a Python script. The Python script loads the MQ virtual image and script packages onto an appliance, and creates a default MQ virtual system pattern.

With the cluster script packages, you can configure a pattern to add or remove a cluster of queue managers. The other script package runs the MQSC command tool.

Supported Hypervisors (1 of 2)

- Windows
 - Microsoft Hyper-V Server 2008 R2, 2012, and 2012 R2
 - VMware ESXi 5.5, 6.0, and 6.1
- Linux
 - IBM PR/SM any version
 - IBM PowerVM Hypervisor (LPAR, DPAR, Micro-Partition) any supported version
 - KVM for IBM z Systems 1.1
 - KVM in SUSE Linux Enterprise Server 12
 - Microsoft Hyper-V Server 2008 R2, 2012, and 2012 R2
 - Red Hat KVM as delivered with Red Hat Enterprise Linux and its RHEV equivalent 7.0
 - VMware ESXi 5.5, 6.0, and 6.1
 - z/VM 6.1, 6.2, and 6.3

See the IBM MQ product website for the most current list and any limitations

This figure summarizes the hypervisors that support IBM MQ V9 on Windows and Linux.

See the IBM MQ product website for the most current list of hypervisors for IBM MQ V9.

Supported Hypervisors (2 of 2)

- AIX
 - IBM PowerVM Hypervisor (LPAR, DPAR, Micro-Partition) any supported version
 - Live Application Mobility (LAM) for Workload Partition (WPAR) AIX 7.1
 - WPAR: Product is installed in Global AIX Instance, which is run in System Workload Partition AIX 7.1
 - WPAR: Product is installed in System Workload Partition AIX 7.1
- IBM i
 - IBM PowerVM Hypervisor (LPAR, DPAR, Micro-Partition) any supported version
- z/OS
 - z/VM 6.1, 6.2, and 6.3

See the IBM MQ product website for the most current list and any limitations

This figure lists hypervisors that are available for IBM MQ on AIX, IBM i, and z/OS.

IBM MQ Advanced for PureApplication

- Uses prebuilt patterns and tools to create a virtual system pattern for an IBM MQ environment
- Install IBM MQ through a graphical interface, with preconfigured defaults
- Supports IBM MQ multi-instance queue managers by using the underlying PureApplication GPFS system, which is coupled with simple drag configuration
- Can use PureApplication console to manage pattern for common tasks such as applying maintenance, reviewing performance metrics, and viewing logs

IBM MQ installation and deployment options

© Copyright IBM Corporation 2017

Figure 2-8. IBM MQ Advanced for PureApplication

IBM MQ Advanced for PureApplication uses prebuilt patterns and tools to create a virtual system pattern for an MQ environment.

With IBM MQ Advanced for PureApplication, you install MQ through a graphical interface with preconfigured defaults; it is not necessary to use any commands to install MQ.

The MQ Advanced PureApplication pattern supports MQ multi-instance queue managers by using the underlying PureApplication GPFS system that is coupled with simple configuration.

The PureApplication console can manage the MQ Advanced PureApplication pattern for common tasks such as applying maintenance, reviewing performance metrics, and viewing logs.

IBM Message Hub on IBM Bluemix

- Hub for asynchronously connecting services inside Bluemix or beyond
 - Scalable, distributed, high throughput message bus based on Apache Kafka
 - Connects Cloud services asynchronously
- Can take advantage of hybrid environments by integrating with IBM MQ
- Enables microservices
 - Applications are broken into smaller parts
 - Changes to individual parts can be quickly made
 - Because they are independent, one change does not always affect the other parts
- Provides developers with the choice of APIs
 - IBM MQ Light
 - REST
 - Kafka
- Supports batch and real-time analytics

IBM MQ installation and deployment options

© Copyright IBM Corporation 2017

Figure 2-9. IBM Message Hub on IBM Bluemix

With the support of the MQ Light API, it is possible to connect MQ to IBM Bluemix so that you can take advantage of the benefits of working in the Cloud.

MQ connects to IBM Message Hub, which is a fast, scalable, durable service that is based on Apache Kafka. Message Hub sits between services in Bluemix and provides buffering, load balancing, and error handling. It is designed to work easily with a range of services that enable further simplification.

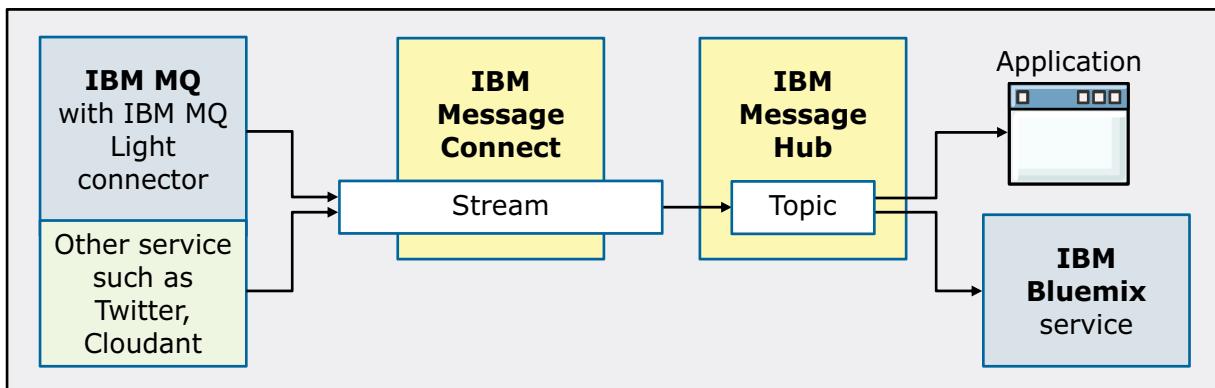
With the advent of Cloud, you have access to more endpoints than ever before. This movement towards Cloud usage prompted more interest in analysis of data. Message Hub can use data from one or many endpoints, which include MQ, and feed to one or more analytics engines, allowing different types of analysis to be run on the same data.

Message Hub enables development in a microservices framework. Microservices accelerate integration to help you use applications and data to respond quickly to business demands and create innovative solutions. For example, with Message Hub you can stream batch and real-time data to analytics applications to gain greater insight and advantage from your information.

Message Hub provides messaging for services inside and outside of Bluemix, and can also be used to take advantage of hybrid environments by integrating with MQ.

IBM Message Connect on IBM Bluemix

- Connects enterprise IBM MQ message network to the Cloud
 - Scalable, distributed, high throughput message bus
 - Loads data into analytics services to optimize business decisions
 - Builds on IBM MQ's dominance in enterprise messaging
- Combination of Message Hub and Message Connect provide features similar to other publish/subscribe systems
 - At one end, one or more publishers write data on to a stream
 - At the other end, one or more consumers receive a push notification of the event and the data packets from the publisher



IBM MQ installation and deployment options

© Copyright IBM Corporation 2017

Figure 2-10. IBM Message Connect on IBM Bluemix

IBM Message Connect connects streams of events from various sources and feeds them to your applications and services in Bluemix by using the IBM Message Hub publish/subscribe messaging service.

Events travel along streams, which act as pathways for data from event sources into Message Hub.

Connectors are prebuilt to connect to data sources and flow-specific data points onto a stream. For example, a connector can provide connectivity to Twitter's streaming search API. Tweets are channeled from Twitter to your stream without multiple connections to Twitter itself. As a Message Connect user, all you need to do is supply your Twitter authentication credentials and search criteria. Message Connect takes care of the rest, and you do not have to write a single line of code.

You can configure Bluemix applications to provide processing for the data, or to act based on incoming messages.

Other Cloud options for IBM MQ

- IBM MQ Advanced image for Docker
 - Run an IBM MQ queue manager inside a Docker container that Linux kernel manages
 - Provides process isolation, resource isolation, and dependency isolation
- Chef cookbook for IBM MQ
 - Source available on GitHub
 - Installs IBM MQ server and client
 - Creates and starts a queue manager
 - Calls MQSC to define a queue
- IBM MQ on OpenStack
 - Create custom virtual images

IBM MQ installation and deployment options

© Copyright IBM Corporation 2017

Figure 2-11. Other Cloud options for IBM MQ

You can also choose to run MQ in Docker or OpenStack.

With the IBM MQ Advanced image for Docker, you can run an MQ queue manager inside a Docker container, which can be useful for several reasons:

- All the processes that are associated with MQ are run in their own process space.
- You can limit the amount of memory and CPU that you allocate to a container.
- All the software on which MQ depends is included in the MQ image.

OpenStack is an infrastructure as a service (IaaS) offering, which provides a platform and unified API for managing infrastructure resources. These infrastructure resources include virtual machines, networks, and storage. MQ currently declares support for many of the individual OpenStack provider technologies, such as the KVM, Hyper-V, and VMware hypervisors.

MQ is also available as an image on Amazon Web Services and on Microsoft Azure.

GitHub contains Chef Cookbook for a basic IBM MQ installation.

IBM MQ on-premises installation

1. Decide on the installation name and path
 - New installation with no previous IBM MQ installation on the computer
 - Upgrade from previous version
2. Get base code and any maintenance level updates (Fix Packs)
3. Evaluate IBM MQ SupportPacs
4. Choose what to install
5. Prepare the system
6. Have a backup and fallback plan
7. Consider application migration issues and develop a plan and strategy
8. Install IBM MQ

IBM MQ installation and deployment options

© Copyright IBM Corporation 2017

Figure 2-12. IBM MQ on-premises installation

This figure summarizes the tasks for an IBM MQ on-premises installation. Each of these steps is described in more detail in this unit.

Multiple version installations

- On UNIX, Linux, and Windows, it is possible to have more than one copy of IBM MQ on a system
 - Each installation has a unique *installation name* that associates queue managers and configuration files with an installation
 - On Windows, you can choose the installation name during the installation process
 - On UNIX and Linux, the first IBM MQ installation is named Installation1; use the `crtmqinst` command for subsequent installation names before installing the product
- Identify *primary installation* to define which IBM MQ installation is used in system-wide locations
 - During installation
 - After installation with a command
 - Governs the level of function available when the queue manager is running
 - Ownership can be changed to a newer installation for migration

Note: You can install IBM MQ V9.0 on the same system with WebSphere MQ V7.1 and higher

Figure 2-13. Multiple version installations

You can install more than one copy of IBM MQ on distributed operating systems. An installation is the collection of binary libraries that make up MQ.

Multiple version installation support simplifies migration strategies because you can continue to use one version of MQ and gradually upgrade applications to a new version, without needing parallel hardware. In addition, vendor-acquired applications can embed MQ in a private directory, and can choose which versions of MQ are supported, without worrying about the “visible” version of MQ that a user might be using.

The installation details vary by operating system, but conceptually you first define an installation, and give it a name and description. You then run the installation program, which copies the code from media onto the disk in your chosen directories.

Generally, no “default” paths are created to an installation. However, a primary installation does insert itself into default locations.

You can have a single primary installation only on a system. On Windows, one installation is always identified as the primary installation because some operating system functions, such as how Windows .dll files are registered, require a single location.

An installation owns each queue manager, which governs the level of function available when a queue manager runs. When you install a newer level of code, the queue manager can be moved to that newer installation, and new functions can be used.

For easier migration, you can have an existing copy of WebSphere MQ V7.0.1.6 (or higher) on your systems. It is not necessary to upgrade before you use the multiple installation capabilities. If you already have WebSphere MQ V7.0.1 installed on your system, it is always the primary installation on all operating systems.

When you install IBM MQ, you can choose the directory into which it is installed.

Multiple version installation commands

- Set subsequent installation name before installation by using **crtmqinst** command:

Example: `crtmqinst -n MQInstall12 -p /opt/MQInstall -d "Second MQ installation"`

- On UNIX and Linux, installation configuration file `mqinst.ini` in the `/etc/opt/mqm` directory contains information about all IBM MQ installations
- Show installation details such as which queue managers exist and owning installations
 - **dspmq** and **dspmqinst**: Display installation information
 - **dspmqver**: Display installation version information
- Change association of queue manager to another installation by using **setmqm** command

Example: `MQ_INSTALL_PATH/bin/setmqm -m QMGR1 -n MQInstallV8`

Figure 2-14. Multiple version installation commands

IBM MQ commands can be used to examine and verify the installation information. In addition, installation details are maintained in the `/etc/opt/mqm/mqinst.ini` file on UNIX and Linux, and in the registry on Windows.

The commands are not in the default PATH for UNIX and Linux. If you are not working with the primary installation, all the control commands must include an explicit path. You can also use the **setmqenv** command to automatically set up the environment for use with a different installation of MQ. The **setmqenv** command and multiple version installations are described in detail in the IBM MQ product documentation.

Multiple version installation commands examples

```
$ /usr/mqm/bin/dspmqver -i
Name: IBM MQ
Version: 9.0.0.0
Level: p000-L110915
BuildType: IKAP - (Production)
Platform: IBM MQ for AIX
Mode: 64-bit
O/S: AIX 6.1
InstName: Installation1
InstPath: /usr/mqm
InstDesc: Default installation
DataPath: /var/mqm
Primary: Yes
MaxCmdLevel: 710

Name: IBM MQ
Version: 8.0.0.0
InstName: Installation2
InstPath: /usr/mqm2/usr/mqm
InstDesc: Second installation
Primary: No
```

```
$ dspmq -o installation
QMNAME (V80A)
    INSTNAME (Installation1)
    INSTPATH (/usr/mqm)
    INSTVER (9.0.0.0)
QMNAME (V80B)
    INSTNAME (Installation1)
    INSTPATH (/usr/mqm)
    INSTVER (9.0.0.0)
QMNAME (INST2QM)
    INSTNAME (Installation2)
    INSTPATH (/usr/mqm2/usr/mqm)
    INSTVER (8.0.0.0)
```

```
$ /usr/mqm/bin/endmqm INST2QM
AMQ5691: Queue manager 'INST2QM' is
associated with a different
installation.
```

Figure 2-15. Multiple version installation commands examples

This figure shows some examples of the commands that assist with installation management.

The `dspmq` and `dspmqver` commands show all of the installations, their directories, and the associated queue managers.

The queue manager control commands must be run from the correct directory. If you try to run them against a queue manager that is not associated with that installation, they return an error.

Optionally, you can also use the `setmqenv` command to automatically set up the environment for a different installation of MQ.

Multiple instance management is described in detail in the WM213/ZM213, *IBM MQ V9 Advanced System Administration*.

Choosing an installation name

- Installation name can be up to 16 bytes and must be a combination of alphabetic and numeric characters in the ranges a–z, A–Z, and 0–9
- Installation name cannot be changed after the product is installed
- Use the `dspmqinst` command to find the installation name that is assigned to an installation in a particular location

IBM MQ installation and deployment options

© Copyright IBM Corporation 2017

Figure 2-16. Choosing an installation name

Each installation of IBM MQ on UNIX, Linux, and Windows has a unique identifier that is known as an *installation name*. The installation name associates objects such as queue managers and configuration files with an installation.

You can choose an installation name that is meaningful to you. For example, you might call a test system **testMQ**.

An installation name can be up to 16 bytes and must be a combination of alphabetic and numeric characters in the ranges a – z, A – Z, and 0 – 9. You cannot use blank characters. The installation name must be unique, regardless of whether uppercase or lowercase characters are used. For example, the names **INSTALLATIONNAME** and **InstallationName** are not unique.

If you do not specify an installation name when MQ is installed, a default installation name is automatically assigned. For the first installation, this name is **Installation1**. For the second installation, the name is **Installation2**. The installation name **Installation0** is reserved for an installation of WebSphere MQ Version 7.0.1.

On UNIX and Linux, the first IBM MQ installation is automatically given an installation name of **Installation1**. For subsequent installations, you can use the `crtmqinst` command to set the installation name before you install MQ.

On Windows, you can choose the installation name during the installation process.

The installation name cannot be changed after MQ is installed.

You can find out what installation name is assigned to an installation in a particular location by using the **dsprqinst** command.

Choosing an installation path

- You can install IBM MQ to a custom location on disk during the installation process
 - Path that is specified must either be an empty directory, the root of an unused file system, or a path that does not exist
 - Path length is limited to 256 bytes
- Default installation path for IBM MQ on Windows
 - Software: C:\Program Files\IBM\MQ
 - Data files and logs: C:\ProgramData\IBM\MQ
- Default installation path for IBM MQ on Linux, HP-UX, and Solaris
 - Software: /opt/mqm
 - Data files and logs: /var/mqm

See the IBM Knowledge Center for IBM MQ V9 installation path restrictions

You can install MQ to a default or customer directory during the installation process. If you specify a custom path, it must be either an empty directory, the root of an unused file system, or a path to a directory that does not exist.

IBM MQ release and support versions

- Long-Term Support Release
 - Intended for systems that require long-term deployment and maximum stability
 - Software fixes and security updates only over specified time period
- Continuous Delivery Release
 - Intended for systems where applications can take advantage of the most recent capabilities of IBM MQ
 - New functions plus software fixes
 - Fixes only on most recent release
- For more information, see “IBM MQ FAQ for Long-Term Support and Continuous Delivery releases”:
<http://www.ibm.com/support/docview.wss?uid=swg27047919>

Figure 2-18. IBM MQ release and support versions

The next step in an on-premises installation is to obtain the base code and fix packs. To accelerate the speed of application development, IBM MQ Version 9.0 introduces a new continuous delivery and support mode for MQ base code and fix packs.

This new delivery model allows developers to access the most recent product enhancements and administrators to update their MQ environments with software fixes only.

This new delivery model also allows IBM to respond faster to enhancement requests and opportunities for expanding MQ availability on new infrastructures and environments.

The two MQ release versions for IBM MQ Version 9.0 are the Long Term Support Release (LTSR) and Continuous Delivery Release (CDR).

The Long-Term Support Release version is the product level for which support, including defect and security updates, is provided over a specified time. This version is intended for systems that require maximum stability.

The Continuous Delivery Release version delivers new functional enhancements, and fixes and security updates, on a much shorter release cycle. This version provides rapid access to new functions. It is intended for systems where enterprises want to use the newest capabilities of MQ.

For more information about the MQ release versions, see the “IBM MQ FAQ for Long-Term Support and Continuous releases” web page at:

<http://www.ibm.com/support/docview.wss?uid=swg27047919>

Supported operating systems and hardware (1 of 2)

- Windows on x/86-64
 - 10 Enterprise
 - 8.1 Enterprise, Professional, and Standard
 - 7 Enterprise, Professional, and Ultimate
 - 8 Enterprise, Professional, and Standard
 - Server 2008 R2 Datacenter, Enterprise, and Standards Editions
 - Server 2012 R2 Datacenter and Standard Editions
- Linux
 - Red Hat Enterprise Linux Server 7.2 (minimum) on IBM z Systems, Power System – Little Endian, and x86-64
 - SUSE Linux Enterprise Server 12 SP1 (minimum) on IBM z Systems, Power System – Little Endian, and x86-64
 - Ubuntu 14.04 LTS Power System – Little Endian, and x86-64

See the IBM MQ product website for the most current information about supported operating system, hardware, and any installation limitations

[IBM MQ installation and deployment options](#)

© Copyright IBM Corporation 2017

Figure 2-19. Supported operating systems and hardware (1 of 2)

This figure lists the supported operating systems for IBM MQ server and client installation on Windows and Linux.

For details of the specific operating system and hardware requirements, see the IBM MQ product documentation.

Supported operating systems and hardware (2 of 2)

- AIX V6.1 and V7.1 on POWER System – Big Endian
- Solaris 10 and 11 on SPARC and x86-64
- HP-UX Itanium 11i v3 on IA64
- IBM z/OS 2.1 and 2.2 on IBM z Systems
- IBM i 7.2 and 7.3 on POWER System – Big Endian

See the IBM MQ product website for the most current information about supported operating system, hardware, and any installation limitations

IBM MQ installation and deployment options

© Copyright IBM Corporation 2017

Figure 2-20. Supported operating systems and hardware (2 of 2)

This figure lists the supported operating systems for IBM MQ server and client installation on AIX, Solaris, HP-UX, IBM z/OS, and IBM i.

Always check the IBM MQ product documentation and the IBM MQ product website for the most current and detailed information about supported operating systems.

Locating IBM MQ Fix Packs

- IBM provides information about planned maintenance availability dates for the IBM MQ family of products:
www.ibm.com/support/docview.wss?uid=swg27006309#9
 - Dates are guidelines as to when future maintenance is scheduled to help you plan maintenance upgrades
 - Published dates are subject to change
- “Recommended Fixes for IBM MQ” web page provides links to newest available maintenance for IBM MQ:
www.ibm.com/support/docview.wss?uid=swg27006037
- Installation varies by operating system
 - See the IBM Knowledge Center for IBM MQ V9 for detailed instructions

IBM MQ installation and deployment options

© Copyright IBM Corporation 2017

Figure 2-21. Locating IBM MQ Fix Packs

IBM provides software updates to MQ in the form of fix packs. Before you install MQ, determine whether any fix packs are available for the version of MQ that you planning to install.

This figure identifies websites that contain information about IBM MQ Fix Packs.

IBM MQ SupportPacs

Downloadable code and documentation that complements the IBM MQ family

- Fee-based services SupportPacs
 - Provide material to be used by IBM Systems Specialists as the basis for fee-earning services such as application migration, and systems management
- As-is SupportPacs
 - Available at no charge to all licensed users of IBM MQ
 - Provided “as-is”
- Product Extensions SupportPacs
 - Available at no charge to all licensed users of the IBM MQ
 - Provide defect correction entitlement for licensed users who are entitled to service
- Third-Party Contributions SupportPacs
 - Provided by third-party suppliers such as licensed users, IBM Business Partners, and independent software vendors
 - Any specific licensing terms and conditions that apply to the use of such material are included with the relevant SupportPac

Figure 2-22. IBM MQ SupportPacs

The third step in an MQ on-premises installation is to evaluate the MQ SupportPacs.

MQ SupportPacs provide you with a wide range of downloadable code and documentation that complements the MQ family of products.

SupportPacs are available in four categories:

- Fee-based services (not available for download).
- Freeware that is provided “as is” with no warranty or defect correction.
- Fully supported product extensions that IBM develops. SupportPacs in this category are available at no charge to all licensed users of MQ products on the operating systems that are specified in the SupportPac description. They are supplied under the standard terms and conditions of the IBM International Program License Agreement (IPLA) and of the associated license information. They provide a defect correction entitlement for those licensed users who are entitled to service for IBM MQ. IBM reserves the right to discontinue service on the SupportPac when it is withdrawn from marketing by IBM.
- Vendor contributions that are provided “as is” with no warranty or defect correction. SupportPacs in this category are provided by third-party suppliers such as licensed users, and IBM Business Partners. While these SupportPacs are downloadable from the IBM website, any

specific licensing terms and conditions that apply to the use of the material are included with the relevant SupportPac.

IBM MQ SupportPac examples

- **MS07: IBM MQ Explorer** provides IBM MQ Explorer as a stand-alone application
- **MQC9: IBM MQ V9.0 Clients** contains all the IBM MQ V9 client components
- **MS81: WebSphere MQ Internet Pass-thru** provides a product extension that can be used to implement messaging solutions between remote sites across the internet

IBM MQ installation and deployment options

© Copyright IBM Corporation 2017

Figure 2-23. IBM MQ SupportPac examples

This figure lists some examples of IBM MQ SupportPacs.

Choosing what to install

- Select components and features that you require when you install IBM MQ:
 - IBM MQ server
 - IBM MQ client
 - IBM MQ Explorer
 - IBM MQ Managed File Transfer components
 - IBM MQ Telemetry server and basic or advanced clients
 - IBM MQ Advanced Message Security
 - Support files for Java, .NET messaging, and web services
 - Development Toolkit
- Available options vary by operating system

IBM MQ installation and deployment options

© Copyright IBM Corporation 2017

Figure 2-24. Choosing what to install

The next step in an MQ on-premises installation is to choose which MQ components you are going to install. Is this MQ installation for a developer or for production? Does this installation require IBM MQ Managed File Transfer or an IBM MQ Telemetry server?

As noted on the figure, the available options and components vary by operating system.

MQ has a typical, a compact, and a custom installation option. To ensure that the installation completes, select **Custom** so that you can select all the features. If you choose a **Typical** installation, you can easily miss new features.

Preparing the system

- Might be necessary to complete several tasks before you install IBM MQ depending on the operating system
- On UNIX, create user ID of `mqm` with a primary group for this user of `mqm`
- On UNIX and Linux:
 - Create the file systems for product code, working data, and error logs
 - After installation, run `mqconfig` script to compare the system kernel settings against the IBM default limits
- On Windows:
 - Ensure that the server name does not contain any spaces
 - For IBM MQ authorizations, ensure that user ID and group names do not exceed 20 characters (spaces are not allowed)
 - Default IBM MQ administration group of `mqm` and user ID of `MUSR_MQADMIN` are automatically created during the installation process

Figure 2-25. Preparing the system

The next step in an MQ on-premises installation is to prepare the system.

On UNIX, you must create a user ID and group that is named `mqm`. In most cases, the Linux installation program automatically creates this user and group.

On UNIX and Linux, run the `mqconfig` script to compare the system kernel settings against the IBM default limits, which should be sufficient for many systems. However, these settings might be insufficient if you have a busy system with high production volumes, or if your system is also hosting other products like databases or application servers. Run the `mqconfig` script when your system is processing its peak volume to ensure that the actual resource usage, which is shown as a percentage, is not excessively high. The `mqconfig` script is described in the IBM technote “How to configure UNIX and Linux systems for IBM MQ” at <http://www.ibm.com/support/docview.wss?uid=swg21271236>

On Windows, ensure that the name of the server does not contain any spaces. Also, ensure that any user ID and group names do not exceed 20 characters. The MQ administrator group and user ID are automatically created during the installation process.

IBM MQ on-premises installation

Similar to installing other software on the same operating system

- IBM AIX
 - SMIT
 - Easy installation
- HP-UX
 - swinstall
- Linux
 - Red Hat Package Manager
- Oracle Solaris
 - Run the supplied installation script from the CD-ROM
- Windows
 - Automatic execution of setup from CD-ROM

Figure 2-26. IBM MQ on-premises installation

Installing IBM MQ is like installing any other software on the same operating system and uses many of the same installation tools.

For example, you can use SMIT to install MQ for AIX, or you can choose the easy installation. The easy installation places a minimal, typical set of components on your system. It excludes, for example, the online documentation and the application development support.

During installation, MQ automatically creates an MQ configuration file, which contains information relevant to all queue managers that are created on the system. Only one MQ configuration file exists for each system. You learn more about this configuration file later in this course.

Finding more information

- Latest details of hardware and software requirements on all supported operating systems on the IBM MQ requirements website:
<http://www.ibm.com/support/docview.wss?uid=swg27006467>
- Latest product support information on the IBM MQ support website:
http://www.ibm.com/support/entry/portal/product/websphere/websphere_mq?productContext=24824631
- Product readme file (`readme.html`) for information about last-minute changes and known problems and workarounds

Figure 2-27. Finding more information

Always review the IBM MQ product websites to see the most recent specifications, requirements, and product support information.

The MQ installation files include a readme file that contains information about last-minute changes, known problems, and workarounds.

Uninstalling IBM MQ

- Uninstall IBM MQ Explorer before uninstalling IBM MQ
- Ensure that all IBM MQ programs and processes are stopped
- If running IBM MQ with Microsoft Cluster Service (MSCS), remove queue managers from the MSCS control before uninstalling IBM MQ

- Methods for uninstalling IBM MQ on Windows:
 - Start the installation process and then select the appropriate option
 - Use the **Add or Remove Programs** option in the Windows control panel

IBM MQ installation and deployment options

© Copyright IBM Corporation 2017

Figure 2-28. Uninstalling IBM MQ

If necessary, you can uninstall MQ. Ensure that the applications and services are stopped before you attempt to uninstall IBM MQ.

On Windows, you can use the **Add or Remove Programs** application in the Windows **Control Panel** to uninstall MQ components and services.

You can also uninstall MQ from a command if it was installed in the default location. For example, on Linux, you would complete the following steps to uninstall MQ:

1. Type: `installp -u mqm`
2. Delete the contents of the MQ data directory.

Unit summary

- Summarize the IBM MQ installation options for on-premises and Cloud implementations
- Find the hardware and software prerequisites for an IBM MQ on-premises installation
- List the steps that are required to install IBM MQ on distributed operating systems
- Locate and install IBM MQ Fix Packs
- Locate IBM MQ SupportPacs

Review questions

1. True or False: Only one IBM MQ installation can own a specific queue manager at a time.
2. True or False: On Windows systems, the installation name is automatically assigned during the installation process.



IBM MQ installation and deployment options

© Copyright IBM Corporation 2017

Figure 2-30. Review questions

Write your answers here:

- 1.
- 2.

Review answers

1. True or False: Only one IBM MQ installation can own a specific queue manager at a time.
The answer is True.

2. True or False: On Windows systems, the installation name is automatically assigned during the installation process.
The answer is: False. On Windows systems, you can choose the installation name during the installation process.



Unit 3. Creating a queue manager and queues

Estimated time

01:00

Overview

In this unit, you learn how to use the IBM MQ commands and command scripts to verify an installation and create a queue manager and local queues.

How you will check your progress

- Review questions
- Hands-on exercise

References

IBM Knowledge Center for IBM MQ V9

Unit objectives

- Use IBM MQ commands to create a queue manager and local queues
- Use IBM MQ commands to start and stop a queue manager
- Create IBM MQ command scripts

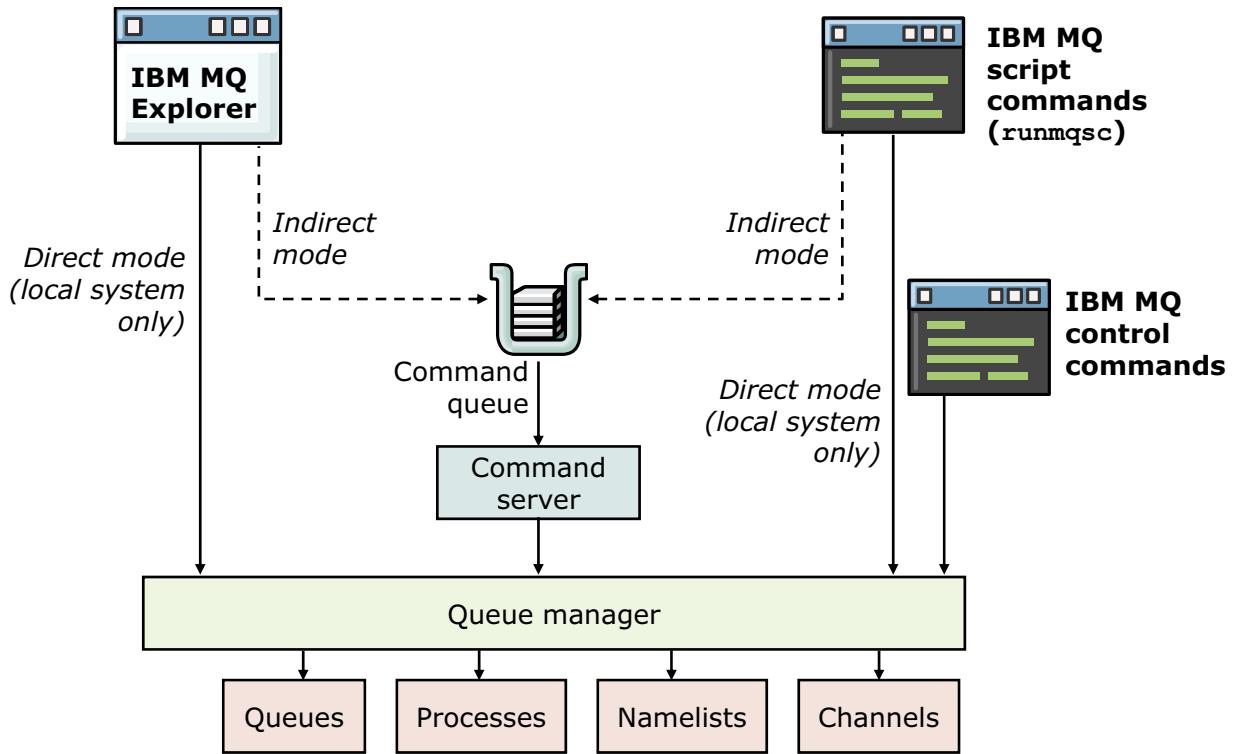
Creating a queue manager and queues

© Copyright IBM Corporation 2017

Figure 3-1. Unit objectives



Administration interfaces



Creating a queue manager and queues

© Copyright IBM Corporation 2017

Figure 3-2. Administration interfaces

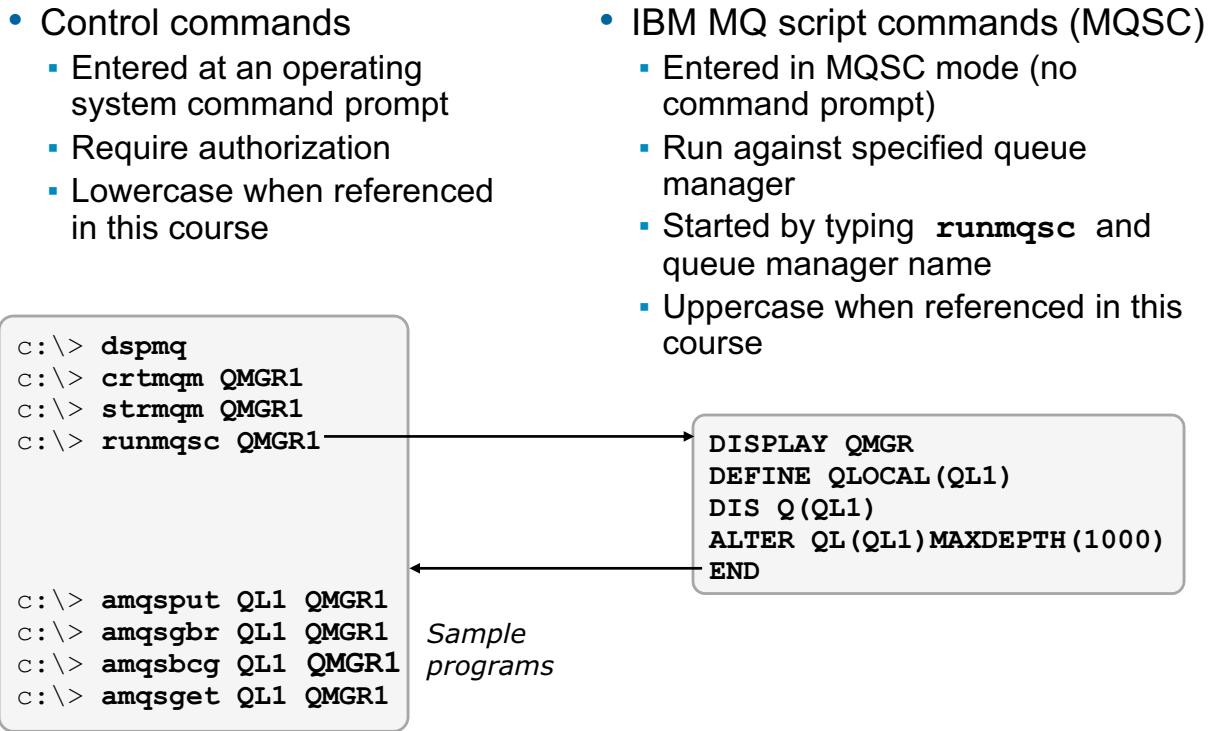
The primary IBM MQ administration interfaces for MQ on distributed operating systems are:

- MQ Explorer
- MQ script commands
- MQ control commands

MQ Explorer is a graphical user interface that runs on Linux and Windows. It can connect to, control, and configure MQ on all operating systems. IBM MQ Explorer is described in detail later in this course.

MQ script commands and MQ control commands are described in more detail in this unit and throughout this course.

IBM MQ command modes



Creating a queue manager and queues

© Copyright IBM Corporation 2017

Figure 3-3. IBM MQ command modes

As shown in the figure, MQ supports two command modes: control command mode and MQSC mode.

Control commands are entered at the operating system command prompt. They can also be included in an operating system command file such as a shell script on UNIX or a batch file in Windows. The sample programs that are included with MQ are run with control commands.

You need the following authority to use a control command.

- On UNIX and Linux, your user ID must belong to the group “mqm”, but not necessarily as its primary group.
- On Windows, your user ID must belong either to the “mqm” local group or to the “Administrators” local group. It is possible for your user ID to belong to a global group, which, in turn, belongs to either of these local groups.

MQSC mode is started by entering the `runmqsc` control command. MQSC commands can be entered interactively, by typing a command at the keyboard and waiting for the result. Alternatively, you can use a text editor to create a file that contains a sequence of MQSC commands and then run the file.

The figure shows examples of control commands and MQSC commands. In the example, the control command `runmqsc QMGR1` starts MQSC mode for the queue manager that is named

QMGR1. The commands in the MQSC mode are sent to QMGR1 until the `END` command is sent and the mode returns to control command mode.



Important

The operating system command prompt is not available when the system is in MQSC mode.

MQSC commands can be entered in uppercase or lowercase. In this course, MQSC commands are always shown in uppercase to help distinguish them from MQ control commands that are shown in lowercase.

Creating a queue manager

- Use `crtmqm` command to create a queue manager and define default and system objects
- Can have multiple queue managers on a server, virtual machine, or appliance
 - Specify unique queue manager name and listener port for each queue manager
- Optionally, can specify:
 - TCP listener port number (`-p`)
 - Name of dead-letter queue (`-u`) that holds undeliverable messages
 - That this queue manager is the default queue manager (`-q`)
 - On UNIX and Linux, allow authorization definitions for both users and groups (`-oa user`)
 - Logging options

Example: Create a queue manager that is named QMR01 on port 1415 that uses the local queue that is named QMR01.DLQ as its dead-letter queue

```
crtmqm -u QMR01.DLQ -p 1415 QMR01
```

Creating a queue manager and queues

© Copyright IBM Corporation 2017

Figure 3-4. Creating a queue manager

After installation, the first MQ object to create is a queue manager by using the `crtmqm` control command. For the complete description of the `crtmqm` control command, see the IBM MQ product documentation.

Typically, you must create only one queue manager per system, but you can create multiple queue managers that are based on needs, such as security or separation of business units, testing, or development purposes.

Every queue manager has a name, which should be unique within a network of queue managers that exchange messages with each other. When a queue manager generates a unique message identifier for a message, it uses the first 12 characters of its name as part of the identifier.

Queue manager names are case-sensitive. Always use uppercase characters to avoid problems that can be caused by using the incorrect case.

Plan conventions for queue manager names. For example, do not use host names as queue manager names when possible. If the host names change or frequent server reassignments occur, then the use of host names as queue names can lead to problems over time. Some large organizations use host names that are combined with Domain Name System (DNS) servers, which can decrease the configuration work that is required when queue managers move to different dedicated IBM MQ servers. Most organizations successfully use geographic area, the application that the MQ queue manager is serving, or a combination of the two.

You can define a dead-letter queue and a default transmission queue when you create the queue manager. You can also identify one queue manager on the system as the default queue manager.

If you are creating a queue manager for some simple tests, accept the default values for all the parameters unless a reason exists to use different ones.

You can modify some queue manager attributes later by using the **ALTER QMGR** MQSC command.

Queue manager default and system objects

- System and default objects are queues that are created automatically when you create the queue manager
 - System objects are those IBM MQ objects that are needed to operate a queue manager or channel
 - Default objects define all the attributes of an object, such as a local queue
 - Identified by SYSTEM queue prefix
- Examples
 - SYSTEM.ADMIN.ACCTING.QUEUE holds accounting monitoring data
 - SYSTEM.DEFAULT.LOCAL.QUEUE defines default attributes for a local queue

Figure 3-5. Queue manager default and system objects

The process of creating a queue manager also creates the processes, special queues that the queue manager needs for operation.

The queue manager queues are identified with the SYSTEM prefix. For example, the queue manager uses SYSTEM.ADMIN.ACCTING.QUEUE to hold accounting monitoring data.

Queue manager logs

- Circular logging (**crtmqm -1c**)
 - Provides restart recovery by using the log to roll back any transactions that were in flight when the queue manager stopped
 - Logs kept in a ring of files where older files are reused when filled
 - Default logging method
- Linear logging (**crtmqm -1l**)
 - Keeps the log data in a continuous sequence of files
 - Can re-create lost or damaged data by replaying log contents (media recovery)
 - Must specify when creating the queue manager
 - Log files are not reused
 - Number of logs can exceed capacity
 - Requires log archival maintenance



When log space runs out, the queue manager stops

[Creating a queue manager and queues](#)

© Copyright IBM Corporation 2017

Figure 3-6. Queue manager logs

MQ supports two ways of maintaining records of queue manager activities: circular logging and linear logging.

The type of logging and disk space to allocate are among the most important considerations to make when you create an MQ infrastructure.



Important

You cannot change the queue manager log type after the queue manager is created.

Circular logging keeps all restart data in a ring of log files. Logging fills the first file in the ring, then moves on to the next, until all the files are full. It then goes back to the first file in the ring and starts again. This process continues while the product is in use, and has the advantage that you never run out of log files. When a queue manager is created on a distributed operating system, the default log type is circular.

Linear logging keeps the log data in a continuous sequence of files. Space is not reused, so you can always retrieve any record that is logged in any log extent that was not deleted. The number of log files that are used with linear logging can be large, depending on your message flow and the

age of your queue manager. As disk space is finite, you might have to think about some form of archiving.

If you want both restart recovery and media recovery to re-create lost or damaged data by replaying the contents of the log, use linear logging. When linear logging is used, MQ tracks which logs are no longer necessary. However, MQ does not discard these files. The log maintenance must be implemented as another automated system process that uses the operating system-specific file utilities.



Important

Linear logging requires regular maintenance to discard logs that are no longer needed. If log maintenance is not done, log space is exhausted and the queue manager stops.

An organization should always establish standards on how to configure logging for a queue manager. Organizations also need to institute automated methods and scripts to maintain the logs.

Naming IBM MQ objects

- Allowable character set: **A – Z, a – z, 0 – 9**, and the characters **. / % _**
- Maximum of 48 characters for the names of:
 - Queue managers
 - Queues
 - Process definitions
 - Namelists
 - Clusters
 - Listeners
 - Services
 - Authentication information objects
 - Topics and subscriptions
- Maximum of 20 characters for the names of channels
- No implied structure in a name
- Object names that begin with **SYSTEM** are reserved
- Names in IBM MQ are case-sensitive

[Creating a queue manager and queues](#)

© Copyright IBM Corporation 2017

Figure 3-7. Naming IBM MQ objects

The figure lists some of the rules for naming MQ objects such as queue managers and queues.

In general, MQ object names are limited to 48 characters. Period, forward slash, percent sign, and underscore are special characters that are allowed in MQ object names. National language characters are not allowed.

Names can be enclosed in double quotation marks. If you use the forward slash or percent sign characters in a name, the name must be enclosed in double quotation marks.

Leading or embedded blanks are not allowed in MQ object names.

Channel names are limited to 20 characters but otherwise follow the standard rules for naming MQ objects.

The rules for naming MQ objects are the same on all supported operating systems. No implied structure exists in a name that you might find in the rules for file names on many operating systems.

Agree on all names before you start.

Names are case-sensitive. Be consistent in using uppercase and lowercase characters. Many organizations use uppercase characters for names, especially when z/OS queue managers are used within the MQ environment.

Basic queue manager control commands

- Start a queue manager

```
strmqm QMgrName
```

- Display names and details about all queue managers or a specific queue manager

```
dspmq  
dspmq -m QMgrName
```

- Delete a queue manager

```
dltmqm QMgrName
```



You must stop the queue manager before you can delete it

Figure 3-8. Basic queue manager control commands

When a queue manager is first created, it is created in a *stopped* state. To start the queue manager, use the **strmqm** command followed by the name of the queue manager.

To display the status and queue manager configuration information, use the **dspmq** command.

To delete a queue manager and its associated objects, use the **dltmqm** command. You must stop the queue manager before you can delete it.

The create queue manager command, **crtmqm**, and the commands that are listed in this figure are control commands. The name of the queue manager is a required parameter on some of these commands. It is a good practice to always include the queue manager name in the command to ensure that the command is run against the correct queue manager.

The control commands are described in IBM MQ product documentation.

Stopping a queue manager

- **Controlled (quiesced)**: Allows connected applications to end, but no new connections are allowed
- **Controlled (wait)**: Same as quiesced, except `endmqm` reports queue manager shutdown status periodically
- **Immediate**: Completes all current MQI calls, but no new ones
- **Preemptive**: Stops without waiting for applications to disconnect or for MQI calls to complete

`endmqm -c QMgrName`

`endmqm -w QMgrName`

`endmqm -i QMgrName`

`endmqm -p QMgrName`



- Use this type of shutdown only in exceptional circumstances, such as when a queue manager does not stop as a result of a normal `endmqm`

Figure 3-9. Stopping a queue manager

The control command to stop the queue manager is `endmqm`. Four options exist for controlling how the queue manager shuts down.

- **Controlled (or quiesced) shutdown**: The queue manager stops after all applications are disconnected. All new requests to connect to the queue manager fail. Controlled shutdown is the default mode.
- **Controlled shut down with wait**: End programs in the same manner as the controlled option. However, the command prompt does not return until the queue manager stops.
- **Immediate shut down**: The queue manager stops after it completes all the in-process MQI calls. Any MQI calls that are sent after this command is entered fail. Any incomplete units of work are rolled back when the queue manager is next started.
- **Preemptive shut down**: The queue manager stops without waiting for applications to disconnect or for MQI calls to complete. Use of this mode can lead to unpredictable results and should be used as the last option.

The normal mode of stopping a queue manager is the controlled, or quiesced, mode. In this mode, applications can continue to do work, but well-behaved applications disconnect as soon as it is convenient.

Use the immediate mode of stopping a queue manager only if you need to stop it quickly with predictable results.

Use the preemptive mode only if all other options to stop the queue manager fail.

Stopping and starting the IBM MQ service on Windows

- On Windows, IBM MQ runs under a Windows service
- If the Windows MQ service is not started automatically, or if the service ended, start the Windows MQ service
 - strmqsvc**
- Stop the Windows MQ service
 - endmqsvc**
- Restart the IBM MQ service to pick up a new environment, including new security definitions

Figure 3-10. Stopping and starting the IBM MQ service on Windows

MQ runs as a service on Windows. It might be necessary to restart the MQ service so that MQ recognizes a new environment.

You can restart the MQ service on Windows by using the Services Administrative Tool or by using the MQ control commands **endmqsvc** and **strmqsvc**.

Queue manager configuration file

- Queue manager configuration file (`qm.ini`) contains configuration attributes relevant to a queue manager
 - On UNIX and Linux, `qm.ini` is in the root of the directory tree that is occupied by the queue manager

Example: `/var/mqm/qmgrs/QMGR1/qm.ini`
 - On Windows, location of the `qm.ini` is given by the WorkPath specified in the HKLM\SOFTWARE\IBM\WebSphere MQ key

Example: `C:\ProgramData\IBM\MQ\qmgrs\QMNAME\qm.ini`
- Created automatically when queue manager is created
- Contains one or more stanzas, which are groups of lines in the file with a common function or which define part of a system
- Any changes usually do not take effect until you restart the queue manager

Figure 3-11. Queue manager configuration file

When a queue manager is created, a queue manager configuration file that is named `qm.ini` is automatically created at the same time. This file contains information that is relevant to a specific queue manager. One queue manager configuration file exists for each queue manager.

The queue manager configuration file contains texts and is readable. The contents of the file might change by certain commands and in special circumstances.

The queue manager configuration file contains configuration parameters that are grouped by function into stanzas.

Most changes to the queue manager configuration file are not recognized until the queue manager restarts.

Example queue manager configuration file stanzas

CHANNELS:

```
MaxChannels=n      ; Maximum channels allowed
MaxActiveChannels=n ; Maximum active channels allowed at any time.
                     ; Default is the value of MaxChannels
MaxInitiators=n   ; Maximum initiators allowed. Default and maximum
                     ; value is 3
MQIBINDTYPE=type1 ; Whether the binding for applications is "fastpath"
                     ; or "standard". Default is "standard".
ThreadedListener=  ; "YES" to run all channels run as threads of a single
                     ; job. "NO" to start a new responder job for each
                     ; inbound TCP/IP channel. Default is "NO".
AdoptNewMca=chltype ; Stops previous process if channel fails to start.
                     ; The default is "NO".
AdoptNewMcaTimeout= ; Amount of time that the new process waits for the
                     ; old process to end. Default is 60.
AdoptNewMcaCheck=  ; Type of checking required when enabling AdoptNewMca.
                     typecheck ; Default is "NAME", "ADDRESS", and "QM".
```

TCP:

```
Port=n            ; TCP entries
KeepAlive=NO      ; Port number, the default is 1414
ListenerBacklog=n ; Switch TCP/IP KeepAlive "ON" or "OFF"
ConnectTimeout=n  ; Maximum outstanding connection requests.
                     ; Seconds before an attempt to connect socket times
                     ; out. Default of zero specifies that there is no
                     ; connect timeout.
```

Creating a queue manager and queues

© Copyright IBM Corporation 2017

Figure 3-12. Example queue manager configuration file stanzas

This figure shows the CHANNELS and TCP stanzas in the queue manager configuration file.

You can modify the `qm.ini` file, use MQSC commands, or use the queue manager **Properties** pages in MQ Explorer to specify channel and network protocol configuration parameters.


Note

Only attributes that represent changes to the default values need to be specified in the queue manager configuration file.

For more information about the queue manager configuration file, see the IBM MQ product documentation.

When do you need to edit the queue manager configuration file?

- Edit a configuration file to recover from backup, move a queue manager, change the default queue manager, or assist IBM support
- You might need to edit a configuration file if:
 - You lose a configuration file
 - You need to move one or more queue managers to a new directory
 - You need to change your default queue manager, which can happen if you accidentally delete the existing queue manager
 - Your IBM Support Center advises you to do so
- Always create a backup copy of the configuration file before you edit it

Figure 3-13. When do you need to edit the queue manager configuration file?

Specific events might require that you edit the queue manager configuration file.

For example, it might be necessary to edit a queue manager configuration file to recover from a backup or to assist IBM support. This figure lists some other reasons why it might be necessary to edit the queue manager configuration file.

Be sure to always create a backup copy of the queue manager configuration file before you edit it.

Using MQSC to configure IBM MQ objects

- Command interface is started by entering `runmqsc` control command with a queue manager name
- Is used to run IBM MQ script commands and script files
- Is used to create, delete, modify, and display IBM MQ queue manager objects such as queues, channels, topics, and subscriptions

Creating a queue manager and queues

© Copyright IBM Corporation 2017

Figure 3-14. Using MQSC to configure IBM MQ objects

On all queue managers, the administration interface for creating, modifying, deleting, and displaying queue manager resources and objects is MQSC.

MQSC commands are known as script commands because they are frequently batched together to create a repeatable set of MQ resource definitions in a script file.

MQSC commands are also used to display the runtime status of objects like channels.

When you enter MQSC commands interactively, you can get help by typing the name of the command and a question mark character: `command?`

Starting the MQSC command interface

- Local queue manager: `runmqsc QMgrName`
- Remote queue manager: `runmqsc -w WaitTime -m LocalQMgrName RemoteQMgrName`
 - w WaitTime**
 - Time in seconds that MQSC waits for replies from remote queue manager
 - Assumes that required channel and transmission queues are configured for MQSC on remote queue manager
 - m LocalQMgrName**
 - Local queue manager to use to submit commands to remote queue manager
 - If this parameter is omitted, local default queue manager is used to submit commands to remote queue manager.
- Verify a command: `runmqsc -e -v`
 - e** Do not copy source text to output report
 - v** Perform syntax check only; this mode is only available locally

[Creating a queue manager and queues](#)

© Copyright IBM Corporation 2017

Figure 3-15. Starting the MQSC command interface

The command for starting MQSC mode is `runmqsc` followed by the name of a local queue manager.

If you are trying to connect to a remote queue manager, you must specify a local queue manager that can be used to submit commands to the remote queue manager. If the remote queue manager is on z/OS, then the `-x` option is required.

The input to `runmqsc` is zero, one, or more MQ script commands, and the output is the result of running those commands, including operator and error messages. Alternatively, you can create a file that contains a sequence of MQSC commands and then have them run with the results that are directed to a file.

MQSC has three modes of operation:

- **Verify (-v):** Input commands are read and checked but not run.
- **Direct:** Connect to a local queue manager and do not use any intermediate queues or channels.
- **Indirect:** Connect to a remote queue manager. For indirect connections, the appropriate channels and transmission queues must be established. This mode allows MQSC to connect and administer a remote queue manager if security is configured to allow administration.

The **-e** option on the `runmqsc` command prevents source text for the MQSC commands from being copied into a report. This option is useful when you enter commands interactively.

The **-w** option on the `runmqsc` command is used for indirect connections and specifies the time, in seconds, that MQSC waits for replies. Any replies that are received after this time are discarded, but the MQSC commands still run.

Stopping the MQSC command interface

- Use the **END**, **EXIT**, or **QUIT** commands to return to operating system command prompt

```
c:\> runmqsc QMGR1
DISPLAY QMGR
DEFINE QLOCAL(QL1)
DISPLAY QUEUE(QL1)
ALTER QLOCAL(QL1) MAXDEPTH(1000)
END
c:\>
```

[Creating a queue manager and queues](#)

© Copyright IBM Corporation 2017

Figure 3-16. Stopping the MQSC command interface

To exit MQSC mode and return to control command mode and the operating system prompt, enter **END**, **EXIT**, or **QUIT**. Optionally, you can enter the end-of-file (EOF) character for the operating system:

- On Windows, type Ctrl+Z, and press **Enter**.
- On UNIX or Linux, type Ctrl+D.

The example in the figure shows the control command that starts MQSC mode for the queue manager that is named QMGR1: **runmqsc QMGR1**

The next four statements are MQSC commands:

- **DISPLAY QMGR** displays information about the queue manager QMGR1.
- **DEFINE QLOCAL(QL1)** defines a local queue that is named **QL1**.
- **DISPLAY QUEUE(QL1)** displays information about the local queue that is named **QL1**.
- **ALTER QLOCAL(QL1) MAXDEPTH(1000)** changes the **MAXDEPTH** property for the local queue that is named **QL1**.

The **END** statement stops MQSC mode and returns control to the operating system.

MQSC command rules

- In most cases, script command format is:
`verb objectType objectName parameter1 ... parametern`
- Parameters are either keyword or keyword with value
Example: `TRIGGER TRIGTYPE (FIRST)`
- Keywords are not case-sensitive but string values are
- Parameter order is not significant although some exceptions do exist
- Some verbs and object names can be abbreviated
Example: `DEFINE` can be abbreviated to `DEF`
`QLOCAL` can be abbreviated to `QL`
- Repeat parameters are not allowed
- Plus sign (+) in command line indicates that the command is continued from the first nonblank character on the following line

Figure 3-17. MQSC command rules

The figure lists some of the rules for the specification of MQSC commands.

The command verb must always come first and is followed by, in most cases, an MQ object, and then optional keyword and keyword/value parameters.

Script commands are not case-sensitive, but names and parameter values are case-sensitive.

In most cases, parameter order is not important, and some verbs can be abbreviated. An exception to parameter order is that the parameter `CHLTYPE` must be the first parameter that is specified in `DEFINE CHANNEL` and `ALTER CHANNEL` commands on distributed operating systems.

Abbreviations are fixed and are described in the MQ product documentation under their relevant MQSC command entries as synonyms. For example, the synonym for `REFRESH CLUSTER` is `REF CLUSTER`. The synonym for `DEFINE QLOCAL()` is `DEF QL()`.

Parameters cannot be repeated within the same script command. For example, `ALTER QLOCAL(ANDREW) TRIGGER TRIGGER` is not valid. Repeating the negation of the same parameter, as in `ALTER QLOCAL(ANDREW) TRIGGER NOTRIGGER` is not valid.

Even though MQSC commands are not case-sensitive, this course represents all MQSC commands in uppercase to distinguish MQSC commands from control commands.

MQSC command summary

Verbs	MQ objects	Name of the MQ object
ALTER	Queues	<ul style="list-style-type: none"> • Most objects have specific names
CLEAR	Queue managers	<ul style="list-style-type: none"> • Wildcard value can be used for displays
DEFINE or DEF	Process definitions	
DELETE or DEL	Namelists	
DISPLAY or DIS	Authentication information	
PING	Channels	
PURGE	Client connection channels	
REFRESH	Listeners	
RESET	Topic objects	
RESOLVE		Examples:
RESUME		CREATE QLOCAL(APP.IN) +
SET		DEFPSIST(YES)
START		DIS QL(APP*) CURDEPTH
STOP		START CHANNEL(MQ00.MQ03)
SUSPEND		ALTER QMGR(QM00) DEADQ(NEW.DLQ)
		DIS QSTATUS(BILL.IN) LPUTDATE +
		LPUTTIME CURDEPTH

Creating a queue manager and queues

© Copyright IBM Corporation 2017

Figure 3-18. MQSC command summary

IBM MQ script commands are available to define, change, delete, start (where applicable), display, or do most of the day-to-day administration functions. MQSC commands start with a verb, and are followed by an MQ object name. Several attributes or qualifiers can follow the verb-object name combination.

Using filters in MQSC commands

- Use **WHERE** to specify a filter condition to display only those objects that satisfy the selection criterion of the filter condition
- Value can be an explicit value or a generic value with wildcard character

Example that uses queues:

```
QUEUE
DISPLAY QLOCAL WHERE (attribute operator value)
QSTATUS
```

LT	Less than
GT	Greater than
EQ	Equal to
NE	Not equal to
LE	Less than or equal to
GE	Greater than or equal to
CT	Contains
EX	Excludes
LK	Matches
NL	Does not match

Figure 3-19. Using filters in MQSC commands

You can use filters on the MQSC **DISPLAY** command to return a subset of data that matches a filter expression.

MQSC filtering example

- Show the queues that accept messages larger than 100,000 bytes:

```
DISPLAY QLOCAL(*) WHERE (MAXMSGL GT 100000)
```

- Show the queues with messages older than 5 minutes:

```
DISPLAY QSTATUS(*) WHERE (MSGAGE GT 300)
```

- Show the queues with a currently active *input* exclusive access:

```
DISPLAY QSTATUS(*) TYPE(HANDLE) WHERE (INPUT EQ EXCL)
```

Figure 3-20. MQSC filtering example

This figure shows examples of MQSC **DISPLAY** commands with filter expressions.

MQSC script files

- Use scripts to automate repetitive tasks
- Each command starts on new line
- Commands and keywords are not case-sensitive
- Blank line and lines that are prefixed with an asterisk (*) are ignored
- Restrict maximum line length to 72 characters for portability
- Last non-blank character determines continuation
 - Minus sign (-) continues command from start of next line
 - Plus sign (+) continues command from first non-blank character in next line

Example:

```
* Start the script with a descriptive comment
DEFINE QLOCAL(MY_DEAD_LETTER_Q) +
REPLACE

ALTER QMGR DEADQ(MY_DEAD_LETTER_Q)
DEF QL(ANOTHER) REPLACE +
DESCR('This is a test queue')
```

[Creating a queue manager and queues](#)

© Copyright IBM Corporation 2017

Figure 3-21. MQSC script files

In most cases, administrators create MQSC scripts to complete repetitive administrative functions. MQSC commands can be included in a text file and then run against a queue manager as a script.

This figure lists some of the rules for writing MQSC script files. More syntax rules for writing MQ commands include the following rules:

- An MQSC command can contain a string of characters. A string that contains blanks, lowercase characters, or special characters other than characters valid in the name of an MQ object, must be enclosed in single quotation marks. Lowercase characters that are not enclosed in single quotation marks are internally converted to uppercase.
- You can optionally use a semicolon (;) to end a command.
- A string that has no characters is not valid.
- The line length is limited to 72 characters. A plus sign (+) is a continuation character.

The figure shows three examples of MQSC commands.

- The first command creates a local queue with the name MY_DEAD_LETTER_Q.
- The second command alters the queue manager by declaring MY_DEAD_LETTER_Q as the dead-letter queue of the queue manager.

- The third command creates another local queue. A keyword, such as DESCRIPTOR, can be in lowercase or uppercase. In this course, all MQSC commands and keywords are shown in uppercase to distinguish them from control commands.

The single quotation marks that enclose the string This is a test queue are required because the string contains blanks.

Running MQSC script files

- Redirect the input from a file to run a sequence of frequently used commands that are contained in the file

Example: `runmqsc QMgrName < mqsc.in`

- Redirect the input from a file and redirect the output report to a file

Example: `runmqsc QMgrName < mqsc.in > mqsc.out`

Figure 3-22. Running MQSC script files

As shown in the figure, the `runmqsc` command can be used to process scripts. It reads from the *standard input device* and writes to the *standard output device*. Typically, the standard input device is the keyboard and the standard output device is the display. However, by using redirection operators, input can be taken from a file and output can be directed to a file.

In the first example, the queue manager that is identified with the `runmqsc` command reads and processes the file that is named `mqsc.in`.

In the second example, the queue manager that is identified with the `runmqsc` command reads and processes the file that is named `mqsc.in`. A report that shows the processing of each command is created in the `mqsc.out` file.

It is useful to maintain MQSC files, for the following reasons:

- To replicate a queue manager configuration on multiple systems
- To recover a queue manager configuration
- When you go through a number of iterations in testing a queue manager configuration

Defining a local queue

- Use **DEFINE QLOCAL** to define a local queue
 - **DEFINE** can be abbreviated to **DEF**
 - **QLOCAL** can be abbreviated to **QL**
- **DEFINE QLOCAL** with **REPLACE** takes unspecified attributes from the object that is named on **LIKE** parameter or from the default definition of local queue (SYSTEM.DEFAULT.LOCAL.QUEUE)
- Queue names are case-sensitive
- Useful naming conventions
 - Name the queue to describe its function, not its type or location
 - Use a common prefix for names of related queues to simplify administration

Figure 3-23. Defining a local queue

The MQSC command **DEFINE QLOCAL** or its synonym of **DEF QL** defines a queue that is local to the queue manager.

Keywords and their values specify the values of attributes of the local queue that is created. The values of the attributes that are not explicitly defined are taken from the values of the corresponding attributes of the default local queue, SYSTEM.DEFAULT.LOCAL.QUEUE. The SYSTEM.DEFAULT.LOCAL.QUEUE is created during MQ installation.

Every queue is owned by a queue manager and possesses a name. An application identifies a queue by its name. Queue names in MQ are case-sensitive. Use a standard convention for case such as all uppercase or all lowercase.

Other queue name guidelines include the following rules:

- Do not use the type or location of the queue in the queue name. If a queue is changed from a local to a remote queue, for example, the same name can still be used for the queue; it is not necessary to change the applications that reference the queue. Instead, name the queue after its function.
- Use a common prefix for the names of related queues to aid administration. For example, name all queues that are related to the same application with the same prefix.

Defining a local queue examples

```
* Define a local queue that is named QL.APPINPUT
DEFINE QLOCAL(QL.APPINPUT)

* Define a local queue that is named QL.TESTQ and replace
* some of the default definition parameters

DEF QL(QL.TESTQ) REPLACE +
DESCR('This is a local test queue') +
PUT(ENABLED) GET(ENABLED) +
DEFPSIST(YES) +
MAXDEPTH(1000) MAXMSGL(2000)

* Define a local queue that is named QL.TESTQ2 by using
* QL.TESTQ as the model

DEF QL(QL.TESTQ2) LIKE(QL.TESTQ)
```

Figure 3-24. Defining a local queue examples

The figure shows three examples of the MQSC command for defining a local queue. The second and third examples use the synonym **DEF QL**.

The first example creates a local queue that is named QL.APPINPUT and uses the default attributes of SYSTEM.DEFAULT.LOCAL.QUEUE for a local queue.

The second example creates a local queue that is named QL.TESTQ and replaces some of the default attributes. The **REPLACE** keyword indicates that if the queue exists, replace its definition with the new one as defined in this command. Any messages on an existing queue are retained.

The third example defines a local queue that is named QL.TEST2. The **LIKE** keyword means that the named queue (QL.TESTQ in this example) is used for the default values of attributes rather than the default local queue SYSTEM.DEFAULT.LOCAL.QUEUE.

This unit provides an overview of command syntax. You learn more about creating queues and queue attributes throughout this course and in the lab exercises.

Defining other queue types

- Alias queue

- Provides a level of indirection to another queue or a topic object
- Uses SYSTEM.DEFAULT.ALIAS.QUEUE for default definition

```
DEFINE QALIAS (Qalias) TARGET (Qname)
```

- Local definition of a remote queue (remote queue definition)

- Local definition of a remote queue, queue manager alias, or a reply-to queue alias
- Uses SYSTEM.DEFAULT.REMOTE.QUEUE for default definition

```
DEFINE QREMOTE (Qname) RNAME (RemoteQ) RQMNAME (RemoteQmgr)
```

- Model queue

- Queue template for creating dynamic queues
- Uses SYSTEM.DEFAULT.MODEL.QUEUE for default definition

```
DEFINE QMODEL (Qname)
```

Figure 3-25. Defining other queue types

MQ supports other queue types.

An *alias queue* is an MQ object that refers indirectly to another queue. The MQSC command to create an alias queue is **DEFINE QALIAS** or **DEF QA**. The **TARGET** keyword specifies the name of the queue to which the alias queue resolves. The target queue can be a local queue or a local definition of a remote queue.

A *local definition of a remote queue*, or a *remote queue definition*, is an MQ object owned by one queue manager that refers to a queue owned by another queue manager. The MQSC command to create a local definition of a remote queue is **DEFINE QREMOTE** or **DEF QR**. The keyword **RNAME** is the name of the queue as it is known on the remote queue manager. The keyword **RQMNAME** is the name of the remote queue manager.

A *model queue* is an MQ object whose attributes are used as a template for creating a dynamic queue. The MQSC command to create a model queue is **DEFINE QMODEL**. When an application opens a model queue, the queue manager creates a dynamic queue. The **DEFTYPE** keyword specifies whether a dynamic queue that is created from the model queue is one of the following types:

- A *temporary dynamic queue*, which is deleted when it is closed and does not survive a queue manager restart

- A *permanent* dynamic queue, whose deletion on the MQCLOSE call is optional and which does survive a queue manager restart.

Of the two types of dynamic queues, only a permanent dynamic queue can store persistent messages. A typical use of a dynamic queue is as a reply-to queue for a client program that is sending requests to a server.

Of the four types of queues, only a local queue can store messages.

Clearing and deleting queues

- Delete all messages on a local queue

```
CLEAR QLOCAL (Qname)
```

- Delete a queue

```
DELETE QLOCAL (Qname) or DEL QL (Qname)  
DELETE QREMOTE (Qname) or DEL QR (Qname)  
DELETE QALIAS (Qname) or DEL QA (Qname)
```



If the queue is in use, commands fail

Figure 3-26. Clearing and deleting queues

This figure shows the MQSC commands that can be used to clear all messages on a queue and delete a queue. These commands fail if the queue:

- Contains uncommitted messages that are put on the queue under sync point
- Is open by an application (with any open options)
- Is a transmission queue, and any queue that is, or resolves to, a remote queue that references this transmission queue, is open

Displaying queue manager and queue attributes

- Display all attributes of a specific local queue

```
DISPLAY QLOCAL (Qname)
```

- Display all or selected attributes of one or more queues of any type

```
DISPLAY QUEUE (Qname) DESCRIPTOR GET PUT
DISPLAY QUEUE (Qname) MAXDEPTH CURDEPTH
```

- Wildcards are allowed to display attributes for multiple queue managers and queues

```
DIS Q(SYSTEM*)
```

- Display all attributes of the queue manager object

```
DISPLAY QMGR
```

Figure 3-27. Displaying queue manager and queue attributes

The MQSC command to display all the attributes of a specific queue is **DISPLAY** or its synonym **DIS** followed by the queue type keyword (**QLOCAL**, **QALIAS**, **QREMOTE**, or **QMODEL**).

To display selected attributes for one or more queues, use the **DISPLAY QUEUE** command. This command applies to all types of queue: local, alias, remote, and model. The **DISPLAY QUEUE** command can specify a generic queue name by using a wildcard trailing asterisk (*). An asterisk without any other characters specifies “all queues”.

The default behavior of the **DISPLAY QUEUE** command is to display all queue attributes. You can request the display of selected attributes by using a **WHERE** clause.

The MQSC command to display the attributes of the queue manager object is **DISPLAY QMGR**. Do not enter the name of the queue manager in the command.

The default action of the **DISPLAY QMGR** command is to display all the attributes. You can choose to display different sets of queue manager parameters, such as **CHINIT**, **CLUSTER**, **EVENT**, **SYSTEM**, and **PUBSUB**.

Display queue manager example

DIS QMGR

```
AMQ8409: Display Queue Manager
details
QMNAME (QMGR01)
. . .
CCSID(500)
CHLEV(DISABLED)
. . .
CMDLEVEL(800)
CONFIGEV(DISABLED)
CONNAUTH(SYSTEM.DEFAULT.AUTHINFO.IDP
WOS)
. . .
DESCR(Windows v9.0.0 QM)
. . .
INHIBTEV(DISABLED)
LOCALEV(DISABLED)
. . .
MAXMSGL(4194304)
MAXPRTY(9)
. . .
PERFMEV(DISABLED)
. . .
STRSTPEV(ENABLED)
SYNCPT(AVAILABLE)
```

```
. . .
AUTHOREV(DISABLED)
CERTLBL(ibmWebSphereMQ00)
. . .
CHLAUTH(ENABLED)
. . .
CHAD(DISABLED)
. . .
CMDEV(DISABLED)
COMMANDQ(SYSTEM.ADMIN.COMMAND.QU
EUE)
. . .
DEADQ(SYSTEM.DEAD.LETTER.QUEUE)
. . .
MAXHANDS(256)
. . .
MAXUMSGS(10000)
. . .
PLATFORM(WINDOWS)
. . .
REMOTEEV(DISABLED)
. . .
SSLEV(DISABLED)
. . .
VERSION(09000000)
```

[Creating a queue manager and queues](#)

© Copyright IBM Corporation 2017

Figure 3-28. Display queue manager example

This figure shows an example of the results from entering the **DIS QMGR** command. This example shows a subset of the results only.

Displaying queue attributes example

DIS QL(APP.QUEUE)

Amq8409: Display Queue details

QUEUE (APP.QUEUE)	TYPE (QLOCAL)
ACCTQ (QMGR)	ALTDATE (2016-09-01)
ALTTIME (14.18.13)	BOQNAME ()
BOTHRESH (0)	CLUSNL ()
CLUSTER ()	CLCHNAME ()
CLWLPRTY (0)	CLWLRANK (0)
CLWLUSEQ (QMGR)	CRDATE (2016-09-01)
CRTIME (14.18.13)	CURDEPTH (0)
CUSTOM ()	DEFBIND (OPEN)
DEFPRTY (0)	DEFPSIST (NO)
DEFPRESP (SYNC)	DEFREADA (NO)
DEFSOPT (SHARED)	DEFTYPE (PREDEFINED)
DESC ()	DISTL (NO)
GET (ENABLED)	HARDENBO
INITQ ()	IPPROCS (0)
MAXDEPTH (5000)	MAXMSG (4194304)
MONQ (QMGR)	MSGDLUSQ (PRIORITY)
NOTRIGGER	NPMCLASS (NORMAL)
OPPROCS (0)	PROCESS ()
PUT (ENABLED)	PROPCCTL (COMPAT)
QDEPTHHI (80)	QDEPTHLO (20)
QDPHIEV (DISABLED)	QDPLOEV (DISABLED)
QDPMQXEV (ENABLED)	QSVCIEV (NONE)
QSVCINT (999999999)	RETINTVL (999999999)
• • •	• • •

Creating a queue manager and queues

© Copyright IBM Corporation 2017

Figure 3-29. Displaying queue attributes example

This figure shows an example of the results that are returned from the **DISPLAY QLOCAL (DIS QL)** command.

Modifying attributes

- **ALTER** to set specified attributes only
 - Alter specified attributes on a queue manager


```
ALTER QMGR DESCRIPTOR('New description')
```
 - Alter specified attributes on a queue


```
ALTER QLOCAL(Qname) PUT(DISABLED)
ALTER QALIAS(AliasQ) TARGET(Qname)
```
 - When using scripts, consider updating the original definition and rerunning the entire script instead of using **ALTER**
- **DEFINE** with **REPLACE** to set all attributes
 - Unspecified attributes are taken either from the object that is named on **LIKE** parameter or from default definition

[Creating a queue manager and queues](#)

© Copyright IBM Corporation 2017

Figure 3-30. Modifying attributes

The MQSC command **ALTER QMGR** is used to modify queue manager attributes.

The MQSC commands **ALTER QLOCAL**, **ALTER QALIAS**, **ALTER QREMOTE**, and **ALTER QMODEL** change the specified attributes of the queue to which the command is applied. The remaining attributes of the queue are left unchanged.

If you attempt to alter a queue and the queue is in use, the command fails.

Special queues

- Dead-letter queue for messages that cannot be delivered
 - One per queue manager
 - Always identify a dead-letter queue for each queue manager
- Initiation queues for triggering
- Transmission queues
- Command queue
- Event queues
- Default queues

[Creating a queue manager and queues](#)

© Copyright IBM Corporation 2017

Figure 3-31. Special queues

Some local queues have special purposes in MQ. You learn more about these special-purpose queues throughout this course.

- A *dead-letter queue* is a designated queue where a queue manager puts messages that cannot otherwise be delivered. It is not mandatory for a queue manager to have a dead-letter queue, but it is a good practice and critical to the health of the queue manager. The SYSTEM queue SYSTEM.DEAD.LETTER.QUEUE is not automatically assigned as the queue manager's dead-letter queue but can be assigned as the dead-letter when the queue manager is created.
- An *initiation queue* is a queue that is used to implement triggering.
- *Transmission queues* are queues that temporarily store messages that are destined for remote queue managers.
- The *command queue*, SYSTEM.ADMIN.COMMAND.QUEUE, is a local queue to which suitably authorized applications can send MQSC commands for processing. The MQ command server retrieves these commands. The command server validates the commands, passes the valid ones on for processing by the queue manager, and returns any responses to the appropriate reply-to queue.
- When a queue manager detects an instrumentation event, which can be either a channel, queue manager, performance, or logger event, the queue manager puts a message that

describes that event on an *event queue*. A system management application can monitor an event queue, get the messages put on these queues, and then take appropriate action.

- The purpose of the *default queues* is to identify the default values of the attributes of any new queue that you create. One default queue exists for each of the four types of queues: local, alias, remote, and model. You need to include in the definition of a queue only those attributes whose values are different from the default values when you create a queue. You can change the default value of an attribute by redefining the appropriate default queue.

Defining a dead-letter queue

Option 1:

- Use the SYSTEM queue SYSTEM.DEAD.LETTER.QUEUE as the dead-letter queue when you create the queue manager

Example: `crtmqm -u SYSTEM.DEAD.LETTER.QUEUE QMGR01`

Option 2:

1. Identify a user-defined dead-letter queue when you create the queue manager

Example: `crtmqm -u QMGR01.DLQ QMGR01`

2. Create dead-letter queue as a local queue on the queue manager

Example: `runmqsc QMGR01
DEF QL(QMGR01.DLQ)
END`

Option 3:

1. Create queue manager and do not identify a dead-letter queue
2. Create a local queue and then alter queue manager to use local queue for the dead-letter queue

Example: `DEF QL(APP.DLQ)
ALTER QMGR DEADQ(APP.DLQ)`

Creating a queue manager and queues

© Copyright IBM Corporation 2017

Figure 3-32. Defining a dead-letter queue

You can define a dead-letter queue for a queue manager by using one of three methods.

The first option is to identify the dead-letter queue when you create the queue manager with the `-u` option. In the example, the command that creates the queue manager that is named **QMGR01** identifies the SYSTEM dead-letter queue that is named **SYSTEM.DEAD.LETTER.QUEUE** as its dead-letter queue.

The second option is to identify a user-defined dead-letter queue when you create the queue manager with the `-u` option. In the example, the command that creates the queue manager that is named **QMGR01** identifies the dead-letter queue that is named **QMGR01.DLQ** as its dead-letter queue. Then, after the queue manager is created, the user-defined queue is created on the queue manager by using the `DEF QL` command.

The third option is to assign the dead-letter queue after the queue manager is created by using the MQSC `ALTER QMGR` command. In this third option, step 2 creates a local queue that is named **APP.DLQ** and then modifies the queue manager `DEADQ` attribute.

Unit summary

- Use IBM MQ commands to create a queue manager and local queues
- Use IBM MQ commands to start and stop a queue manager
- Create IBM MQ command scripts

Creating a queue manager and queues

© Copyright IBM Corporation 2017

Figure 3-33. Unit summary

Review questions (1 of 2)

1. True or False: Queue manager names can contain any printable ASCII character.
2. Which of the following queue types can be the target of an alias queue?
 - A. Another alias queue
 - B. Local queues
 - C. Model queues
3. Any local queue can be a dead-letter queue if it:
 - A. Is identified as the dead-letter queue to the queue manager
 - B. Has PUT enabled
 - C. Is not in use by any other application
4. True or False: You can alter queue attributes while the queue manager is running, and the changes take effect immediately.

Creating a queue manager and queues

© Copyright IBM Corporation 2017

Figure 3-34. Review questions (1 of 2)

Write your answers here:

- 1.
- 2.
- 3.
- 4.

Review questions (2 of 2)



5. What does the command `crtmqm -q -u DLQ CHIWINSVR` create?
- A. A queue manager that is named `DLQ`
 - B. A default queue manager that is named `CHIWINSVR` with queue `DLQ` assigned to the queue manager as the dead-letter queue
 - C. A queue manager that is named `CHIWINSVR`
 - D. A queue manager that is named `CHIWINSVR` with a default transmission queue `DLQ` assigned to the queue manager

Figure 3-35. Review questions (2 of 2)

Write your answer here:

5.

Review answers (1 of 2)

1. True or False: Queue manager names can contain any printable ASCII character.
The answer is False. Queue manager names do not support the entire set of printable ASCII characters.

2. Which of the following queue types can be the target of an alias queue?
 - A. Another alias queue
 - B. Local queues
 - C. Model queuesThe answer is B.

3. Any local queue can be a dead-letter queue if it:
 - A. Is identified as the dead-letter queue to the queue manager
 - B. Has PUT enabled
 - C. Is not in use by any other applicationThe answer is A.



Creating a queue manager and queues

© Copyright IBM Corporation 2017

Figure 3-36. Review answers (1 of 2)

Review answers (2 of 2)



4. True or False: You can alter queue attributes while the queue manager is running, and the changes take effect immediately.
The answer is True.
5. What does the command `crtmqm -u DLQ -q CHIWINSVR` create?
A. A queue manager that is named DLQ
B. A default queue manager that is named CHIWINSVR with queue DLQ assigned to the queue manager as the dead-letter queue
C. A queue manager that is named CHIWINSVR
D. A queue manager that is named CHIWINSVR with a default transmission queue DLQ assigned to the queue manager
The answer is B.

Exercise: Using commands to create a queue manager and queues

© Copyright IBM Corporation 2017

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Figure 3-38. Exercise: Using commands to create queue managers and queues

In this exercise, you create a queue manager, start it, and then create queues.

Exercise objectives

- Use IBM MQ commands to create a local queue manager, local queues, and alias queues
- Use IBM MQ commands to display and alter queue manager and queue attributes
- Create and run an IBM MQ command file

Creating a queue manager and queues

© Copyright IBM Corporation 2017

Figure 3-39. Exercise objectives

See the *Course Exercises Guide* for the exercise description and detailed instructions.

Unit 4. Introduction to IBM MQ Explorer

Estimated time

00:30

Overview

IBM MQ Explorer is a graphical interface for administering IBM MQ. In this unit, you learn how to use IBM MQ Explorer to administer IBM MQ objects.

How you will check your progress

- Review questions
- Hands-on exercise

References

IBM Knowledge Center for IBM MQ v9

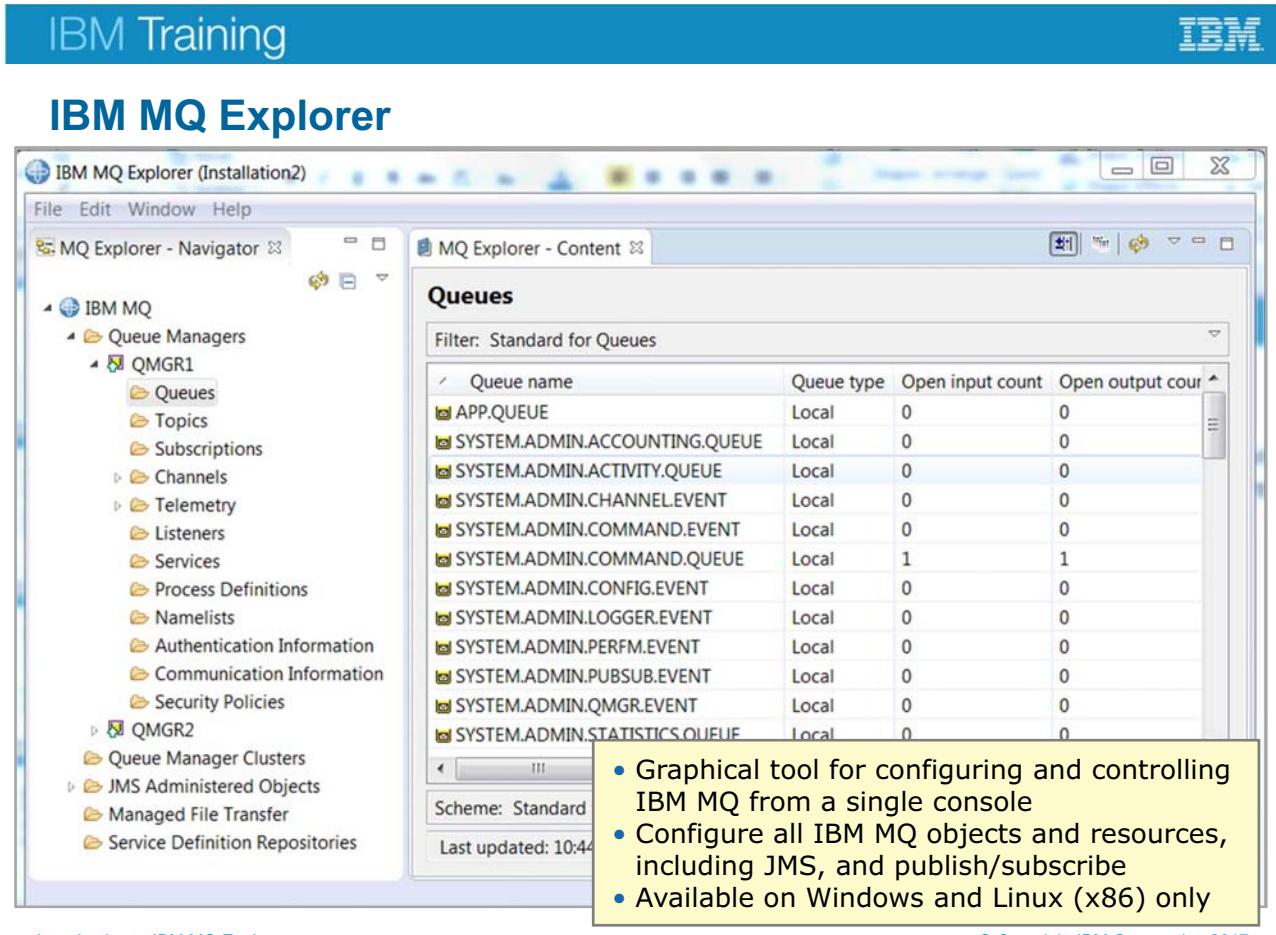
Unit objectives

- Use IBM MQ Explorer to create a local queue manager and queues
- Use IBM MQ Explorer to create and manage queue manager sets
- Use IBM MQ Explorer to run tests to verify IBM MQ object definitions

Introduction to IBM MQ Explorer

© Copyright IBM Corporation 2017

Figure 4-1. Unit objectives



Introduction to IBM MQ Explorer

© Copyright IBM Corporation 2017

Figure 4-2. IBM MQ Explorer

IBM MQ Explorer is the graphical user interface for administering and monitoring IBM MQ components, whether your local computer or a remote server hosts them.

MQ Explorer also supports:

- Advanced filtering
- Management of application connections
- Grouping of queue managers to simplify management
- Publish/subscribe administration

MQ Explorer is built as a rich client platform (RCP) application on Eclipse. It runs on Windows and Linux (x86) operating systems but can be used to remotely manage MQ objects on other operating systems.

All functions that were provided in the previous release are available with the new version of MQ.

Starting IBM MQ Explorer

- On Windows, can be started from:
 - Windows **Start > All Programs** menu
 - `strmqcfg.cmd`
- On Linux, can be started from:
 - System menu
 - `<INSTALL_DIR>/bin/MQExplorer.cmd`
 - `<INSTALL_DIR>/bin/strmqcfg.cmd`
- **strmqcfg** optional parameters:
 - c** Delete any cached data
 - i** Discard configuration information that IBM MQ uses
 - x** Write debug messages to the console

Introduction to IBM MQ Explorer

© Copyright IBM Corporation 2017

Figure 4-3. Starting IBM MQ Explorer

IBM MQ Explorer can be started from a command or from the system **Start** menu.

On both Windows and Linux, the command to start MQ Explorer is `strmqcfg`.

The `strmqcfg` command has three optional parameters to clear the cache and generate debug messages to the console.

On Linux, MQ Explorer can be started from the **System** menu that is based on Linux provider.

IBM Training

IBM

Welcome page

- View installations of IBM MQ on this computer and optionally transfer a queue manager to a different installation
- Connect to the IBM MQ web page
- Start the IBM Knowledge Center for IBM MQ
- Create the default configuration
- Verify the installation by using the Postcard application

© Copyright IBM Corporation 2017

Figure 4-4. Welcome page

The IBM MQ Explorer **Welcome** page is displayed the first time that MQ Explorer starts.

As shown in the figure, the **Welcome** page contains links for information about IBM MQ, for creating the default configuration, and verifying the installation.

The **Welcome** page can be accessed at any time by clicking **IBM MQ** in the **MQ Explorer - Navigator** view.

You can use the **Default Configuration** wizard to add the first configured queue manager to this system. You can use the Default Configuration wizard to create, view, or alter the default configuration. You can also use this wizard to alter or display details of an existing queue manager that was created in the default configuration.

If you migrated existing queue managers, or created any queue managers since installing MQ, you cannot run the **Default Configuration** wizard.

Start the **Default Configuration** wizard by selecting **Create the Default Configuration** on the **Welcome** page.

For a new installation of IBM MQ, you can create a default configuration to explore features of MQ by using the Postcard application and MQ Explorer. The Postcard application provides a fast and simple way to verify that your MQ installation was successful. It uses the default queue manager that is created in the **Default Configuration** wizard.

Welcome to MQ Explorer

In MQ Explorer, you can administer local and remote queue managers, and their resources, such as queues, channels, and listeners.

With this installation, existing local queue managers are displayed automatically in the Navigator view. To create a new local queue manager, right-click 'Queue Managers', select 'New', and then 'Queue Manager'. To add a remote queue manager, right-click 'Queue Managers', and select 'Add Remote Queue Manager'.

IBM MQ

- View the installations of IBM MQ on this machine and transfer queue managers to this installation.** **1**
- Find out more about IBM MQ on the Web.** **IBM MQ web page**
- Create the Default Configuration**

IBM MQ Installations

Installation of IBM MQ that this MQ Explorer is associated with:

Name	Version	Desc...	Path	Primary	State	Identifier
Installation2	9.0.0.0		C:\Cours...	No	Available	2

Other installations of IBM MQ on this machine:

Name	Version	Desc...	Path	Primary	State	Identifier
Installation1	8.0.0.0		C:\Progra...	Yes	Available	1

Transfer queue managers to this installation

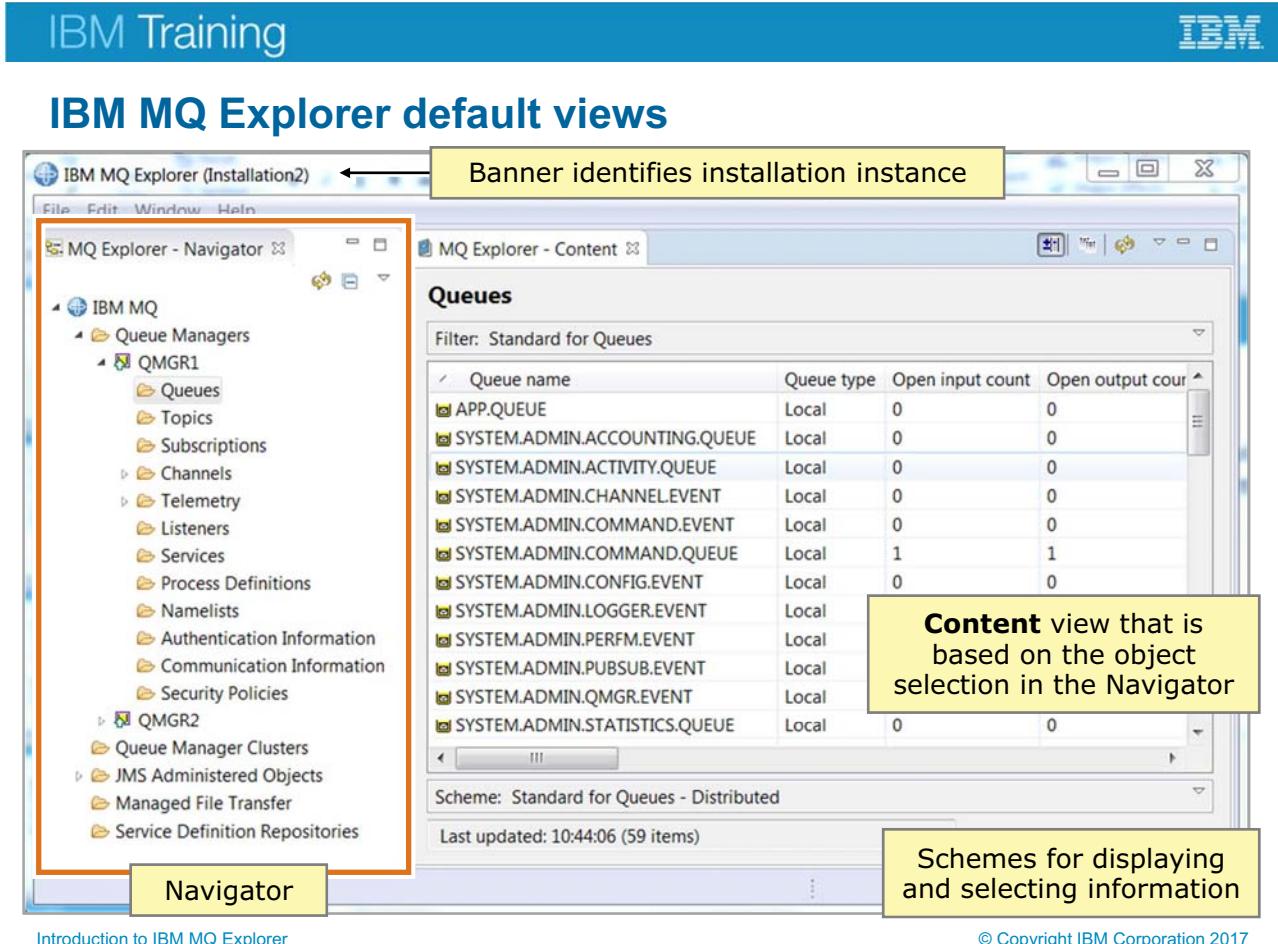
© Copyright IBM Corporation 2017

Figure 4-5. Viewing installations of IBM MQ

The Welcome page in MQ Explorer contains a link for viewing the MQ installations and optionally, transferring a queue manager to a different installation.

To view the MQ installations:

1. Click **View Installations** on the **Welcome** page.
2. The IBM MQ Installations page identifies the installation with which this MQ Explorer is associated. It also lists other installations of MQ on the same computer.



Introduction to IBM MQ Explorer

© Copyright IBM Corporation 2017

Figure 4-6. IBM MQ Explorer default views

The figure shows the general layout of the MQ Explorer user interface.

The **MQ Explorer - Navigator** view contains a list of all the objects that MQ Explorer manages. Status icons next to some of the objects, such as queue managers, indicate the status of the object. Each queue manager is listed under the **Queue Managers** folder. You can expand the queue manager to access the queue manager objects such as queues, topics, and subscriptions.

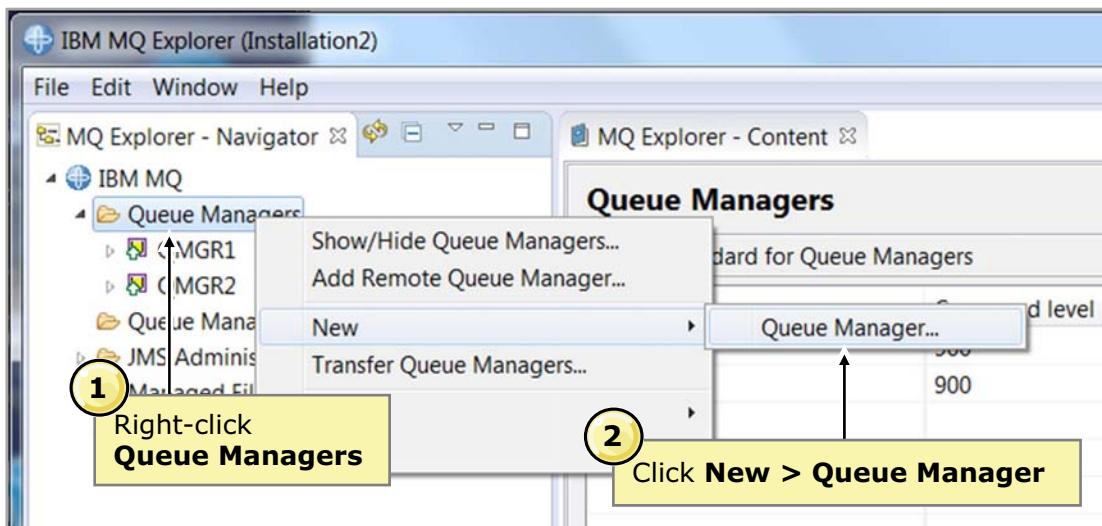
The **Content** views contain more information about the object that is selected in the **MQ Explorer - Navigator** view. For example, if the **Queues** folder under a queue manager is selected in the **MQ Explorer - Navigator** view, the **Content** view contains information about the queues on that queue manager.

You can use the **Scheme** icon (bottom portion of the **Queue** content view) to edit the current scheme and select the information to display in the property columns. You can also change the order of the columns.

The example in the figure shows the **Queues** content for the queue manager that is named QMGR1.



Creating a local queue manager (1 of 3)



Introduction to IBM MQ Explorer

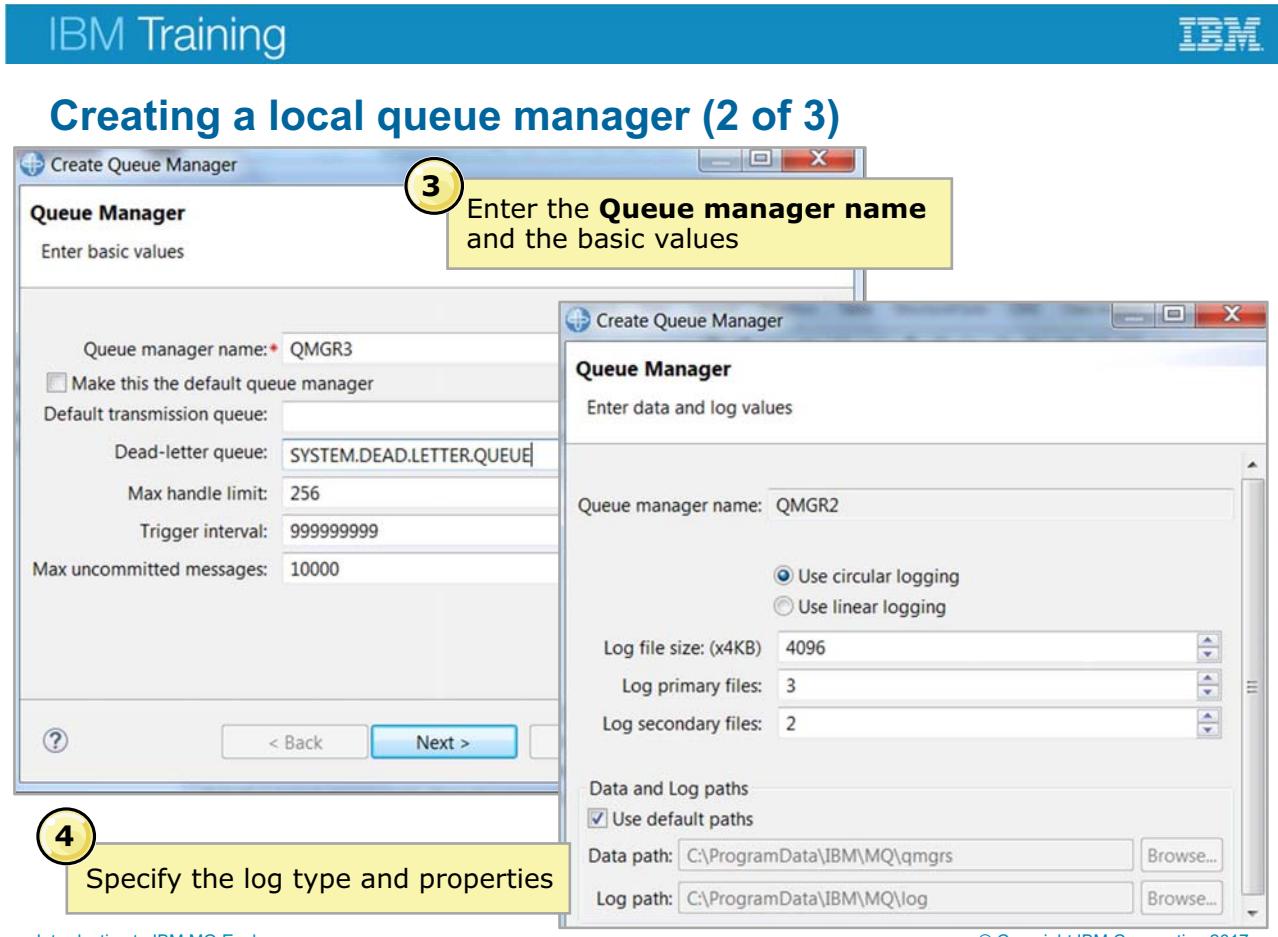
© Copyright IBM Corporation 2017

Figure 4-7. Creating a local queue manager (1 of 3)

You can create a local queue manager by using an MQ control command or by using MQ Explorer. MQ Explorer automatically shows all local queue managers regardless of the method that is used to create them.

The figure shows the first two steps for creating a local queue manager by using MQ Explorer.

1. Right-click **Queue Managers** in the **MQ Explorer - Navigator** view.
2. Click **New > Queue Manager**.



Introduction to IBM MQ Explorer

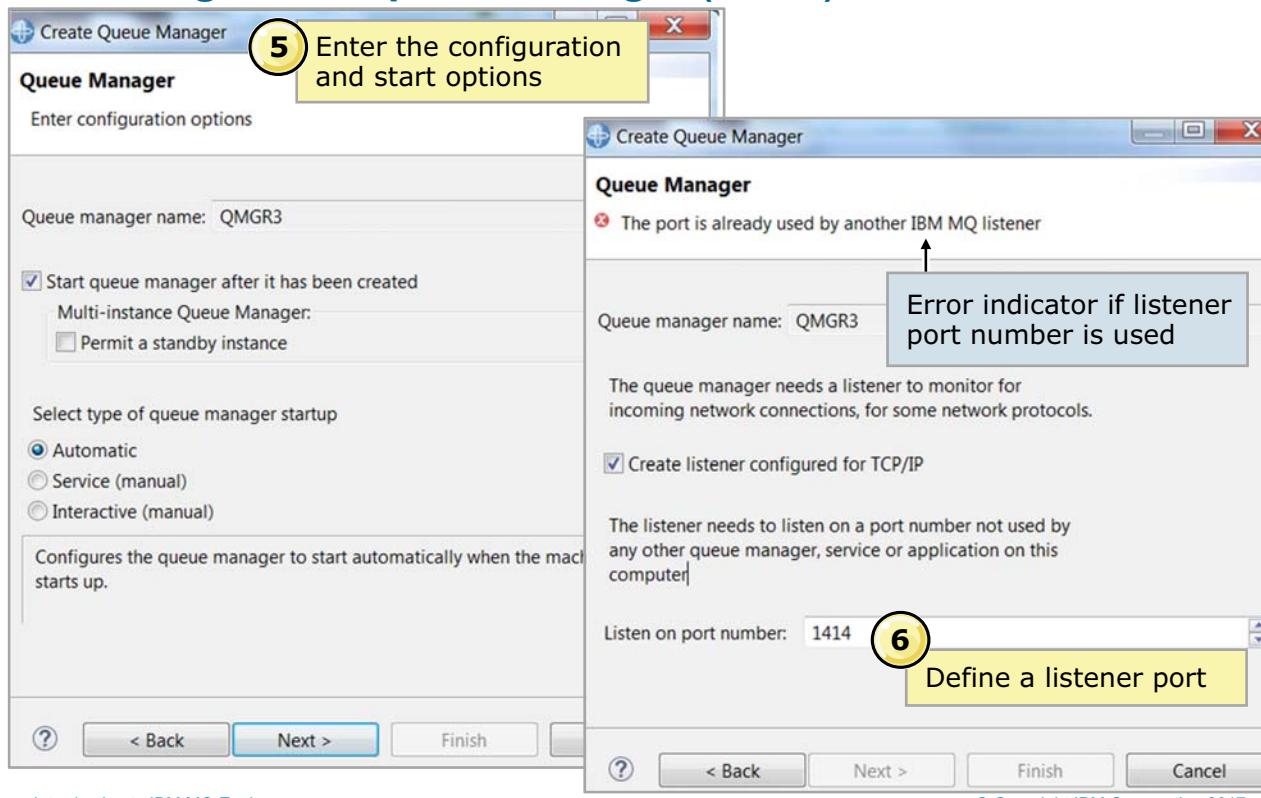
© Copyright IBM Corporation 2017

Figure 4-8. Creating a local queue manager (2 of 3)

3. Specify a queue manager name and other basic configuration. Basic configuration includes specifying a default transmission queue, dead-letter queue, and trigger interval.
4. Specify the queue manager log type and properties. Queue manager log files are described in more detail throughout this course.



Creating a local queue manager (3 of 3)



Introduction to IBM MQ Explorer

© Copyright IBM Corporation 2017

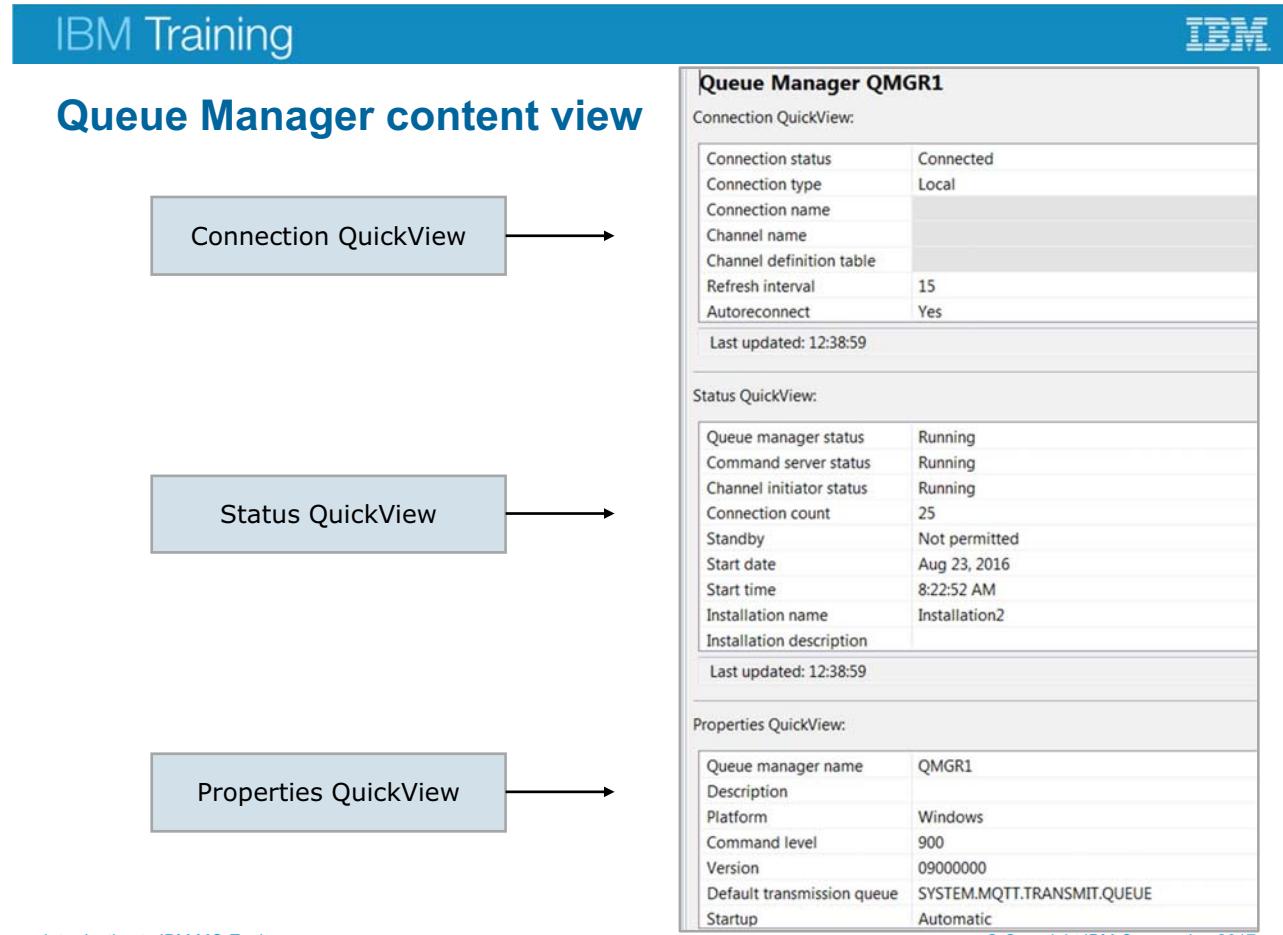
Figure 4-9. Creating a local queue manager (3 of 3)

5. Enter the queue manager configuration options.

- If you want the queue manager to start immediately after it is created, select **Start queue manager after it has been created**.
- If you want the queue manager to automatically start after a system restart, select **Automatic** for the startup type.

6. Create a listener for TCP/IP and identify the listener port value.

A unique port number is required for each listener on the same host. If MQ detects that another listener is using the TCP/IP port number, an error message is displayed at the top of the page, as shown in the example.



Introduction to IBM MQ Explorer

© Copyright IBM Corporation 2017

Figure 4-10. Queue Manager content view

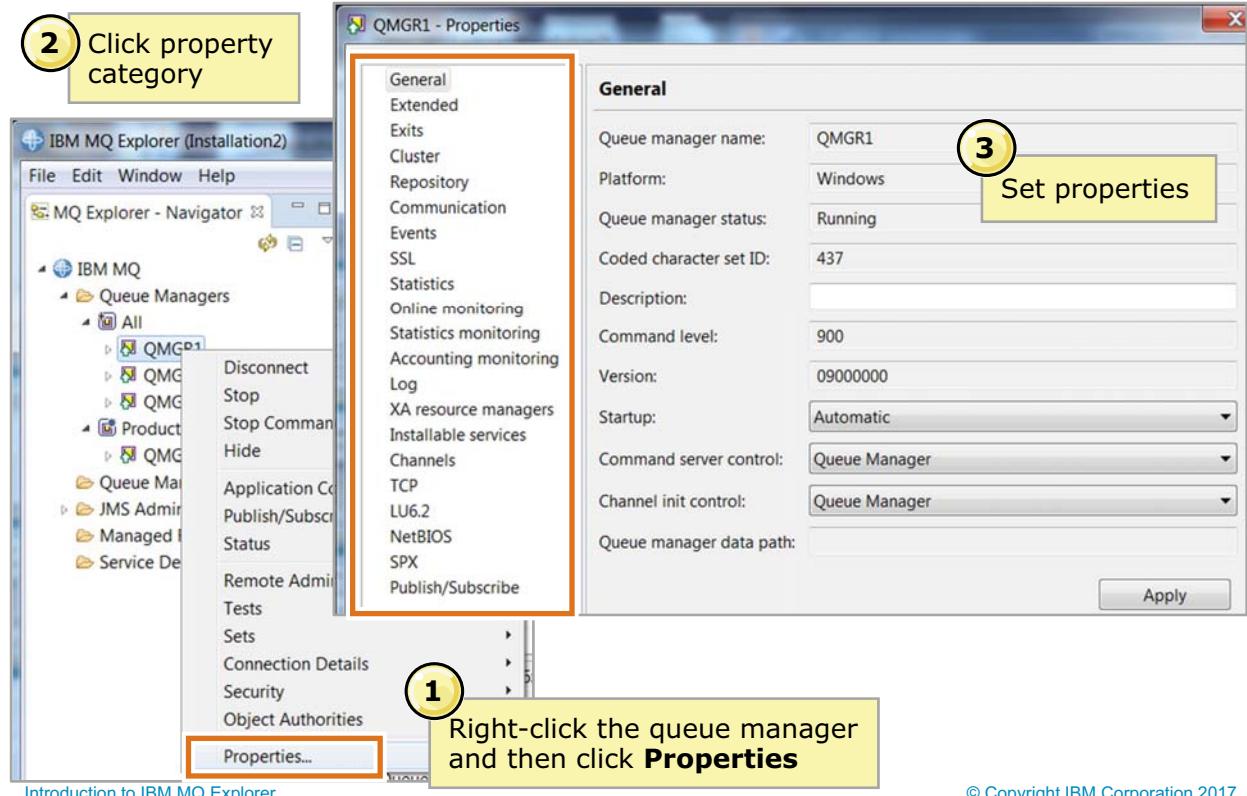
The **Queue Manager** content view is divided into three QuickView panes.

- **Connection QuickView** shows the queue manager connection status information, such as connection status, connection type, and connection name.
- **Status QuickView** shows the status of the queue manager. This QuickView includes the command server status, channel initiator status, and the installation name.
- **Properties QuickView** shows some of the configurable properties of the queue manager such as the operating system, command level (version of MQ), and default transmission queue.

The **Queue Manager** content view shows many of the properties that are returned with the `DIS QMGR` command.



Changing queue manager properties



© Copyright IBM Corporation 2017

Figure 4-11. Changing queue manager properties

You can view and optionally change queue manager properties in MQ Explorer. Queue manager properties are categorized by function.

To view and change the queue manager properties:

1. Right-click the queue manager in the **MQ Explorer - Navigator** view and then click **Properties**.
2. Click the property category.
3. Change the property value and then click **Apply**.

Queue manager property categories

- General
- Extended
- Exits
- Cluster
- Repository
- Communication
- Events
- SSL
- Statistics
- Online monitoring
- Statistics monitoring
- Accounting monitoring
- Log
- XA resource manager
- Installable services
- Channels
- TCP
- LU6.2
- NetBIOS
- SPX
- Publish/subscribe

Introduction to IBM MQ Explorer

© Copyright IBM Corporation 2017

Figure 4-12. Queue manager property categories

This figure lists the queue manager property categories.

Grouping queue managers

- Queue managers can be grouped into sets
- Actions can be completed on a set of queue managers:
 - Show or hide all
 - Connect or disconnect all
 - Start or stop all local
 - Run default or custom tests
- Grouping can be done:
 - Manually
 - Automatically
- Automatic grouping by filtering on:
 - Command level
 - Operating system
 - Any queue manager attribute by creating a custom filter
- Queue managers can be members of none, one, or many sets
- Sets cannot contain other sets

Introduction to IBM MQ Explorer

© Copyright IBM Corporation 2017

Figure 4-13. Grouping queue managers

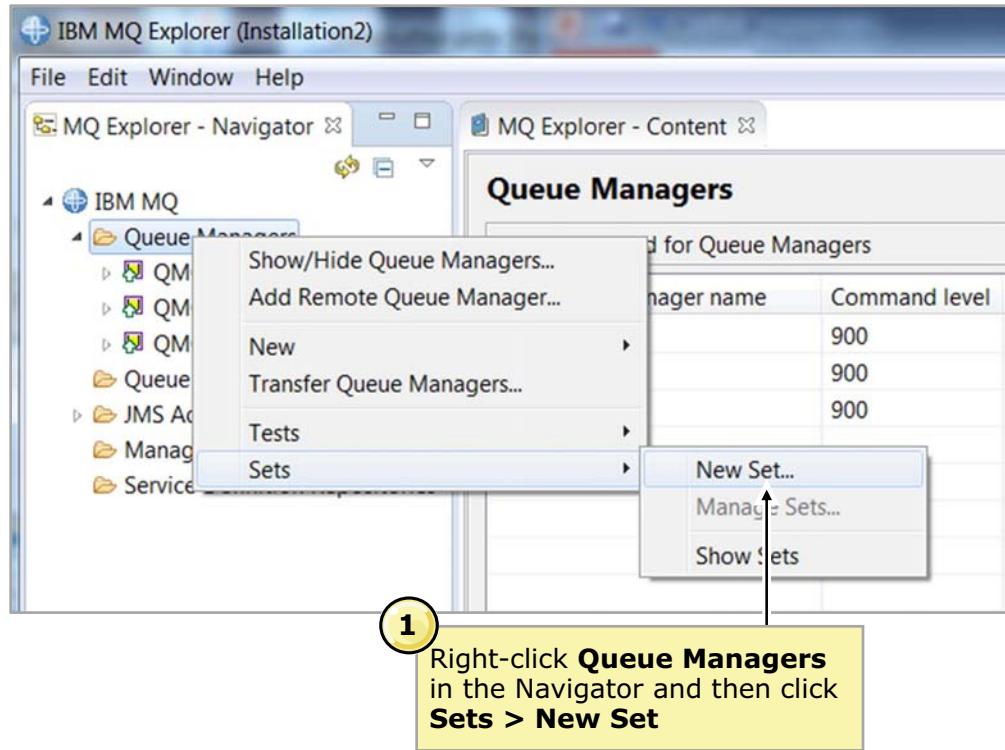
You can group queue managers into sets.

Grouping queue managers into sets makes it more efficient to complete operations on a select set of queue managers at the same time rather than one at a time. For example, you can stop and start all queue managers in a group at the same time.

You can group queue managers manually by selecting the queue managers. You can also group queue managers automatically by defining filter conditions. For example, you can define a group that includes running queue managers, or queue managers that use the same dead-letter queue.



Creating a queue manager set (1 of 7)



Introduction to IBM MQ Explorer

© Copyright IBM Corporation 2017

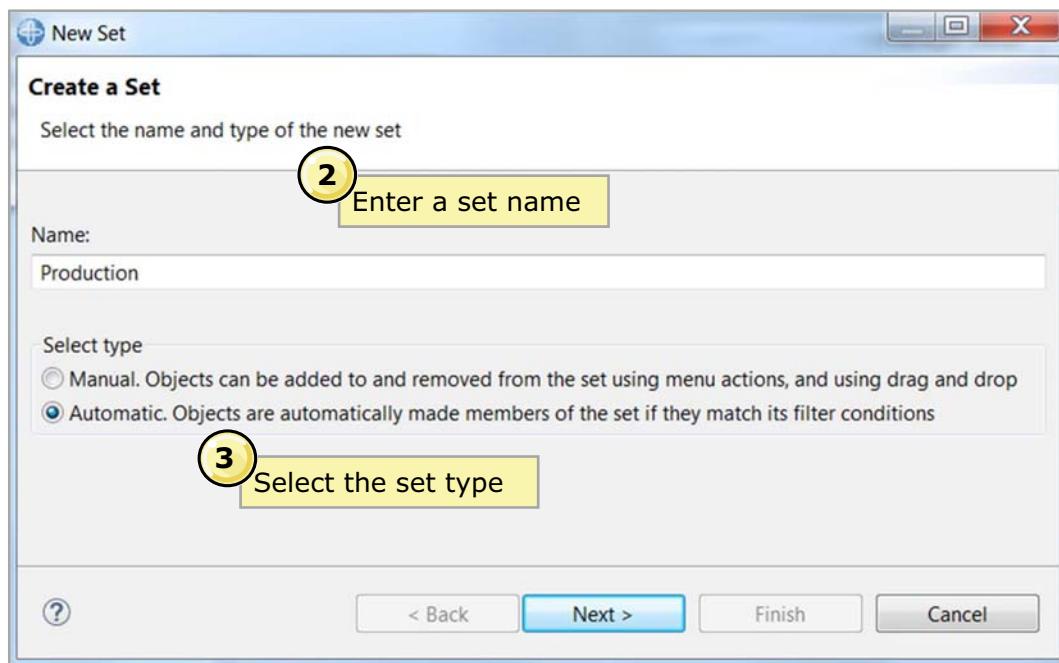
Figure 4-14. Creating a queue manager set (1 of 7)

This example shows you how to create an automatic queue manager that uses a custom filter to determine members of the group. In this example, the queue manager is automatically added to the group if the queue manager **Description** attribute is set to **Production**.

1. Right-click **Queue Managers** in the **MQ Explorer - Navigator** view and then click **Sets > New Set**.



Creating a queue manager set (2 of 7)



Introduction to IBM MQ Explorer

© Copyright IBM Corporation 2017

Figure 4-15. Creating a queue manager set (2 of 7)

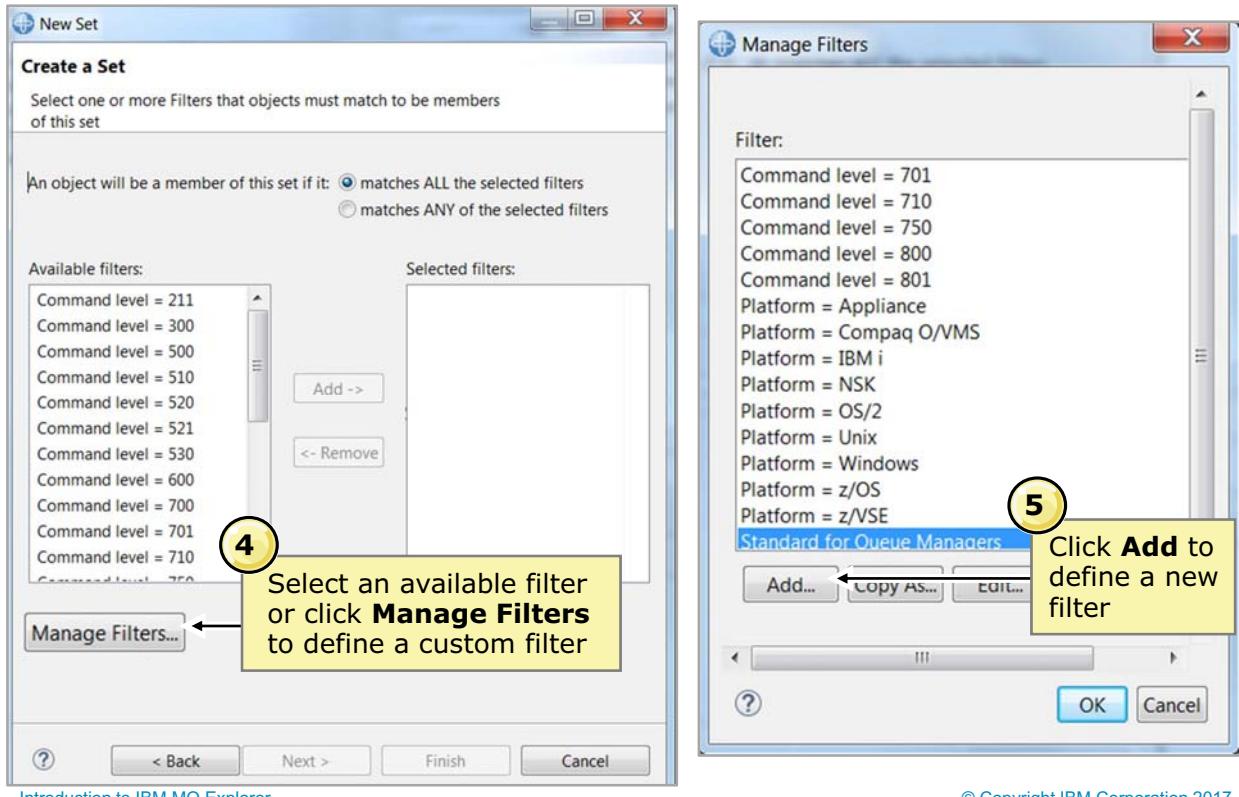
2. Enter a descriptive set name.

For example, create a set that is named **Production** that includes all queue managers that are identified as queue managers that are used in a production environment.

3. If you want to select the queue managers to add to the set by name, select **Manual**. If you want to use a filter to determine the queue managers that are added to the set, select **Automatic**. The example creates an automatic set that uses a custom filter.



Creating a queue manager set (3 of 7)



Introduction to IBM MQ Explorer

© Copyright IBM Corporation 2017

Figure 4-16. Creating a queue manager set (3 of 7)

If you selected **Manual** as the set type, the final step for creating a queue manager set is to select the queue managers that are members of the set.

If you selected **Automatic** for the queue manager set type, the next steps are to:

4. Select the filter that automatically adds queue managers. For example, you can create a queue manager set that is named "Windows Queue Managers". You can then select the filter condition of **Platform = Windows** so that any queue managers that are running on Windows are automatically added to the set.

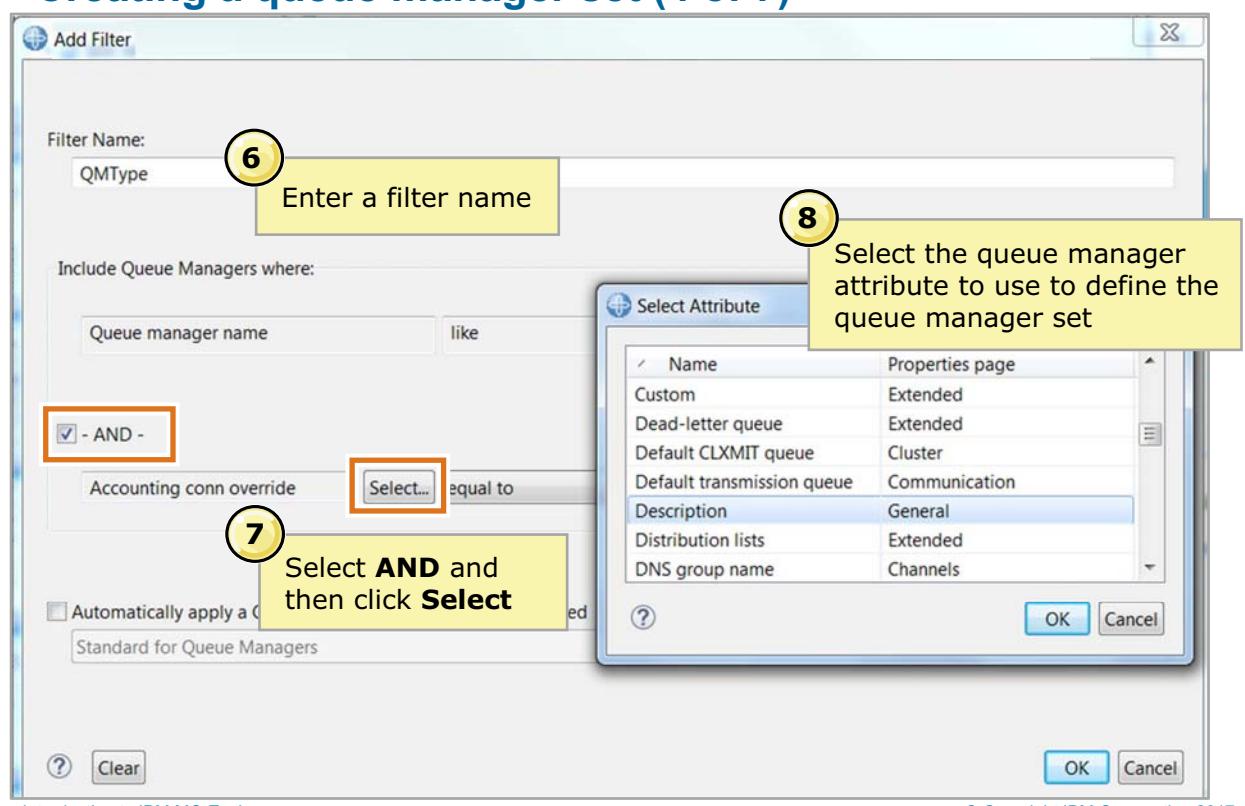
As an option, you can click **Manage Filters** to define a custom filter. The example in this figure and the next set of figures show the steps for defining a custom filter.

5. Click **Add** on the **Manage Filters** window to define a new filter.

IBM Training



Creating a queue manager set (4 of 7)



Introduction to IBM MQ Explorer © Copyright IBM Corporation 2017

Figure 4-17. Creating a queue manager set (4 of 7)

This figure shows the next steps for creating a custom queue manager set filter for all queue managers.

6. Enter a unique descriptive filter name.
7. Select the **AND** check box and then click **Select** to continue to define the custom filter expression.
8. Select the queue manager attribute in the **Select Attribute** window and then click **OK**.

IBM Training



Creating a queue manager set (5 of 7)

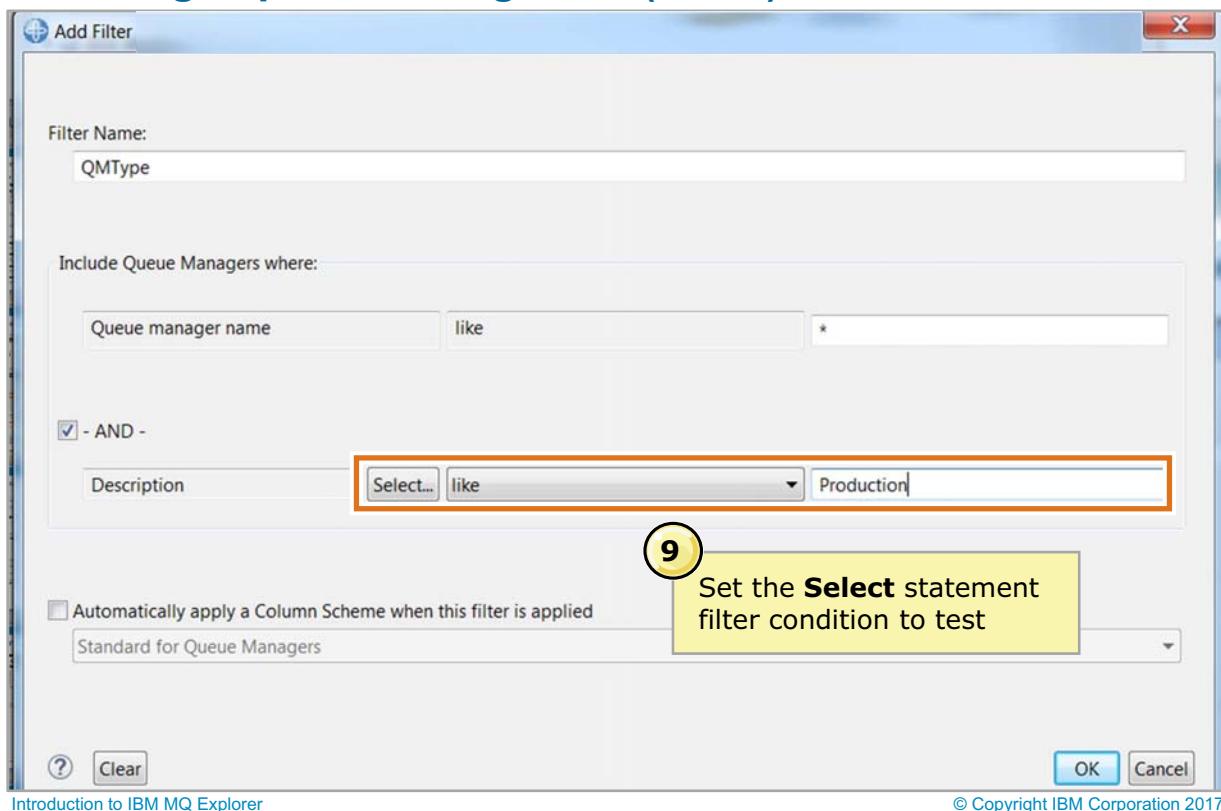


Figure 4-18. Creating a queue manager set (5 of 7)

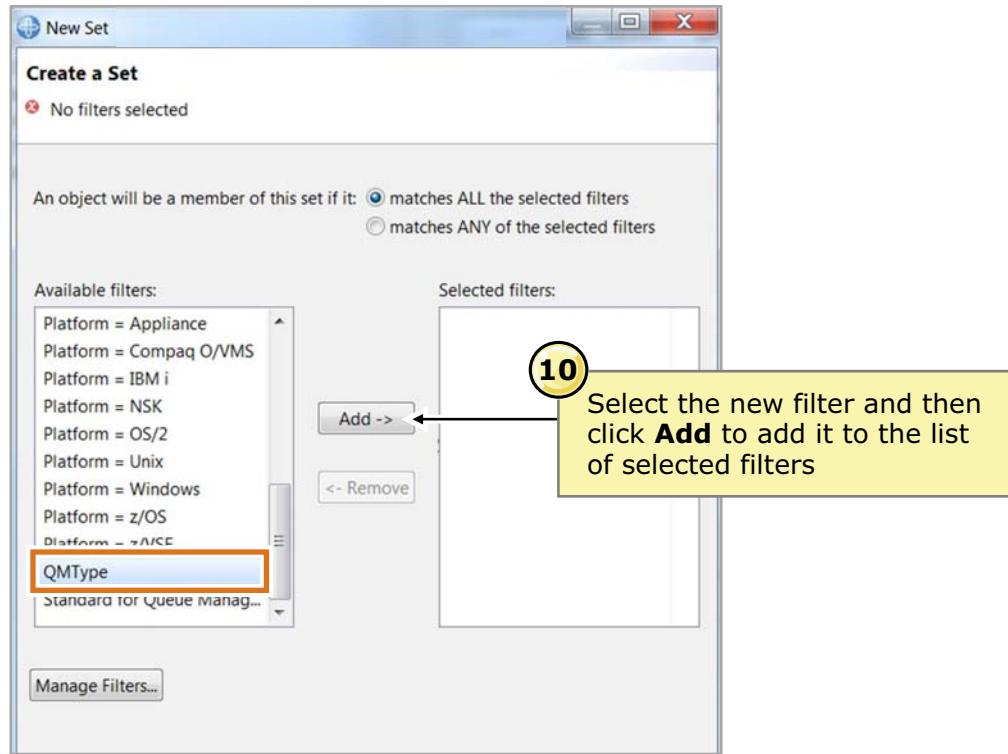
This figure shows the next step for defining a custom filter expression.

9. Select the expression condition such as **equal to** or **like** and then enter the string to evaluate.

In this example, the queue manager is automatically added to the group if the queue manager **Description** attribute is set to **Production**.



Creating a queue manager set (6 of 7)



Introduction to IBM MQ Explorer

© Copyright IBM Corporation 2017

Figure 4-19. Creating a queue manager set (6 of 7)

This figure shows the final step for defining an automatic queue manager set.

10. Select the filter in the **Available filters** pane and then click **Add** to move it to the **Selected filters** pane.

If you add more than one filter, be sure to verify the options for whether an object must match ALL selected filters or ANY of the selected filters.



Creating a queue manager set (7 of 7)

The queue manager set folder is added to the **Queue Managers** folder and contains all queue managers that match the filter criteria

MQ Explorer - Navigator

- IBM MQ
 - Queue Managers
 - All
 - QMGR1
 - QMGR2
 - QMGR3
 - Production
 - QMGR3
- JMS Administered Objects
- Managed File Transfer
- Service Definition Repositories

MQ Explorer - Content

Queue Manager QMGR3

Connection QuickView:

Connection status	Connected
Connection type	Local
Connection name	

Last updated: 13:11:49

Status QuickView:

Queue manager status	Running
Command server status	Running
Channel initiator status	Running
Connection count	23
Standby	Not permitted

Last updated: 13:11:49

Properties QuickView:

Queue manager name	QMGR3
Description	Production
Platform	WINDOWS

Last updated: 13:11:49

Introduction to IBM MQ Explorer

© Copyright IBM Corporation 2017

Figure 4-20. Creating a queue manager set (7 of 7)

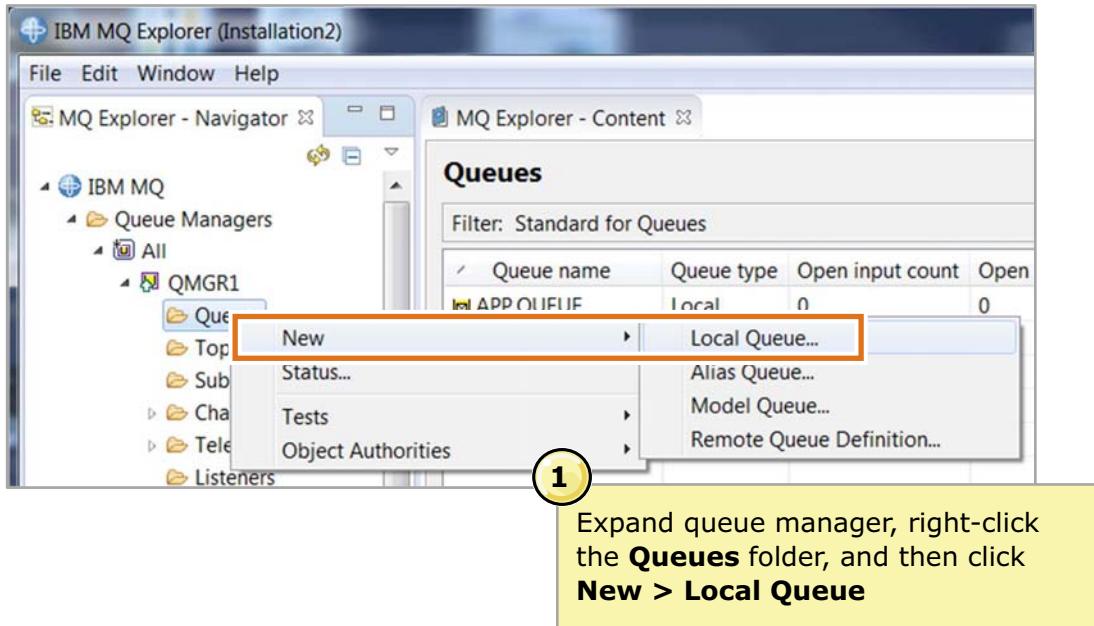
After you define a queue manager set, the **MQ Explorer- Navigator** view shows the new queue manager set and another queue manager set that is named **All**. The **All** set contains all queue managers that MQ Explorer can manage.

In this example, the queue manager that is named QMGR3 is automatically added to the **Production** set because the queue manager **Description** property is set to **Production**. You can create another queue manager set that is named **Development** and define another custom filter where **Description** is equal to **Development** to further separate and manage the queue managers by use.

You create a queue manager set in the exercise for this unit.



Creating a local queue (1 of 3)



Introduction to IBM MQ Explorer

© Copyright IBM Corporation 2017

Figure 4-21. Creating a local queue (1 of 3)

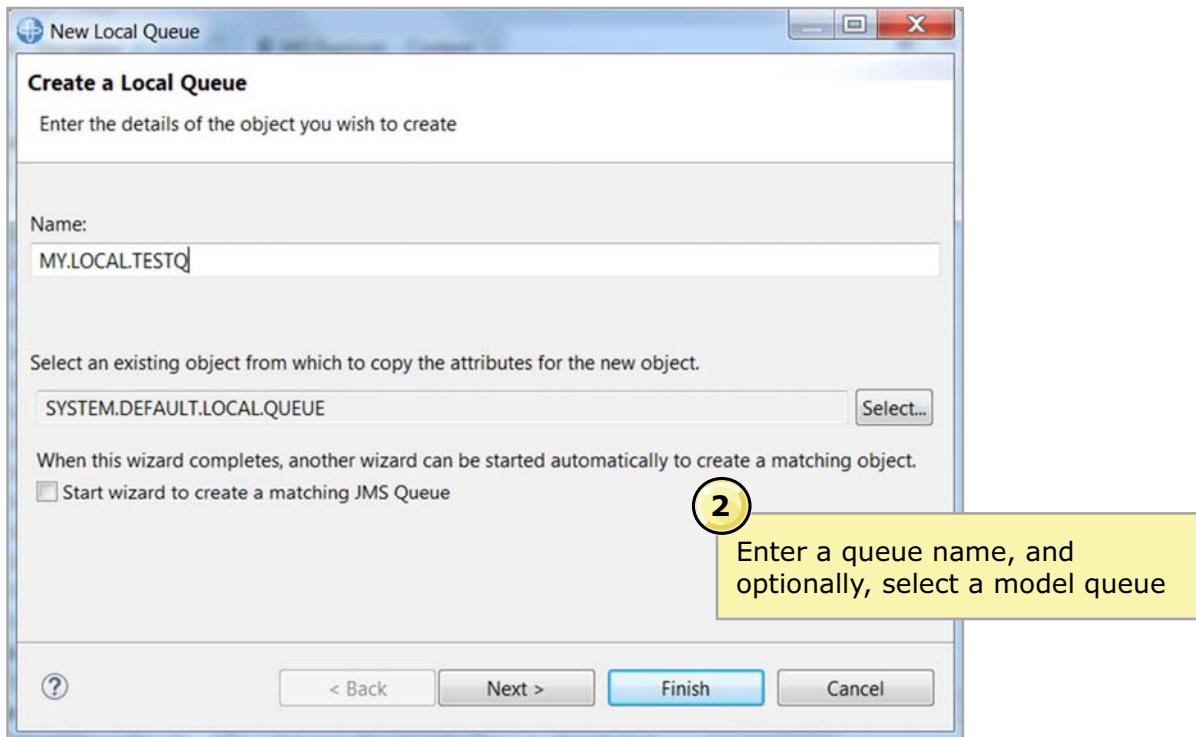
It is possible to define local, alias, model, and local definitions of remote queues for any queue manager that is connected to MQ Explorer.

To create a new local queue in MQ Explorer:

1. Right-click the **Queues** folder under the queue manager in the **MQ Explorer - Navigator** and then click **New > Local Queue**.

IBM Training

Creating a local queue (2 of 3)



Introduction to IBM MQ Explorer

© Copyright IBM Corporation 2017

Figure 4-22. Creating a local queue (2 of 3)

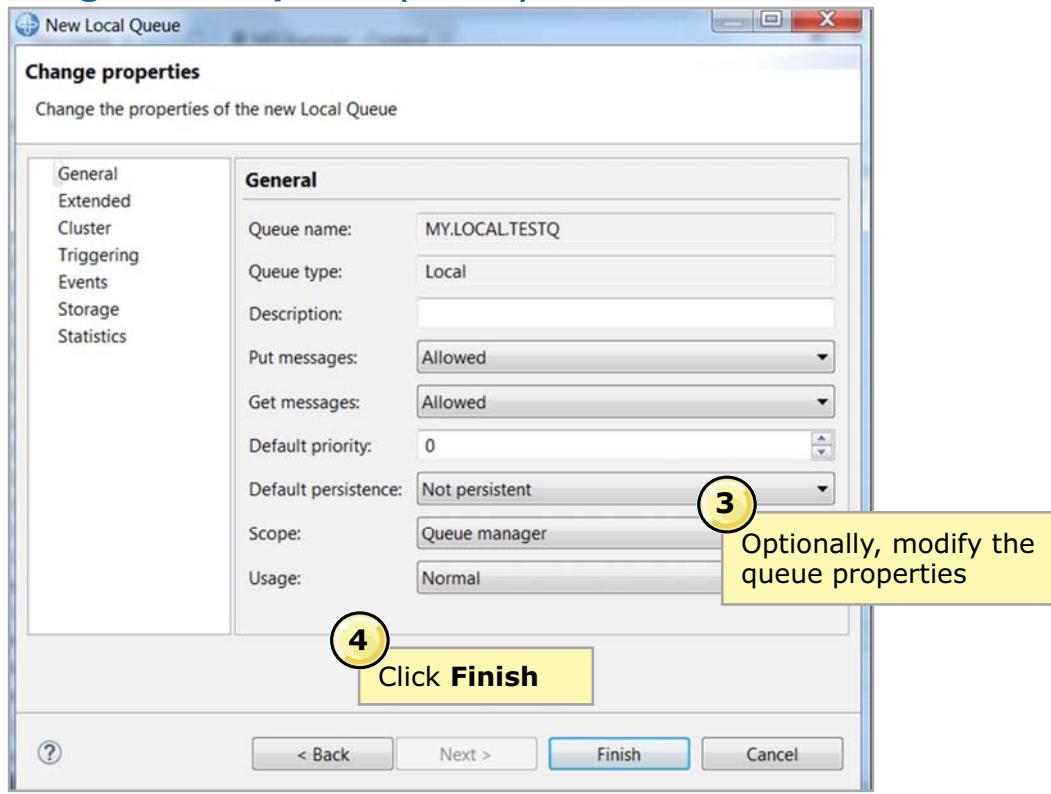
2. Enter a unique queue name.

By default, MQ uses a default SYSTEM queue as the model for the new queue. If you want, you can specify a different queue to use as the model for this queue.

An option also exists to automatically start another wizard to create a matching JMS queue.



Creating a local queue (3 of 3)



Introduction to IBM MQ Explorer

© Copyright IBM Corporation 2017

Figure 4-23. Creating a local queue (3 of 3)

This figure shows the next steps for creating a local queue.

3. Modify the queue properties, if necessary. The queue properties are categorized by function:
 - General
 - Extended
 - Cluster
 - Triggering
 - Events
 - Storage
 - Statistics
4. Click **Finish**.

Queues

Filter: Standard for Queues

Queue name	Queue type	Open input count	Open output count	Current queue depth	Put messages	Get messages	Remote queue
APP.QUEUE		0	0	0	Allowed	Allowed	
MY.LOCAL.TEST		0	0	0	Allowed	Allowed	

Scheme: Standard for Queues - Distributed

Last updated: 14:05:40 (2 items)

Double-click a queue to open the **Properties** view, or right-click a queue for more actions

Introduction to IBM MQ Explorer

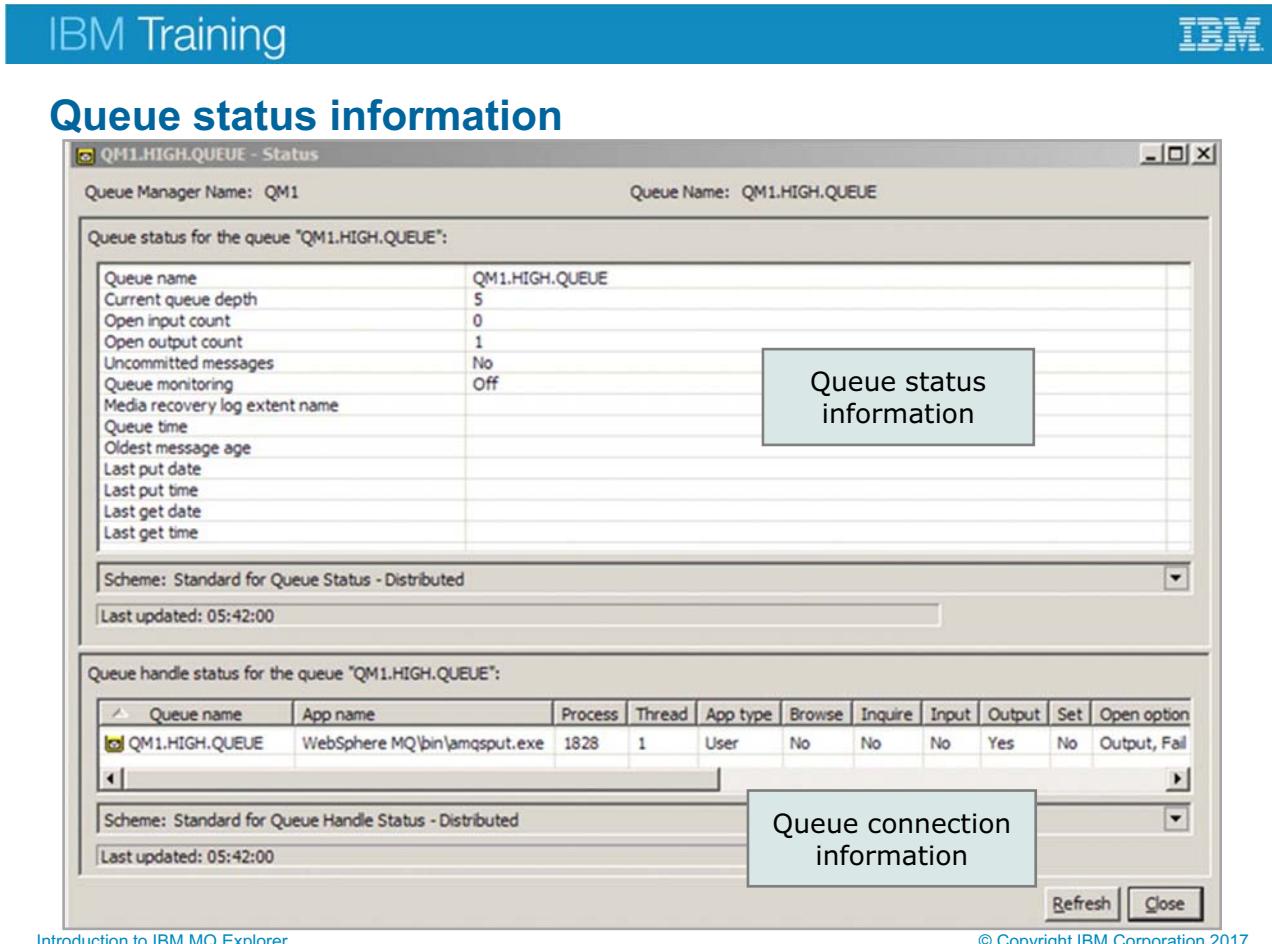
© Copyright IBM Corporation 2017

Figure 4-24. Queues content view

The **Queues** content view is used to view, clear, put messages, and obtain status information for the selected queue. It is also used to define object authorities on the queue that identify groups and users that can put and get messages.

You can modify queue properties by double-clicking a queue or right-clicking a queue and then clicking **Properties**.

Right-click the queue for actions such as clearing a queue, putting test messages, getting messages, and browsing messages.



Introduction to IBM MQ Explorer

© Copyright IBM Corporation 2017

Figure 4-25. Queue status information

The queue status view can help to determine real-time queue usage information. This information can assist application programmers with diagnostic information.

Obtain status information by right-clicking the queue in the **Queues** content view and then clicking **Status**.

Status information shows many of the same attributes that are returned by the **DISPLAY** command:

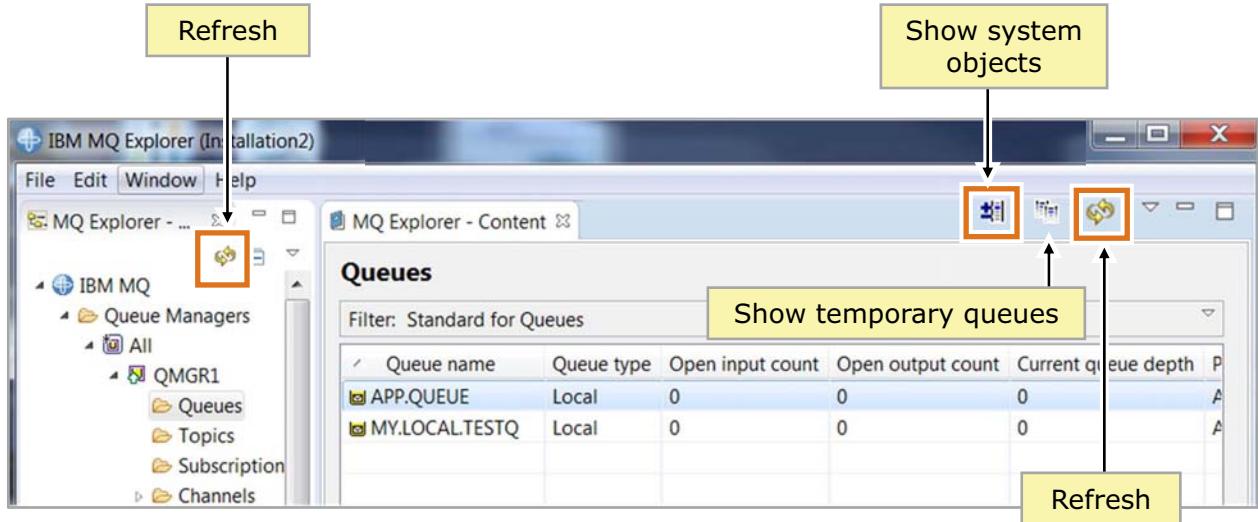
- Number of uncommitted messages
- Last put and get times
- Queue connection information
- Application usage information

The **Queue handle status** view, which is shown in the figure, contains the active applications that are using the queue. It contains the process ID, number of threads, and open options (**MQGET** or **MQPUT**). It also shows what open options are requested and the user ID that the process is running under. The queue connection information can be instrumental in assisting application testing.

The **Queue handle status** example in the figure shows that an **amqsput** application with process number 1828 is using one thread.



Shortcut icons



Introduction to IBM MQ Explorer

© Copyright IBM Corporation 2017

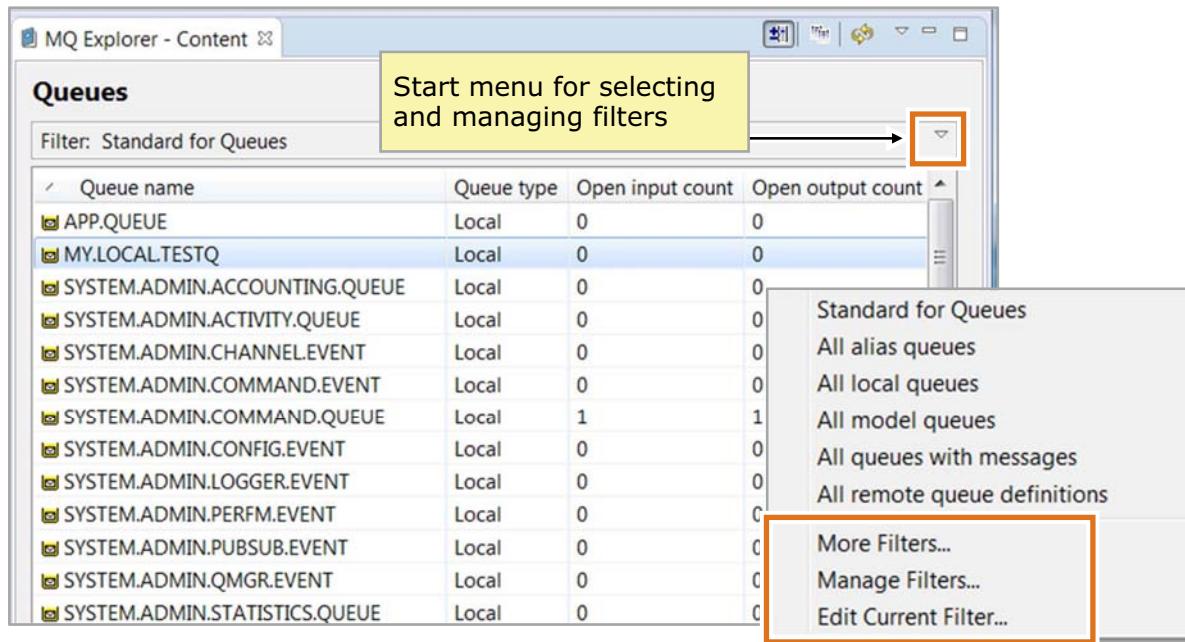
Figure 4-26. Shortcut icons

As shown in the figure, MQ Explorer has shortcut icons for refreshing lists and views and showing system objects.

More shortcut icons are available based on the selected content. For example, when the content area contains the list of queues, a shortcut icon is available for showing temporary queues and SYSTEM queues.

IBM Training

Viewing queue filters



Introduction to IBM MQ Explorer

© Copyright IBM Corporation 2017

Figure 4-27. Viewing queue filters

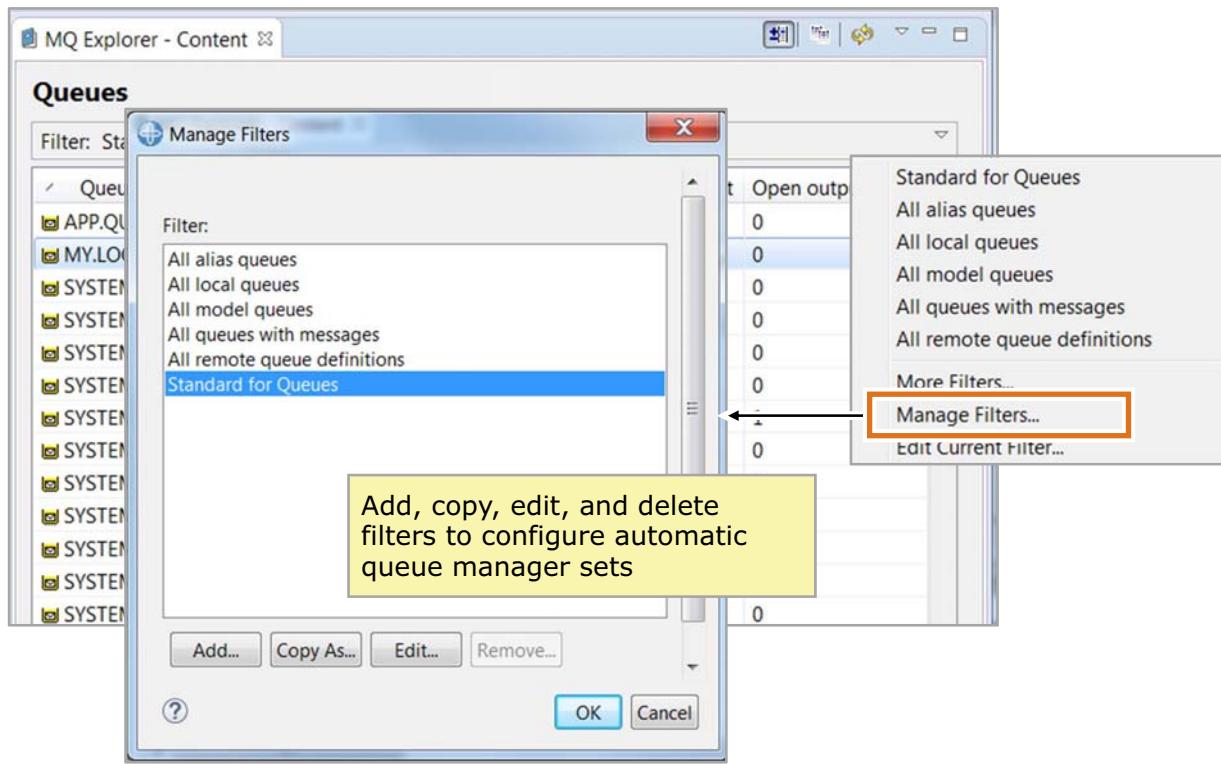
Filters provide a way to selectively view queues that meet certain criteria such as queues that contain messages. A filter allows the display of queues that meet the filter criteria and hides queues that do not conform to the selection criteria.

Click the **Filter** menu icon to access the options for selecting and managing the filters.

The **Show system objects** icon is a toggle. When active, it shows the system queues for the queue manager.

IBM Training

Managing queue filters



Introduction to IBM MQ Explorer

© Copyright IBM Corporation 2017

Figure 4-28. Managing queue filters

By using the **Manage Filters** option, you can create, modify, and delete queue display filters.

In the figure, the list of filters includes a filter to show all alias queues, all local queues, all model queues, all queues with messages, and all remote queues.

In the **Manage Filters** window, you can:

- Click **Add** to create a filter.
- Click **Copy As** to copy a filter.
- Click **Edit** to edit filters to add, remove, or change the criteria that are set for the filter.
- Click **Remove** to delete a filter.

IBM Training

Adding queue filters

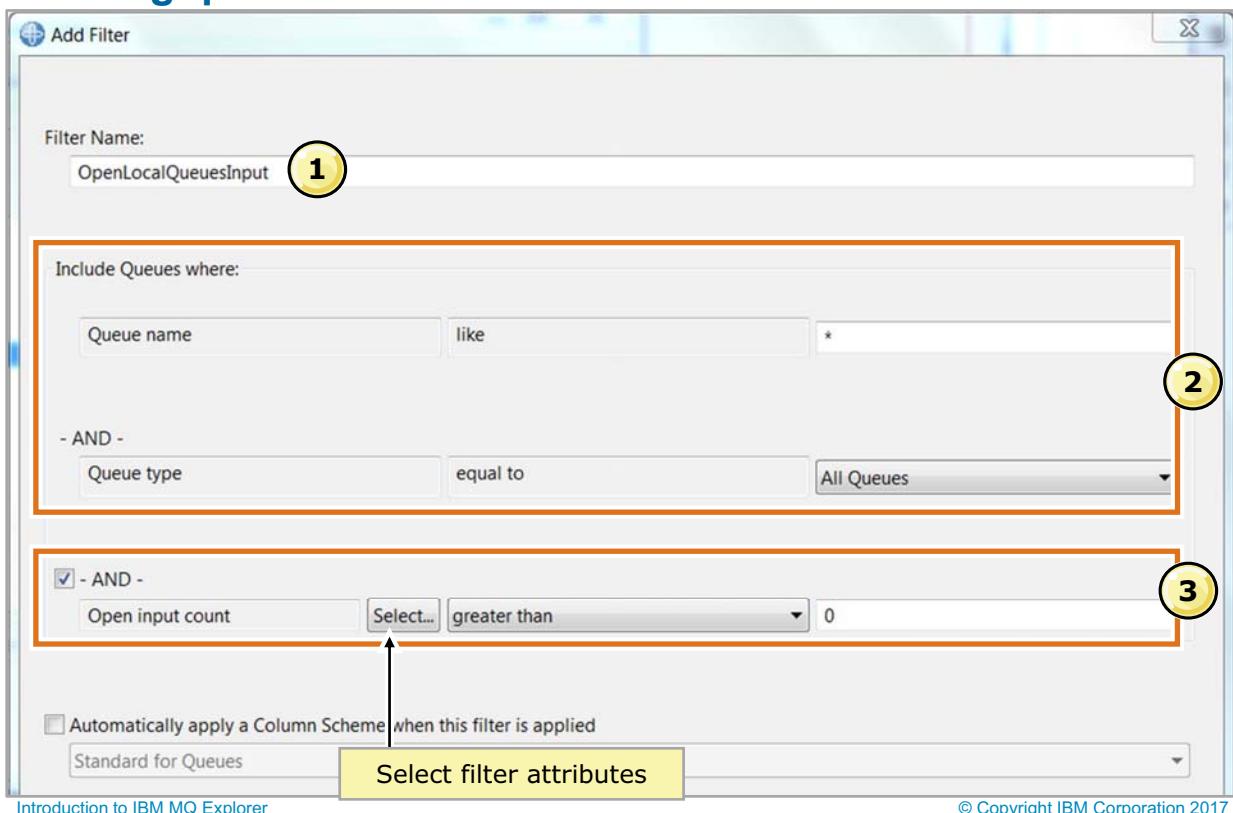


Figure 4-29. Adding queue filters

Using advanced filtering, you can customize the views of information gathering.

To add a queue display filter in MQ Explorer, click **Add** on the **Manage filters** window. The figure shows the next steps for defining a custom filter.

1. Enter a descriptive filter name.
2. Select the filter conditions.
3. Optionally, add a condition by selecting **AND** and the filter attributes. Click **Select** to select the filter attribute from a list of valid attributes.

In the example, a filter that is named **OpenLocalQueuesInput** filters all queues. The filter expression displays the queues where the open input count for the queue is not equal to zero.

IBM Training

Queue filter attributes

Select Attribute	
✓ Name	Properties page
Archive	Storage
Automatic	Reorganization
Backout requeue queue	Storage
Backout threshold	Storage
Base object	General
Base type	General
Channel name	Triggering
CICS file name	Extended
Cluster channels	Extended
Cluster name	Cluster
Cluster name list	Cluster
Cluster workload use queue	Cluster
CLWL queue priority	Cluster
CLWL queue rank	Cluster
Coupling facility name	Storage
Current queue depth	Statistics
Custom	Extended
Default bind type	Cluster
Default input open option	Extended
Default persistence	General
Default priority	General
Default put response type	Extended
Default read ahead	Extended
Definition type	Extended
Description	General
Distribution lists	Extended
Get messages	General

Select Attribute	
✓ Name	Properties page
Harden get backout	Storage
Index type	Extended
Initiation queue	Triggering
Interval	Reorganization
Maximum global locks	Extended
Maximum local locks	Extended
Maximum single queue acce...	Extended
Maximum starts	Triggering
Max message length	Extended
Max queue depth	Extended
Message delivery sequence	Extended
NPM class	Storage
Open input count	Statistics
Open output count	Statistics
Page set ID	Extended
Pipe name	Extended
Process name	Triggering
Program id	Triggering
Put messages	General
Queue accounting	Statistics
Queue depth high events	Events
Queue depth high limit	Events
Queue depth low events	Events
Queue depth low limit	Events
Queue depth max events	Events
Queue monitoring	Statistics
Queue service interval	Events

Select Attribute	
✓ Name	Properties page
Queue service interval events	Events
Queue statistics	Statistics
Remote queue	General
Remote queue manager	General
Restart	Triggering
Retention interval	Extended
Scope	General
Shareability	Extended
Start time	Reorganization
Storage class name	Storage
Terminal id	Triggering
Transaction id	Triggering
Transmission queue	General
Trigger control	Triggering
Trigger data	Triggering
Trigger depth	Triggering
Trigger message priority	Triggering
Trigger type	Triggering
Usage	General
VSAM catalog	Reorganization

Introduction to IBM MQ Explorer

© Copyright IBM Corporation 2017

Figure 4-30. Queue filter attributes

The figure shows the filter attributes that are available when you specify an extra **AND** condition on the queue filter expression. Examples of filter attributes are cluster name, current queue depth, maximum queue depth, and process name.

The filter attributes are displayed when you click **Select** in the **Add filter** view.



Comparing queues (1 of 2)

- Can compare two like resources
- Can compare resources across queue managers

Queue name	Queue type	Open input count	Open output count	Current queue depth	Put messages
APP.QUEUE		0	0	0	Allowed
MY.LOCAL.TEST		0	0	0	Allowed

Introduction to IBM MQ Explorer

© Copyright IBM Corporation 2017

Figure 4-31. Comparing queues (1 of 2)

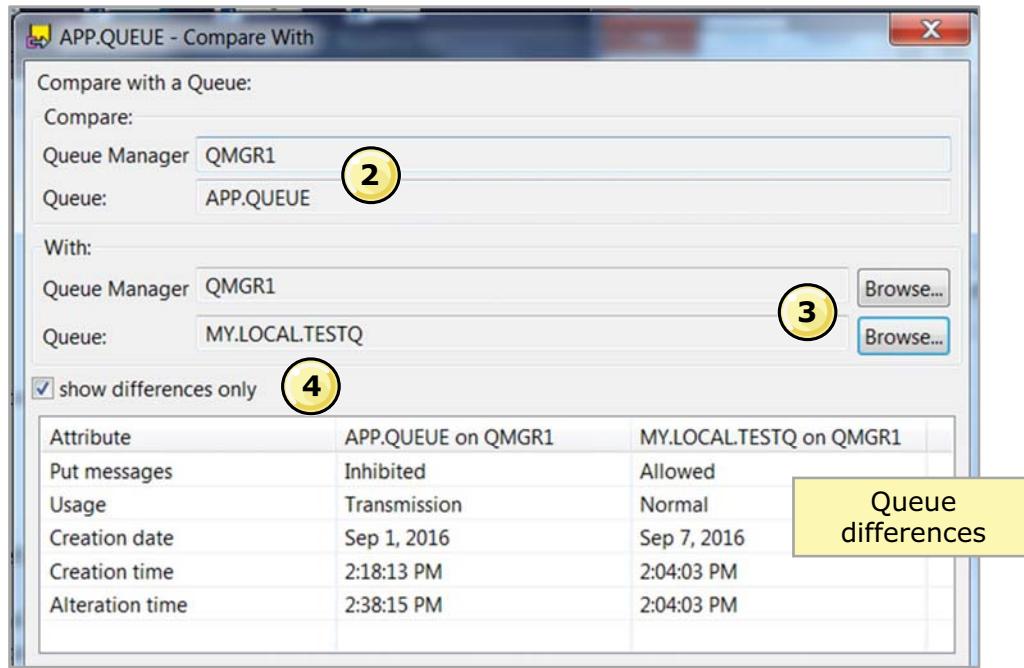
Using MQ Explorer, you can compare queues. The queue compare functions can be used as a migration aid to verify queue attributes after they are moved into production, for example.

To compare two queues, complete the following steps:

1. In the **Queues** view, right-click the first queue and then click **Compare with**.

IBM Training

Comparing queues (2 of 2)



Introduction to IBM MQ Explorer

© Copyright IBM Corporation 2017

Figure 4-32. Comparing queues (2 of 2)

2. Select the queue manager for the second queue.
3. Select the queue in the **With** field.
4. Optionally, select **show differences only** to display the queue properties that are different.

If you do not select **show differences only**, the list shows all the properties for both queues.

The example in the figure compares two queues from the same queue manager.

This example is previewing queue attributes that are different.

IBM MQ object definition tests

- General tests
 - Queue manager names
 - Dead-letter queue definitions
 - FFST error log
 - Stopped queue managers
 - Verify default transmission queue
- Queue tests
 - Identify full queues
 - Verify alias queue definitions
 - Verify queue names
 - Verify that queues are get-enabled
 - Verify that queues are put-enabled
 - Verify remote queue definitions
 - Verify use of transmission queue in queues

Introduction to IBM MQ Explorer

© Copyright IBM Corporation 2017

Figure 4-33. IBM MQ object definition tests

You can use MQ Explorer to test the general functions of MQ, test queues, and test queue connectivity.

This figure summarizes the MQ Explorer object definition tests.

IBM MQ general tests

- Queue manager names
 - Looks for similar names
 - Displays warnings for queue managers that are hosted on different servers but with identical names
- Dead-letter queue definitions
 - Warning for any queue manager that does not have a dead-letter queue
 - Error for any queue manager with invalid dead-letter queue attributes
- If any FFST logs were written on the local system, FFST error log displays an error
- Displays a warning for each queue manager that is stopped
- Displays errors for any invalid uses of the **Default Transmission Queue** attribute

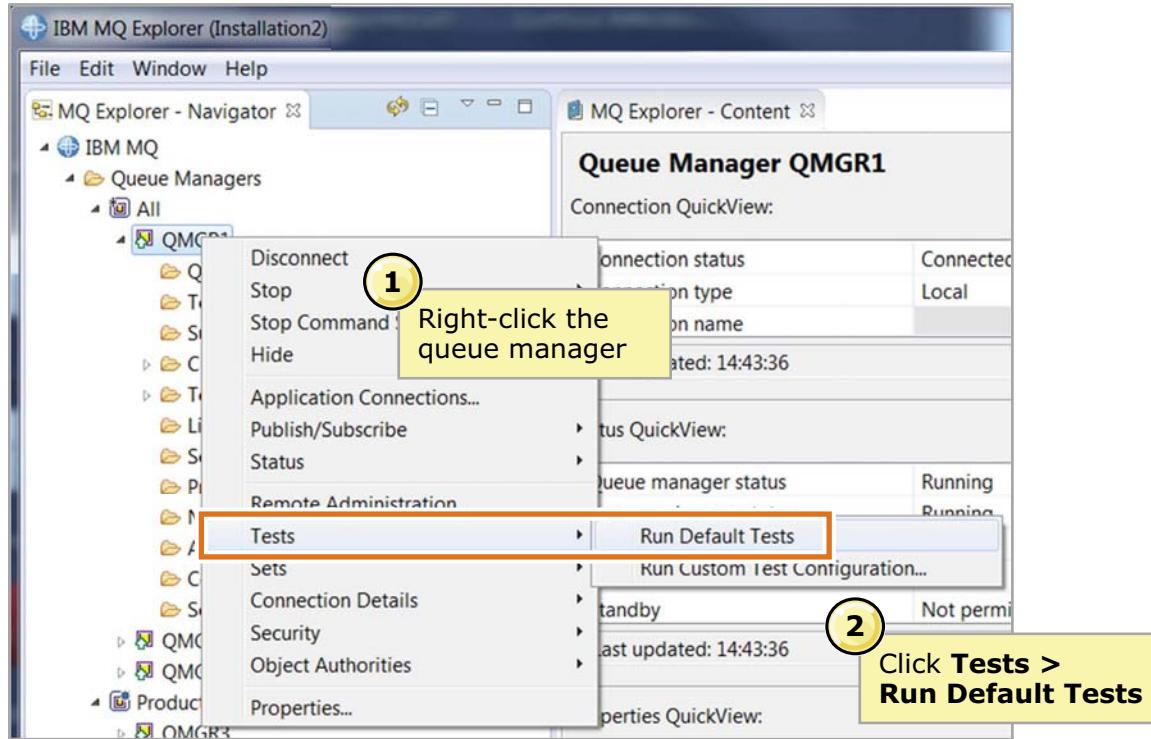
Introduction to IBM MQ Explorer

© Copyright IBM Corporation 2017

Figure 4-34. IBM MQ general tests

The MQ Explorer tests provide some basic information about queue manager names, dead-letter queue definitions, the First Failure Support Technology (FFST) error log, stopped queue managers, and default transmission queues.

Running the IBM MQ tests



Introduction to IBM MQ Explorer

© Copyright IBM Corporation 2017

Figure 4-35. Running the IBM MQ tests

To run the MQ tests in MQ Explorer:

1. Right-click a queue manager in the **MQ Explorer - Navigator** view.
2. Click **Tests > Run Default Tests**.



IBM MQ test results example

Description	Object name	Category
✖ 1 errors have been written to the FFST (first-failure support technology) directory		Queue Manager / General
⚠ SSL key repository file cannot be found	QM1	Queue Manager / SSL
⚠ Stash file for SSL key repository cannot be found	QM1	Queue Manager / SSL
⚠ No dead-letter queue is defined for QM1	QM1	Queue Manager / General
ⓘ Test completed: 'Verify process names'	QM1	Queue Manager / Triggering
ⓘ Test completed: 'Verify initiation queue definitions'	QM1	Queue Manager / Triggering
ⓘ Test completed: 'Verify use of triggered queues'	QM1	Queue Manager / Triggering
ⓘ Test completed: 'Verify process definitions of queues'	QM1	Queue Manager / Triggering
ⓘ Test completed: 'Verify trigger data in queue definitions'	QM1	Queue Manager / Triggering
ⓘ Test completed: 'Verify process definitions'	QM1	Queue Manager / Triggering
ⓘ Test completed: 'Discover topic inheritance'	QM1	Queue Manager / Topics
ⓘ Test completed: 'Verify the basic topic setup'	QM1	Queue Manager / Topics
ⓘ Test completed: 'Verify topic names'	QM1	Queue Manager / Topics
ⓘ Test completed: 'Verify subscription names'	QM1	Queue Manager / Subscriptions
ⓘ Test completed: 'Verify the basic subscription setup'	QM1	Queue Manager / Subscriptions
ⓘ Test completed: 'Verify that channels have been restarted'	QM1	Queue Manager / SSL
ⓘ Test completed: 'Verify SSL key repository files'	QM1	Queue Manager / SSL
ⓘ Test completed: 'Verify SSL channel authentication'	QM1	Queue Manager / SSL
ⓘ Test completed: 'Verify SSL client authentication'	QM1	Queue Manager / SSL
ⓘ Test completed: 'Verify SSL peer values'	QM1	Queue Manager / SSL
ⓘ Test completed: 'Verify alias queue definitions'	QM1	Queue Manager / Queues
ⓘ Test completed: 'Verify that queues are put-enabled'	QM1	Queue Manager / Queues
ⓘ Test completed: 'Verify remote queue definitions'	QM1	Queue Manager / Queues
ⓘ Test completed: 'Identify full queues'	QM1	Queue Manager / Queues

Introduction to IBM MQ Explorer

© Copyright IBM Corporation 2017

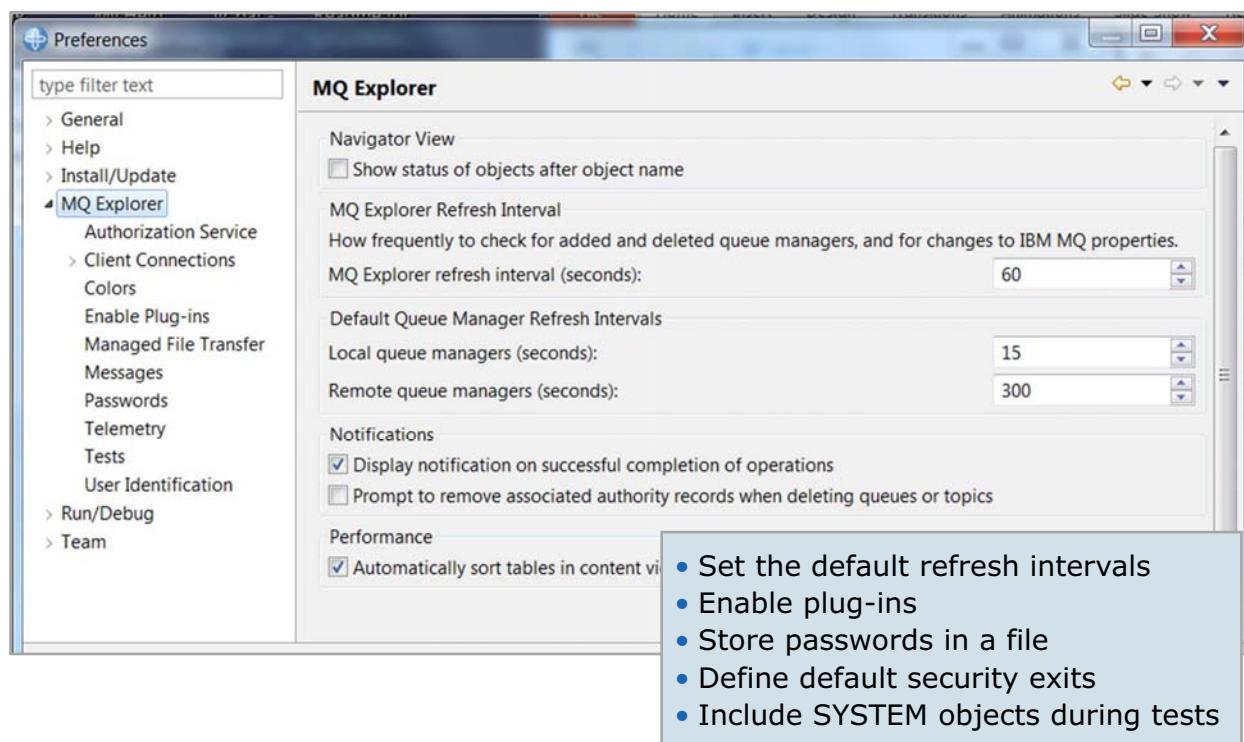
Figure 4-36. IBM MQ test results example

This figure is an example of the MQ test results.

An entry is provided for each test, which is identified by a **Category**. The test results are listed in order with all tests that generated errors listed first, followed by all tests that produced warnings, and finally, all tests that completed successfully.



IBM MQ Explorer preferences



Introduction to IBM MQ Explorer

© Copyright IBM Corporation 2017

Figure 4-37. IBM MQ Explorer preferences

MQ Explorer preferences can be used to change some of the default behaviors and display options. Preferences include default refresh intervals and enabled plug-ins.

To display the MQ Explorer preferences, click **Windows > Preferences**.

IBM Training

Context-sensitive help

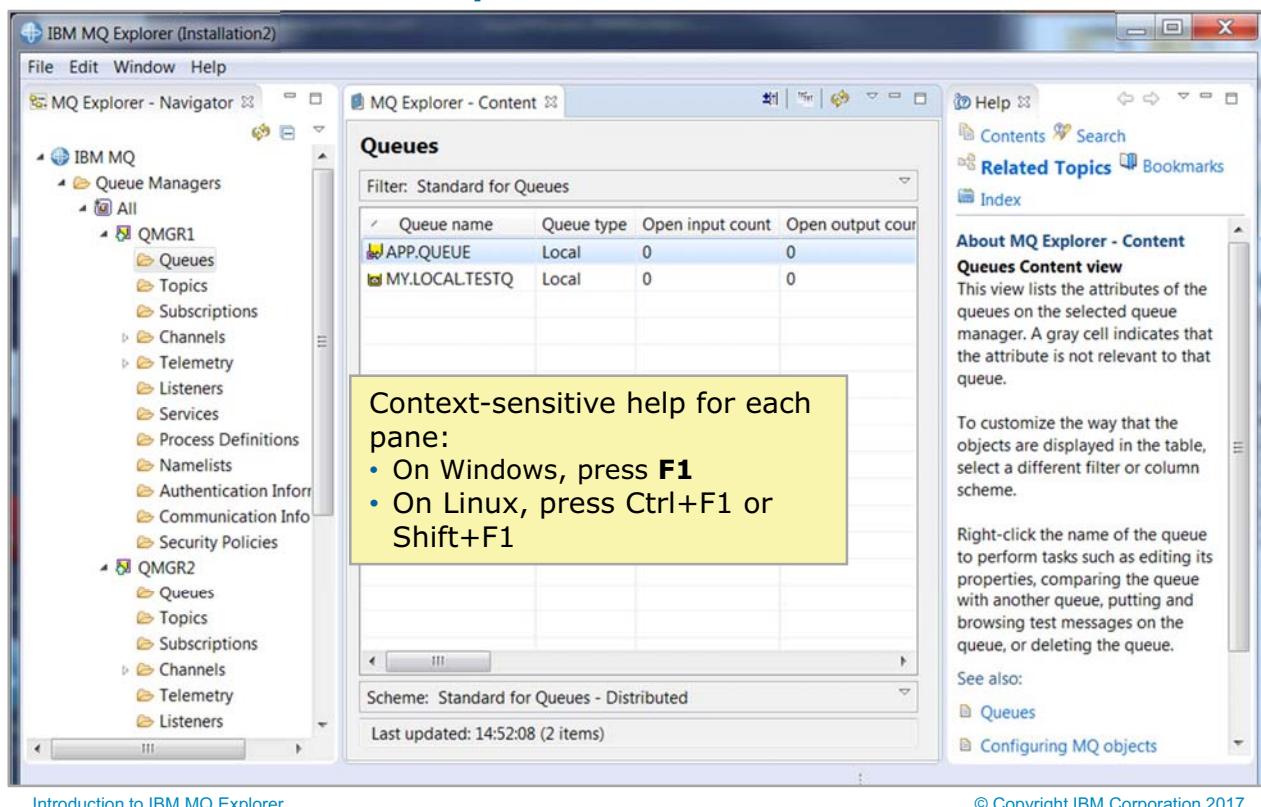


Figure 4-38. Context-sensitive help

Quick access to reference documentation is supported in MQ Explorer by using context-sensitive help.

Context-sensitive help can be started by pressing **F1** on your keyboard while your cursor is over the required field.

The MQ Explorer **Help** preferences control the help information display mode. By default, the **Help** view is an extra view, but it can be changed to a window as shown in the figure, if you prefer. Help preferences are configured by clicking **Windows > Preferences > Help**.

Unit summary

- Use IBM MQ Explorer to create a local queue manager and queues
- Use IBM MQ Explorer to create and manage queue manager sets
- Use IBM MQ Explorer to run tests to verify IBM MQ object definitions

Introduction to IBM MQ Explorer

© Copyright IBM Corporation 2017

Figure 4-39. Unit summary

Review questions

1. Displaying the status of a selected queue shows:
 - A. Current queue depth
 - B. Name of the applications that are currently using the queue
 - C. The time of the last PUT operation on the queue
 - D. Whether messages are persistent
 - E. User ID of the process that is using the queue
2. True or False: Queue managers can be grouped into sets, and the IBM MQ Explorer allows actions on all the queue managers that are defined to the set.



Introduction to IBM MQ Explorer

© Copyright IBM Corporation 2017

Figure 4-40. Review questions

Write your answers down here:

1.

2.

Review answers

1. Displaying the status of a selected queue shows:
 - A. Current queue depth
 - B. Name of the applications that are currently using the queue
 - C. The time of the last PUT operation on the queue
 - D. Whether messages are persistent
 - E. User ID of the process that is using the queue

The answer is A, B, C, and E.
2. True or False: Queue managers can be grouped into sets, and the IBM MQ Explorer allows actions on all the queue managers that are defined to the set.
The answer is True.



Introduction to IBM MQ Explorer

© Copyright IBM Corporation 2017

Figure 4-41. Review answers

Exercise: Using IBM MQ Explorer to create queue managers and queues

Introduction to IBM MQ Explorer

© Copyright IBM Corporation 2017

Figure 4-42. Exercise: Using IBM MQ Explorer to create queue managers and queues

In this exercise, you use the IBM MQ Explorer to create a queue manager, start it, and then create queues. You also use IBM MQ Explorer to create queue manager sets to simplify the management of many queue managers.

Exercise objectives

- Use IBM MQ Explorer to create a local queue manager, local queues, and alias queues
- Use IBM MQ Explorer to display and modify queue manager and queue properties
- Use IBM MQ Explorer to create a queue manager set



Introduction to IBM MQ Explorer

© Copyright IBM Corporation 2017

Figure 4-43. Exercise objectives

See the *Course Exercises Guide* for the exercise description and detailed instructions.

Unit 5. Testing the IBM MQ implementation

Estimated time

00:30

Overview

This unit provides a brief introduction to the IBM MQ Message Queue Interface (MQI). You also learn about the IBM MQ sample programs that you can use to test IBM MQ applications and the network.

How you will check your progress

- Checkpoint
- Hands-on exercises

References

IBM MQ product documentation

Unit objectives

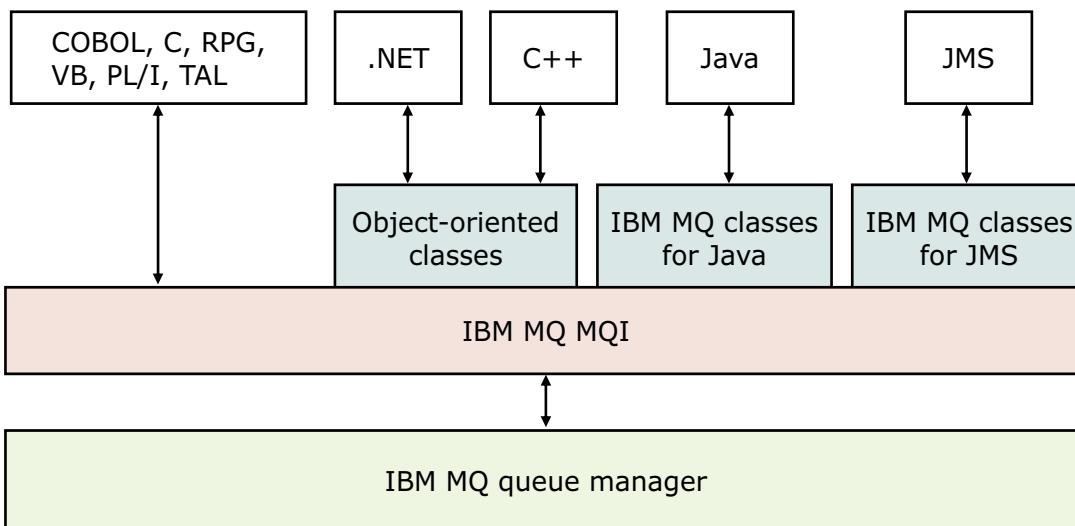
- Recognize IBM MQ MQI calls in a program
- Explain the purpose of the fields in the IBM MQ message descriptor
- Use IBM MQ sample programs to put, get, and browse messages
- Use IBM MQ Explorer to put, get, and browse messages

Testing the IBM MQ implementation

© Copyright IBM Corporation 2017

Figure 5-1. Unit objectives

IBM MQ Message Queue Interface (MQI)



- Procedural languages use the MQI to access message queuing services that IBM MQ queue managers provide

Testing the IBM MQ implementation

© Copyright IBM Corporation 2017

Figure 5-2. IBM MQ Message Queue Interface (MQI)

The IBM MQ Message Queue Interface (MQI) allows an application to access its queues and the messages they contain. The MQI is a simple application programming interface that is consistent across all operating systems that MQ supports. The MQI effectively protects applications from needing to know how a queue manager physically manages messages and queues. The MQI allows full access to MQ messaging support.

A common application programming interface across all supported operating systems is one of the major benefits of MQ.

You can develop client applications by using the MQ classes for .NET, MQ classes for Java, or MQ classes for Java Message Service (JMS).

MQ classes for .NET allow a program that is written in the .NET programming framework to connect to MQ as an MQ MQI client or to connect directly to an MQ server.

JMS is a specification of a portable application programming interface (API) for asynchronous messaging. JMS is an object-oriented Java API with a set of generic messaging objects for programmers to write event-based messaging applications. You can use Java and JMS clients on IBM i, UNIX, Linux, and Windows.

Basic MQI calls

- Send messages
 - MQPUT: Put message
 - MQPUT1: Put one message (MQOPEN + MQPUT + MQCLOSE)
- Receive messages
 - MQGET: Get message
- Housekeeping calls
 - MQCONN: Connect queue manager
 - MQDISC: Disconnect queue manager
 - MQOPEN: Open object
 - MQCLOSE: Close object
- View or change properties
 - MQINQ: Inquire object attributes
 - MQSET: Set object attributes
- Perform transactions
 - MQBEGIN: Begin unit of work
 - MQCMIT: Commit changes
 - MQBACK: Back out changes

Testing the IBM MQ implementation

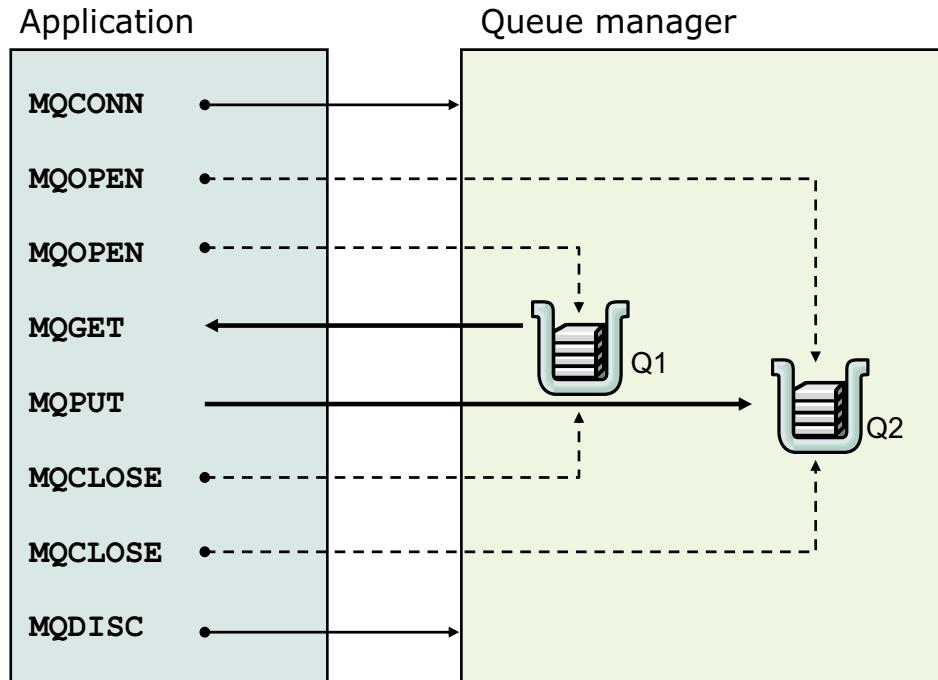
© Copyright IBM Corporation 2017

Figure 5-3. Basic MQI calls

By using the most basic MQI calls, an application can put a message on a queue and get a message from a queue. The MQI calls are described in detail in the production documentation.

- **MQPUT** and **MQPUT1** put a message on a named queue. Generally, a message is added to the end of a queue.
- **MQGET** gets a message from a named queue. Generally, a message is removed from the front of a queue.
- **MQCONN**, **MQCONNX**, and **MQDISC** are used by an application to connect to and disconnect from a queue manager. An application must connect to a queue manager before it can send any more MQI calls.
- **MQOPEN** and **MQCLOSE** open a queue for specified operations and close the queue when access to it is no longer required. An application must open a queue before it can put messages on it, or get messages from it.
- **MQINQ** and **MQSET** inquire and set the attributes of an object. All MQ objects, such as a queue, a process, and the queue manager object, have a set of attributes.
- **MQBEGIN**, **MQCMIT**, and **MQBACK** allow an application to put and get messages as part of a unit of work.

Basic MQI calls in use



Testing the IBM MQ implementation

© Copyright IBM Corporation 2017

Figure 5-4. Basic MQI calls in use

The MQI is a simple call interface. The figure shows the most common calls that are used to connect and disconnect to a queue manager, and to put and get messages from a queue.

In a typical sequence:

- MQCONN connects to the queue manager.
- MQOPEN establishes access to an MQ object such as a queue.
- MQGET retrieves a message from a queue.
- MQPUT places a message on a queue or publishes to a new topic.
- MQCLOSE releases access to an MQ object.
- MQDISC disconnects from the queue manager.

MQ application development is taught in course WM513, *IBM MQ V9 Application Programming*.

Order of retrieving messages

- Application can retrieve messages on a queue in the same order as another application puts them on the queue, provided that:
 - Messages all have the same priority
 - Messages are all put within the same unit of work, or all put outside of a unit of work
 - No other application is getting messages from the queue
 - Queue is local to the putting application
 - Messages might be interspersed with messages that are put by other applications
- If the queue is not local to the putting application, the order of retrieval is still preserved, provided that:
 - First three conditions that are listed in the preceding list still apply
 - Only a single path is configured for the messages
 - No message is put on a dead-letter queue
 - No non-persistent messages are transmitted over a fast message channel

Testing the IBM MQ implementation

© Copyright IBM Corporation 2017

Figure 5-5. Order of retrieving messages

Most applications do not have a strict need to process messages in the same order as they were created, but some applications might have such a requirement, even a legal obligation to do so.

MQ documents the conditions under which an application can retrieve a sequence of messages from a queue in the same order that another application puts them. The conditions assume that the application that retrieves the sequence of messages is not using selection criteria. If the conditions are not met, applications must either include sequencing information in the application data of a message or establish a means of acknowledging receipt of a message before the next one is sent.

The first set of conditions in the figure assumes that the application that puts the messages and the application that gets the messages are both connected to the same queue manager, so no remote queuing is involved.

The second set of conditions describes the process when the “putting” and “getting” applications are connected to different queue managers.

On a fast message channel, nonpersistent messages overtake persistent messages on the same channel and so arrive out of sequence. This behavior is because the receiving MCA commits a nonpersistent message on its destination queue as soon as it receives it instead of waiting for the end of the batch.



Browsing messages in IBM MQ Explorer

1 Right-click the queue and then click **Browse Messages**

Position	Put date/time	User identifier	Put application name	Format	Total length	Data length	Message data
1	Sep 7, 2016 3:54:46 PM	wattss	Install\bin64\MQExplorer.exe	MQSTR	14	14	This is a test
2	Sep 7, 2016 3:55:28 PM	wattss	Install\bin64\MQExplorer.exe	MQSTR	9	9	Message 2
3	Sep 7, 2016 3:55:35 PM	wattss	Install\bin64\MQExplorer.exe	MQSTR	10	10	Message 3

2 Double-click a message to view the message properties

Figure 5-6. Browsing messages in IBM MQ Explorer

You can browse the queue at any time to see the messages that are on a queue.

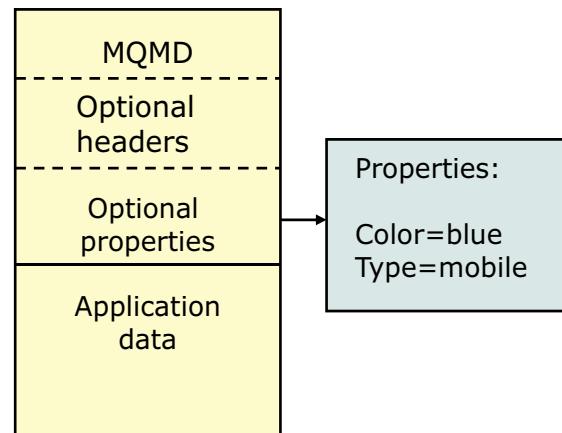
This figure and the next show the steps for browsing the contents of a queue in MQ Explorer.

1. Click **Queues** in the **MQ Explorer - Navigator** view to display the **Queues** content view.
2. Right-click the queue and then click **Browse Messages**.
3. The **Message browser** view provides a summary of each message that is on the queue.

For more information about a specific message, double-click the message in the **Message browser** view.

Message properties

- Attributes of IBM MQ messages that are associated with the message but not part of the body
 - Consist of a textual name and a value of a particular type
- Properties can be integers, strings, Booleans, and more
- Allows explicit statement of relationships between messages
- Can be viewed in IBM MQ Explorer



Testing the IBM MQ implementation

© Copyright IBM Corporation 2017

Figure 5-7. Message properties

Message selectors use message properties to filter publications to topics or to selectively get messages from queues. Message properties are optional. They can be used to include business data or state information without storing it in the message data.

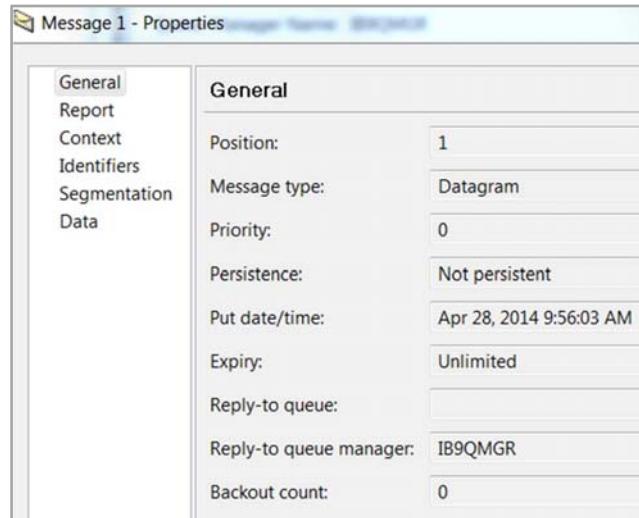
Message properties are arbitrary name-value pairs that can be assigned to a message. Properties are assigned by using two MQI verbs: MQSETPROP to set the properties and MQINQPROP to inquire on the message property value.

Message properties are primarily used for establishing explicit relationships between messages by supplementing the message ID and correlation ID. For example, message properties can specify that *message X* is a reply to *message Y*.

You can view the message properties in MQ Explorer or by using the WebSphere MQ SupportPac RFHUtil.

General message properties

- **Position:** Position in queue
- **Message type:** Type of message
- **Priority:** Priority of message
- **Persistence:** Indicates whether message is persistent or not persistent
- **Put date/time:** Date and time message was put on queue
- **Expiry:** Time after which message is eligible to be discarded
- **Reply-to queue:** Name of message queue to which application that gets the message should send reply messages
- **Reply-to queue manager:** Name of queue manager on which reply-to queue is defined
- **Backout count:** Number of times MQGET call returned the message as part of a unit of work, and then backed out



The screenshot shows the 'Message 1 - Properties' dialog box in IBM MQ Explorer. The left pane lists categories: General, Report, Context, Identifiers, Segmentation, and Data. The right pane displays the 'General' properties for Message 1:

Position:	1
Message type:	Datagram
Priority:	0
Persistence:	Not persistent
Put date/time:	Apr 28, 2014 9:56:03 AM
Expiry:	Unlimited
Reply-to queue:	
Reply-to queue manager:	IB9QMGR
Backout count:	0

Testing the IBM MQ implementation

© Copyright IBM Corporation 2017

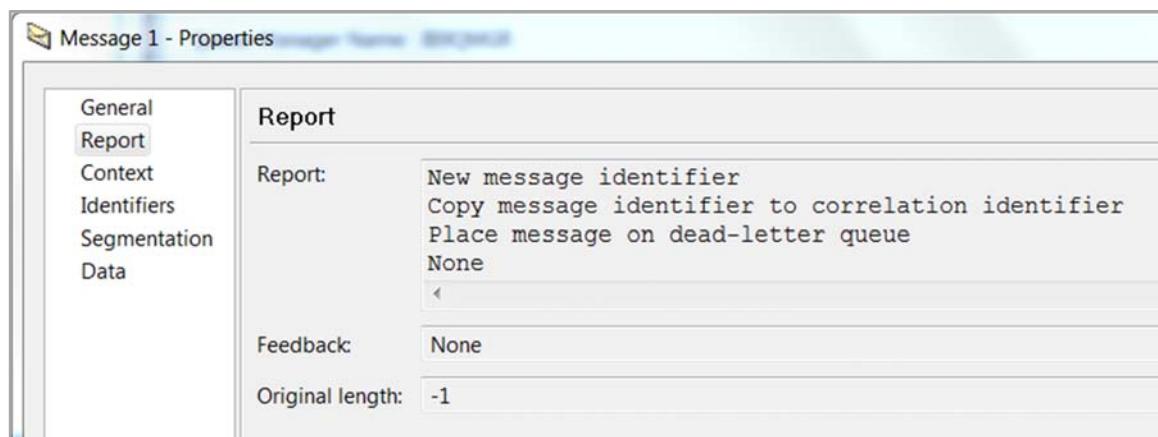
Figure 5-8. General message properties

You can view parts of the MQMD and other properties of the message in MQ Explorer. This figure provides an overview of the **General** properties in the MQ Explorer message browser.

The application sets some properties, such as message type, priority, and persistence, in the MQMD.

Report message properties

- Report messages inform applications about events such as the occurrence of an error when processing a message
 - **Report:** Sender application specifies whether report messages are required, whether the application data is to be included in report messages, and how message and correlation identifiers in the report or reply message are set
 - **Feedback:** Nature of report
 - **Original length:** Length of original message to which the report relates



Testing the IBM MQ implementation

© Copyright IBM Corporation 2017

Figure 5-9. Report message properties

A report message is a message about another message. It is used to inform the application about expected or unexpected events that relate to the original message.

The **Report** properties page displays the attributes that are related to report messages. This figure provides an overview of the **Report** properties in the MQ Explorer message browser.

Context message properties

- Context information allows application that retrieves message to determine originator
- *Identity context* identifies user of the application that first put the message
 - **User identifier** of application that originated message
 - **Accounting token** allows application to charge for work
 - **Application identity data** that application defines to provide information about the message or its originator
- *Origin context* describes the application that put the message on the current queue
 - **Application type** that put the message
 - **Put application name** of the application that put the message
 - **Application origin data** that application defines to provide more information about message origin

The screenshot shows a Windows-style dialog box titled "Message 1 - Properties". On the left, there's a vertical list of tabs: General, Report, Context (which is selected), Identifiers, Segmentation, and Data. On the right, under the "Context" tab, there are several entries:

User identifier:	wattssibm
Application type:	32-bit Windows
Put application name:	Sphere MQ\bin\MQExplorer.exe
Application identity data:	(empty)
Application origin data:	(empty)
Accounting token:	00000 16 01 05 15 00 00 00010 50 9D D5 E8 03 00

Testing the IBM MQ implementation

© Copyright IBM Corporation 2017

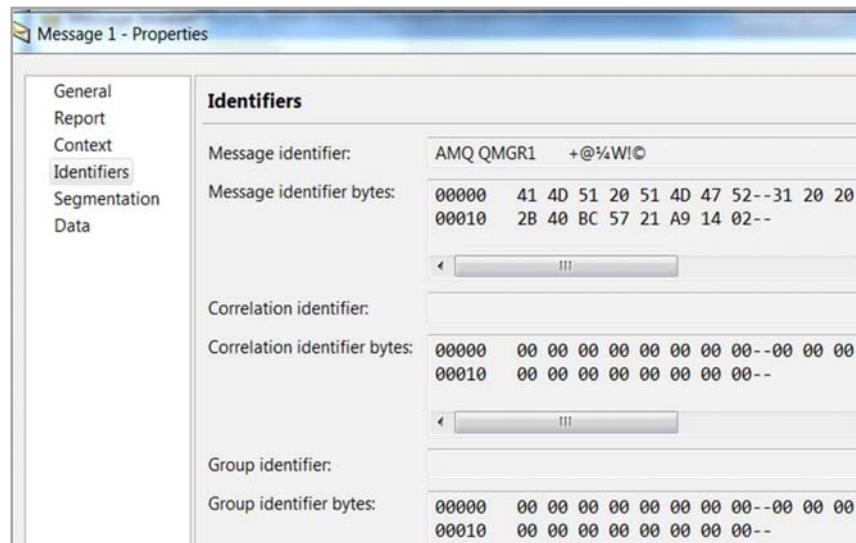
Figure 5-10. Context message properties

The **Context** message properties page displays information from the sender application about the message.

This figure provides an overview of the message properties that are contained on the **Context** message properties page.

Identifiers message properties

- Displays identification information that is associated with the message
 - **Message identifier:** Distinguishes one message from another
 - **Message identifier bytes:** Message identifier in byte form
 - **Correlation identifier:** Identifier that application can use to relate one message to another, or to relate message to other work that the application completes
 - **Correlation identifier bytes:**
Correlation identifier in byte form
 - **Group identifier:** Identifies particular message group or logical message to which the physical message belongs
 - **Group identifier bytes:** Group identifier in byte form



Testing the IBM MQ implementation

© Copyright IBM Corporation 2017

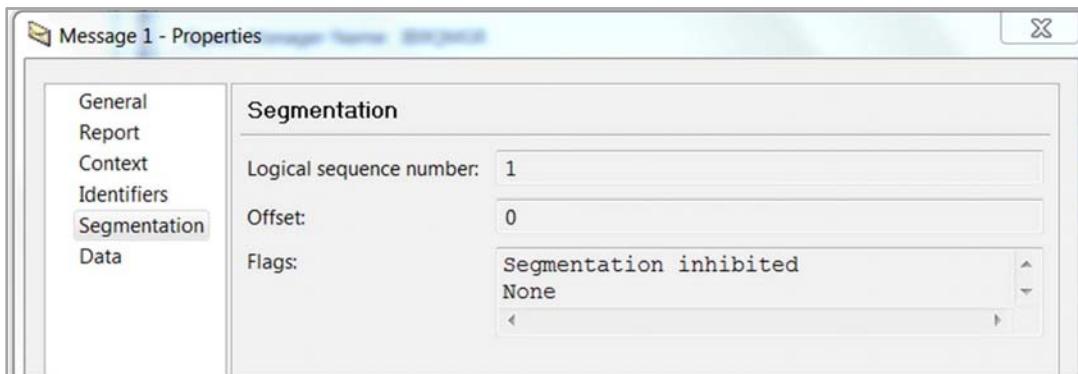
Figure 5-11. Identifiers message properties

The **Identifiers** message properties page displays identification information that is associated with the message.

This figure provides an overview of the message properties that are contained on the **Identifiers** page in MQ Explorer.

Segmentation message properties

- Displays attributes related to segmenting large messages
 - **Logical sequence number:** Sequence number of logical message within the group
 - **Offset:** Offset of data in physical message from the start of the logical message
 - **Flags:** Message flags that specify attributes of message, or control its processing



Testing the IBM MQ implementation

© Copyright IBM Corporation 2017

Figure 5-12. Segmentation message properties

The **Segmentation** message properties page in MQ Explorer displays the attributes that are related to segmenting large messages.

This figure provides an overview of the message properties that are contained on the **Segmentation** message properties page.

Data message properties

- Contains information about the message data and data format
 - Total length:** Length of the retrieved message
 - Data length:** Length of the original message
 - Format:** Name that message sender uses to indicate to the receiver the type of the data in the message
 - Coded character set identifier:** Identifier of character data in the application message data
 - Encoding:** Numeric encoding of numeric data in the message
 - Message data:** Message data in human readable ASCII text
 - Message data bytes:** Message data in hexadecimal format

Message 1 - Properties	
	Data
General	
Report	
Context	
Identifiers	
Segmentation	
Data	
	Total length: 14
	Data length: 14
	Format: MQSTR
	Coded character set identifier: 1208
	Encoding: 546
	Message data: This is a test
	Message data bytes: 00000 54 68 69 73 20 69 73 20

Testing the IBM MQ implementation

© Copyright IBM Corporation 2017

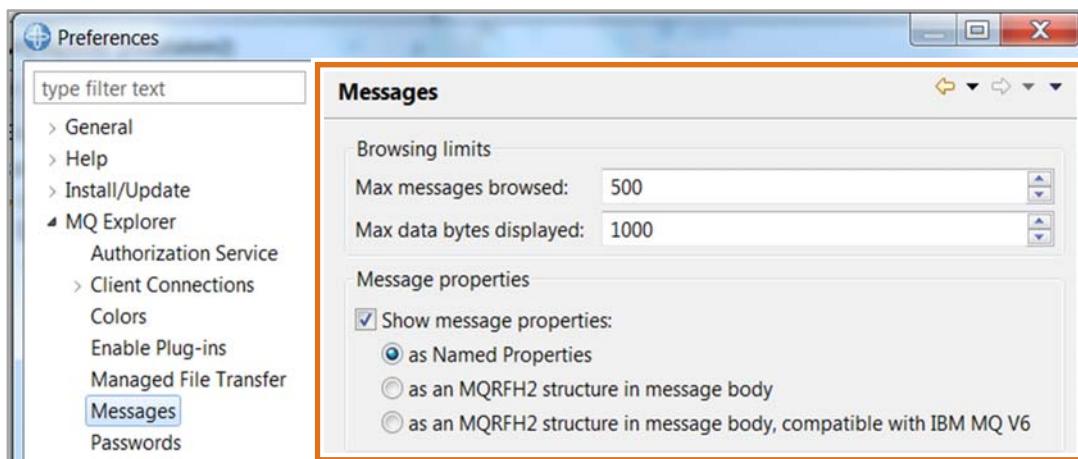
Figure 5-13. Data message properties

The **Data** message properties page in MQ Explorer displays the message data itself and information about the data format.

This figure provides an overview of the message properties on the **Data** message properties page.



Message property preferences in IBM MQ Explorer



- Browsing limits
 - Maximum number of messages that can be browsed (1 – 5000)
 - Maximum number of bytes of data to display per message (0 – 16384)
- Show message properties not contained in MQMD
 - As name-value pairs
 - As an MQRFH2 header structure

Testing the IBM MQ implementation

© Copyright IBM Corporation 2017

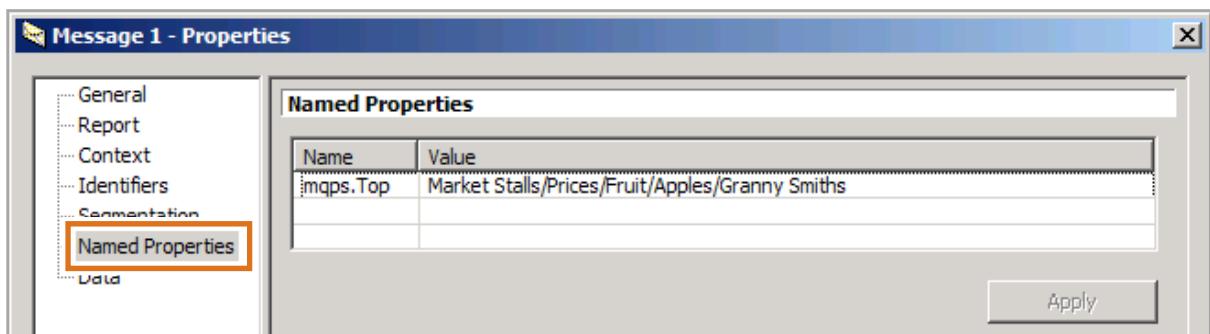
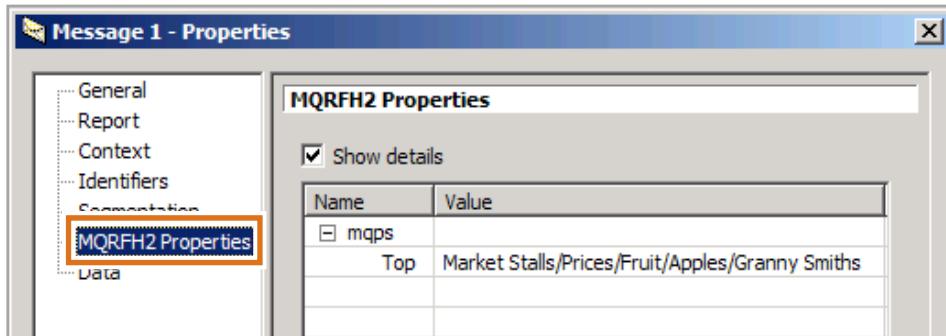
Figure 5-14. Message property preferences in IBM MQ Explorer

MQ Explorer can be configured to read properties in messages on the queue based on the message properties.

The way message properties are shown in MQ depends on the configuration of MQ Explorer and the definition of the queue.

Depending on the **Preference** property setting, you see properties either as **Named Properties** or **MQRFH2 Properties** when you look at the properties of a browsed message.

Message property preferences: Examples



Testing the IBM MQ implementation

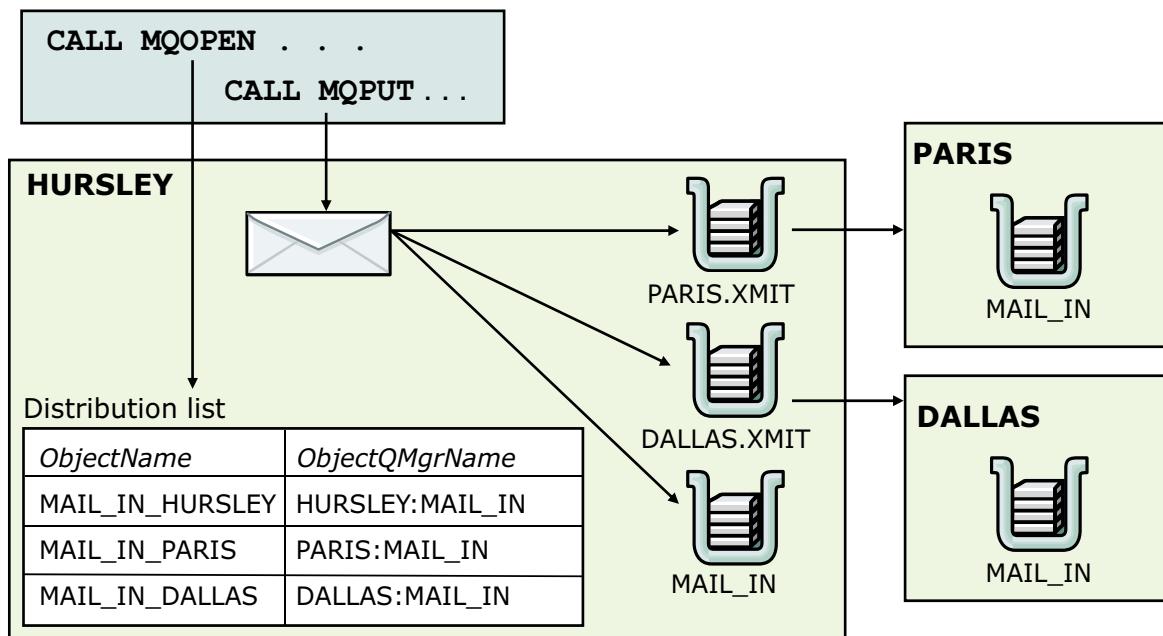
© Copyright IBM Corporation 2017

Figure 5-15. Message property preferences: Examples

This figure shows the two different ways that you can display message properties, based on your selection in the MQ Explorer **Messages Preferences**.

If you select the option to show message properties as named properties, the **Named Properties** message properties contain the name of the message property and the value of the named property.

Distribution lists



- Allow an application to send one `MQOPEN` call to open multiple queues and a one `MQPUT` or `MQPUT1` call to put a message to each of those queues

Testing the IBM MQ implementation

© Copyright IBM Corporation 2017

Figure 5-16. Distribution lists

You can use distribution lists to put a message to multiple destinations in a single `MQPUT` or `MQPUT1` call. A single `MQOPEN` call can open multiple queues, and a single `MQPUT` call can then put a message to each of those queues.

A distribution list is a set of object records, where each object record has two fields. *ObjectName* specifies the queue name. *ObjectQMgrName* specifies the queue manager name of a single destination queue.

A distribution list might contain the name of an alias queue or the name of a local definition of a remote queue. The example of a distribution list that is shown in the figure makes the following assumptions:

- `MAIL_IN_HURSLEY` is an alias queue that resolves to the local queue `MAIL_IN` on queue manager `HURSLEY`.
- `MAIL_IN_PARIS` is a remote queue definition that resolves to the queue `MAIL_IN` on queue manager `PARIS` by using transmission queue `PARIS.XMIT`.
- `MAIL_IN_DALLAS` is a remote queue definition that resolves to the queue `MAIL_IN` on queue manager `DALLAS` by using transmission queue `DALLAS.XMIT`.

After a distribution list is opened, the application can call `MQPUT` to put a message on the queues in the list. As a response to the call, the queue manager puts **one** copy of the message on each

local queue, including the transmission queues. So, only one copy of the message is put on the transmission queue DALLAS even though the message is ultimately destined for two queues.

A distribution header and a transmission queue header prefix the application data on the transmission queue. The message that is transmitted between HURSLEY and DALLAS effectively contains a distribution list for the two destinations.

An important property of the implementation is that a message that is destined for multiple queues is replicated only at the last possible point at each stage of its journey. In this way, network traffic is minimized.

You learn more about transmission queues and queue manager communication in Unit 6, “Implementing distributed queuing”.

Testing IBM MQ configuration

- IBM MQ sample programs
 - Available as C source, COBOL source, and C runtime
 - Runtime files on Linux: `/opt/mqm/samp/bin`
 - Runtime files on Windows:
`C:\Program Files\IBM\MQ\Tools\c\Samples\Bin64`
- IBM MQ Explorer
 - Put test messages
 - Browse messages
- WebSphere MQ SupportPac IH03 RfhUtil
 - Put, get, and browse test messages
 - View message content in multiple formats
 - Modify message headers
 - Test functions for JMS and publish/subscribe

[Testing the IBM MQ implementation](#)

© Copyright IBM Corporation 2017

Figure 5-17. Testing IBM MQ configuration

MQ provides many ways to test your MQ configuration.

IBM MQ contains sample programs that illustrate the use of MQI calls. MQ sample programs are available as C source, COBOL source, and C runtime. They can be run from any location on Windows. On Linux and UNIX, you must run the sample program from its home directory or set the PATH to the sample program directory.

MQ Explorer also has some sample programs for putting and browsing messages.

Many MQ administrators use the WebSphere SupportPac IH03 RFHutil for testing queues. It allows test messages to be captured and stored in files. Messages can also be read and displayed in various formats. At the time of publication of this course, WebSphere SupportPac IH03 was not yet updated to support MQ V9.

Testing with IBM MQ sample programs

- **amqsput QName QMgrName**
Read text from the standard input device and put messages
- **amqsget QName QMgrName**
Get messages and write to the standard output device
- **amqsbcg QName QMgrName**
Browse messages and show both application data and message descriptor
- **amqsbr QName QMgrName**
Browse messages and show application data only
- **amqsreq QName QMgrName ReplyToQName**
Read lines of text from the standard input device, convert them to request messages, and MQPUT the messages on the named queue

Note: Queue manager name is optional when referencing the default queue manager

Figure 5-18. Testing with IBM MQ sample programs

This figure describes some of the sample programs that MQ provides.

The Put sample program, **amqsput**, connects to the queue manager and opens the queue. It reads lines of text from the standard input device, generates a message from each line of text, and puts the messages on the named queue. After it reads a null line or the EOF character from the standard input device, it closes the queue, and disconnects from the queue manager.

The Get sample program, **amqsget**, connects to the queue manager, opens the queue for input, and gets all the messages from the queue. It writes the text within the message to the standard output device, waits 15 seconds (60 seconds if no message exists at the start) in case any more messages are put on the queue. The program then closes the queue and disconnects from the queue manager.

MQ provides two browser sample programs:

- The **amqsbcg** program connects to the queue manager and opens the queue for browsing. It browses all the messages on the queue and writes their contents, in both hexadecimal and character format, to the standard output device. It also shows, in a readable format, the fields in the message descriptor for each message. It then closes the queue and disconnects from the queue manager.
- The **amqsbr** program browses messages on a queue by using the MQGET call.

The Request sample program, **amqsreq**, demonstrates client/server processing. It puts a series of request messages on the target server queue by using the MQPUT call. These messages specify the local queue, SYSTEM.SAMPLE.REPLY, as the reply-to queue, which can be a local or remote queue.

You use many of these sample programs in the lab exercises for this course.

Sample program example

```
C:\Users\MQ_ADMIN>amqspput QL.TEST OMGR1
Sample AMQSPUT0 start
Target queue is QL.TEST
This is my test message
This is another test message
[Press Enter] ←
Sample AMQSPUT0 end
```

A blank line indicates the end of message input

```
C:\Users\MQ_ADMIN>amqsget QL.TEST OMGR1
Sample AMQSGET0 start
message <This is my test message>
message <This is another test message>
C:\Users\MQ_ADMIN>
```

Testing the IBM MQ implementation

© Copyright IBM Corporation 2017

Figure 5-19. Sample program example

This figure shows an example of the **amqspput** and **amqsget** sample programs.

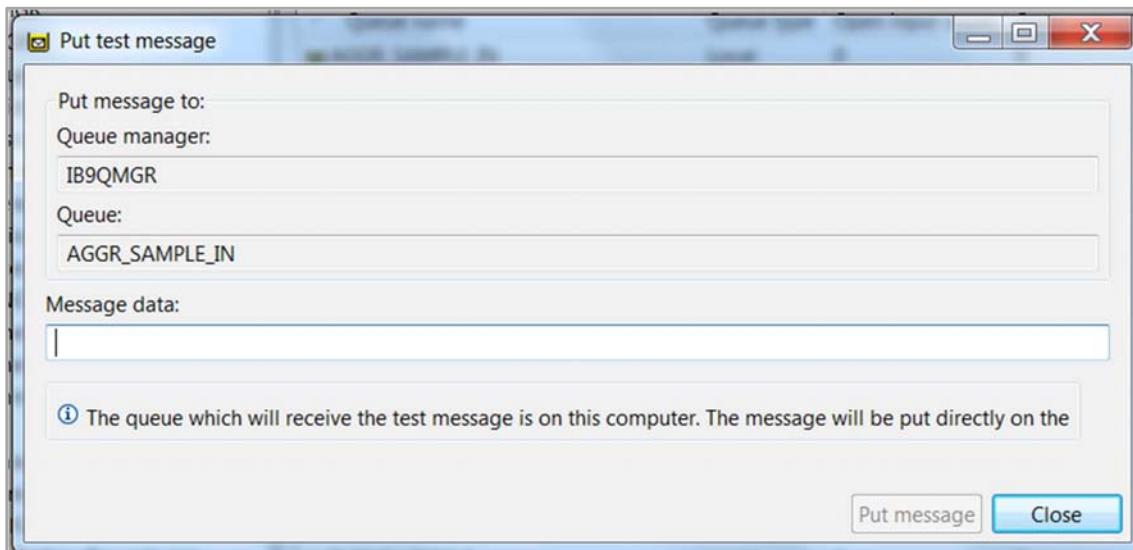
The **amqspput** and **amqsget** sample programs require two parameters: the queue name and the queue manager name.

To use the **amqspput** sample program, enter one or more test messages after the program starts. When you are finished entering test messages, press Enter on a blank line to end the sample program.

As shown in the example, the **amqsget** program gets all the messages that are in the queue, which empties the queue. When you run the **amqsget** sample program, the program waits for some time in case any more messages are written to the queue; it might take a minute for the program to complete.

Testing with IBM MQ Explorer

- Browse messages on queue
- Clear messages on queue
- Put messages on queue
- View message contents and properties



Testing the IBM MQ implementation

© Copyright IBM Corporation 2017

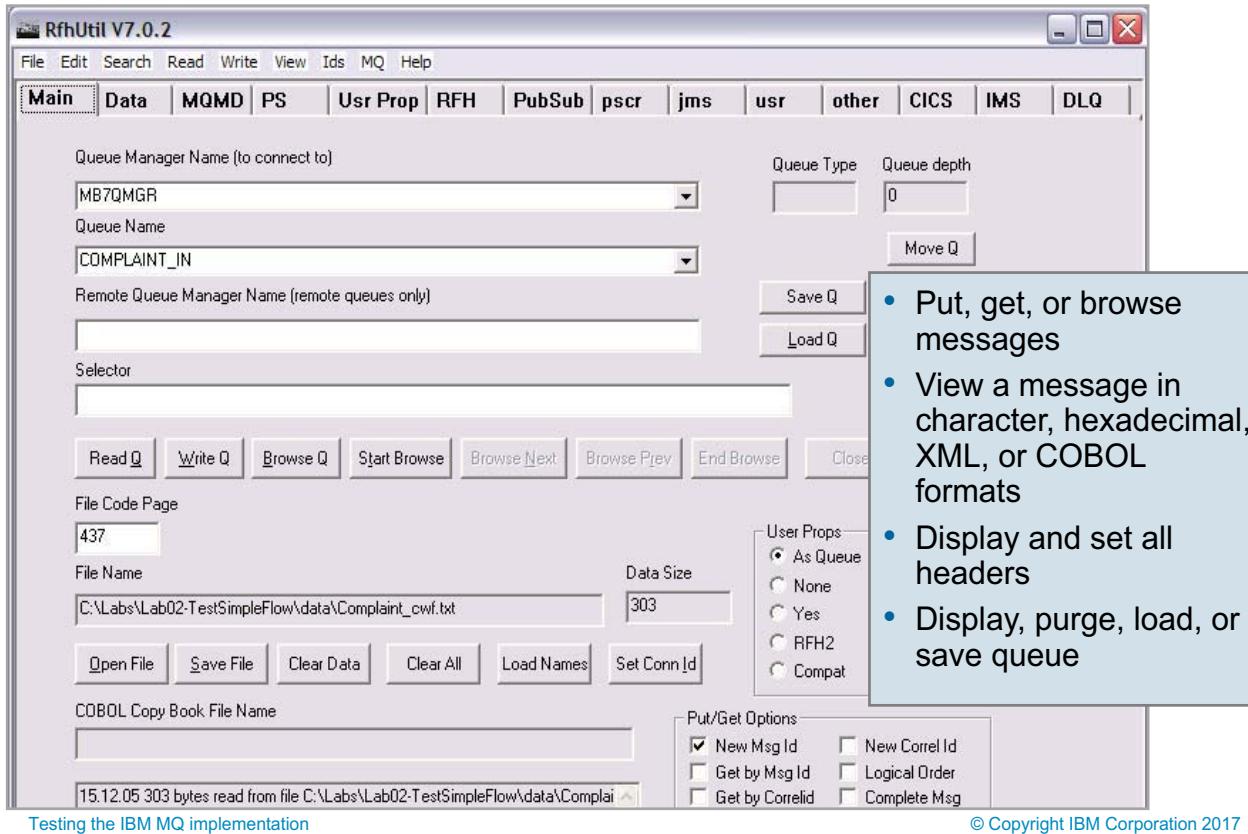
Figure 5-20. Testing with IBM MQ Explorer

With MQ Explorer, you can put messages, browse messages, clear (get) messages, and view message contents and properties.

To use the MQ Explorer test options, right-click the queue name in the **Queues** content view and then click **Test**.

IBM Training

Testing with RfhUtil (SupportPac IH03)



Testing the IBM MQ implementation

© Copyright IBM Corporation 2017

Figure 5-21. Testing with RFHUtil (SupportPac IH03)

The RFHUtil utility program is a WebSphere MQ SupportPac. It reads data from files, queues, or both. It also writes data to files, queues, or both, and displays data in various formats.

The user data portion of the message can be displayed in various formats, but it cannot be changed. Another program must be used to create or change the user data.

With RFHUtil, you can add rules and formatting headers to messages or files. It writes and formats these headers when they are found in messages or files that it reads. The headers can include publish/subscribe commands.

You can download RFHUtil from the WebSphere MQ SupportPac website at:
<http://www.ibm.com/support>

RFHUtil is a category 2 SupportPac. It is free and fully supported.

Unit summary

- Recognize IBM MQ MQI calls in a program
- Explain the purpose of the fields in the IBM MQ message descriptor
- Use IBM MQ sample programs to put, get, and browse messages
- Use IBM MQ Explorer to put, get, and browse messages

Testing the IBM MQ implementation

© Copyright IBM Corporation 2017

Figure 5-22. Unit summary

Review questions

1. True or False: The correlation identifier is normally used to provide an application with a means of matching a reply or report message with the original message.

2. An application can retrieve a message from a queue that is based on:
 - A. Message ID
 - B. Correlation ID
 - C. Next available message that is based on the MsgDeliverySequence value of the queue
 - D. All of the above



Testing the IBM MQ implementation

© Copyright IBM Corporation 2017

Figure 5-23. Review questions

Write your answers down here:

- 1.

- 2.

Review answers

1. True or False: The correlation identifier is normally used to provide an application with a means of matching a reply or report message with the original message.
The answer is True.

2. An application can retrieve a message from a queue that is based on:
 - A. Message ID
 - B. Correlation ID
 - C. Next available message that is based on the MsgDeliverySequence value of the queue
 - D. All of the above
The answer is D.



Exercise: Using IBM MQ sample programs to test the configuration

Testing the IBM MQ implementation

© Copyright IBM Corporation 2017

Figure 5-25. Exercise: Using IBM MQ sample programs to test the configuration

In this exercise, you use the IBM MQ sample programs to put, get, and browse queues and test your queue manager configuration. You then use IBM MQ Explorer and IBM MQ commands to verify the actions of the sample programs. You also define and test an alias queue.

Exercise objectives

- Use IBM MQ sample programs to put messages onto a queue, browse messages on a queue, and get messages from a queue
- Use IBM MQ Explorer and IBM MQ commands to display queue contents
- Define and test an alias queue that refers to another queue



Testing the IBM MQ implementation

© Copyright IBM Corporation 2017

Figure 5-26. Exercise objectives

See the *Course Exercises Guide* for detailed instructions.

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

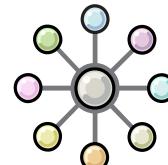
Unit objectives

- 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

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.

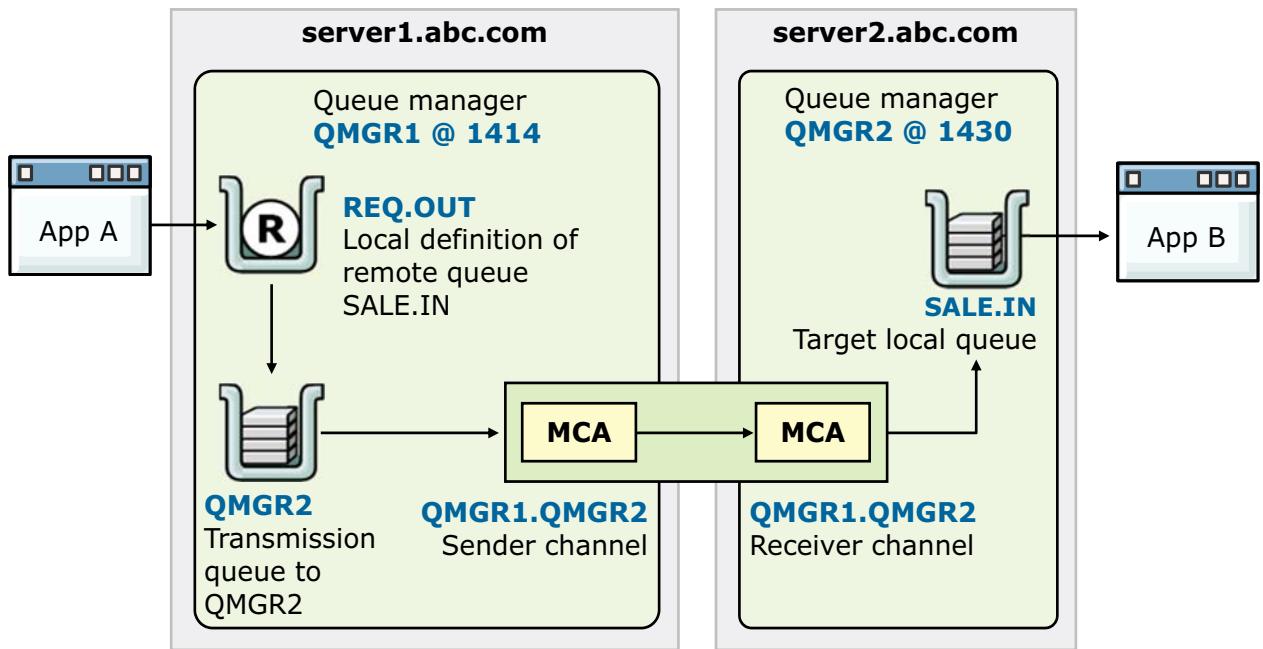
In point-to-point messaging, a sending application must know information about the receiving application before it can send a message to that application. For example, the sending application might need to know the name of the queue to which to send the information, and might also specify a queue manager 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.

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.

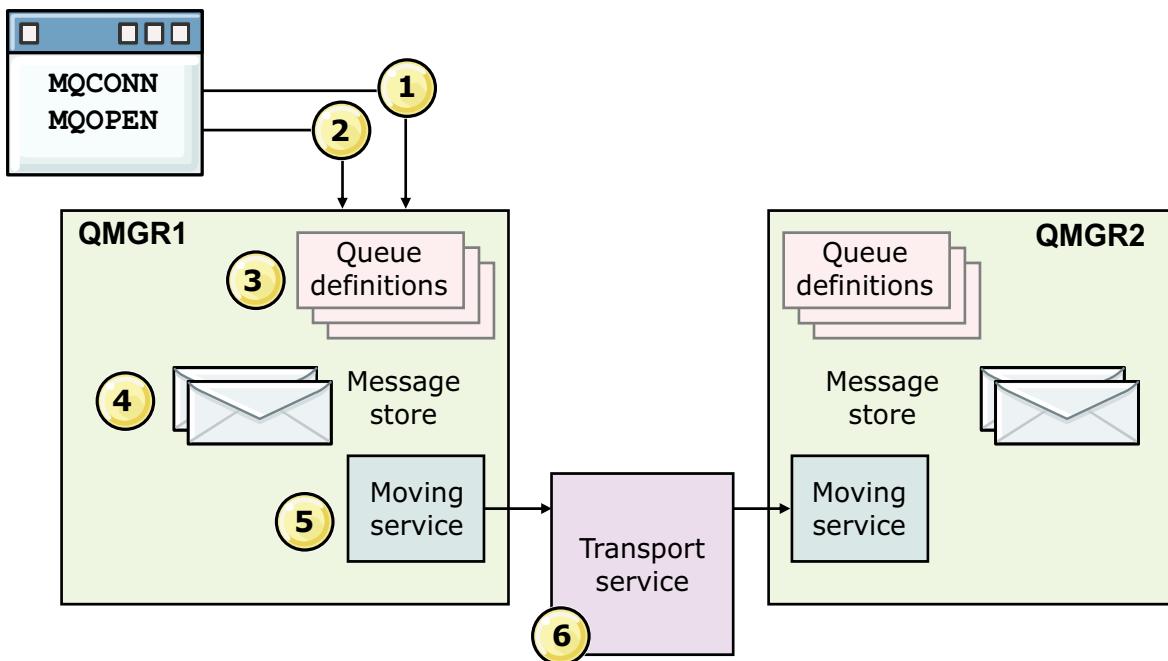
In the example, the sending and receiving applications are on different servers and connect to different queue managers.

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.

Distributed queuing components overview (1 of 2)



Implementing distributed queuing

© Copyright IBM Corporation 2017

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)

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.

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)**

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 **xmlio** attribute identifies the transmission queue.



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

Queue name:	PAYROLLIN
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

Queues
Filter: Standard for Queues

Queue name	Queue type	Open input count
MY.LOCALTESTQ	Local	0
PAYROLLIN	Remote	0
QMGR2	Local	0

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

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

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.



Using IBM MQ Explorer to define a transmission queue

The screenshot shows the 'New Local Queue' dialog in IBM MQ Explorer. The 'General' tab is selected. The 'Queue name' field contains 'QMGR2'. The 'Queue type' field is set to 'Local'. In the 'Usage' dropdown, the option 'Transmission' is highlighted with a red box and a callout bubble. A yellow callout bubble above the dialog states: 'Define local queue and set General > Usage property to Transmission'. Below the dialog, the 'Queues' content view is shown, displaying a table with two rows:

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

A yellow callout bubble next to the 'QMGR2' row in the table states: 'Icon identifies transmission queue in Queues content view'.

Implementing distributed queuing

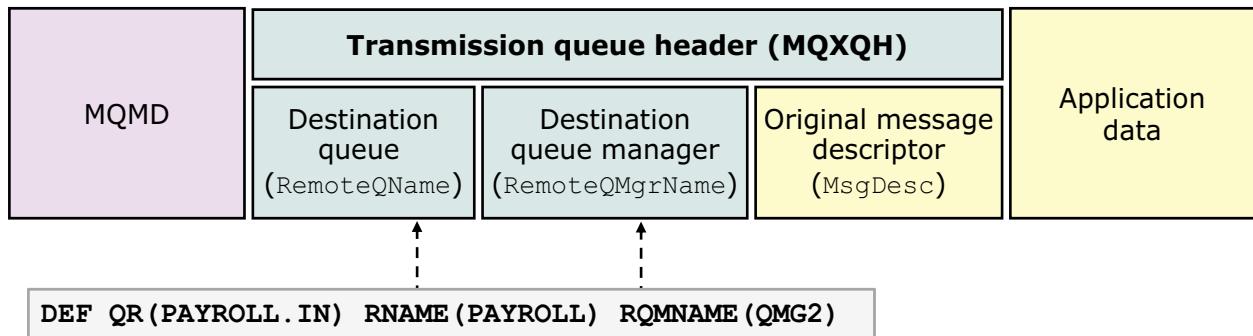
© Copyright IBM Corporation 2017

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.

Transmission queue header



- 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.

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)

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.

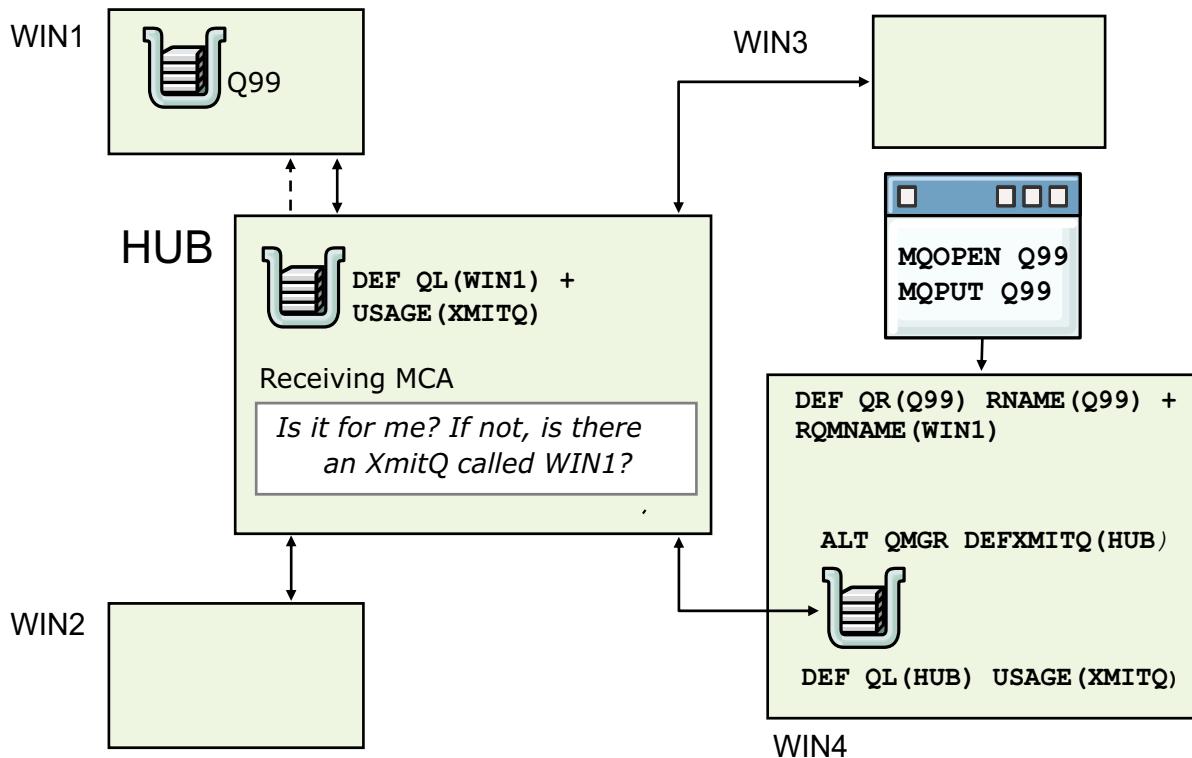
An example command for defining a default transmission queue is:

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.



Example of using a default transmission queue



Implementing distributed queuing

© Copyright IBM Corporation 2017

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

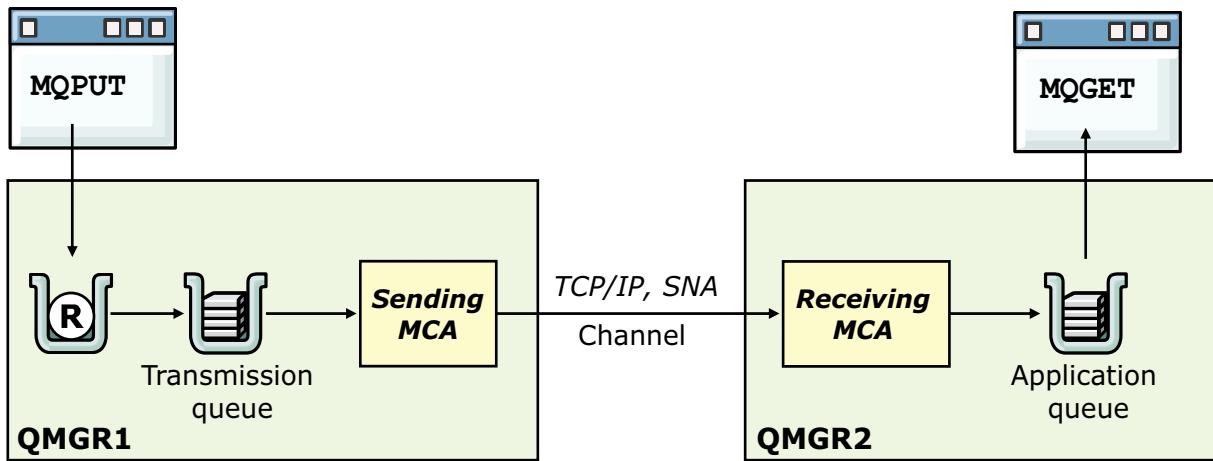
The default transmission queue is the destination for messages that are sent to a remote queue when no other suitable transmission queue is defined.

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.

Message channel



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

[Implementing distributed queuing](#)

© Copyright IBM Corporation 2017

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.

Message channel types

- For distributed queuing
 - Sender: **CHLTYP (SDR)**
 - Server: **CHLTYP (SVR)**
 - Receiver: **CHLTYP (RCVR)**
 - Requester: **CHLTYP (RQSTR)**
- For clustered environments
 - Cluster-sender: **CHLTYP (CLUSSDR)**
 - Cluster-receiver: **CHLTYP (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

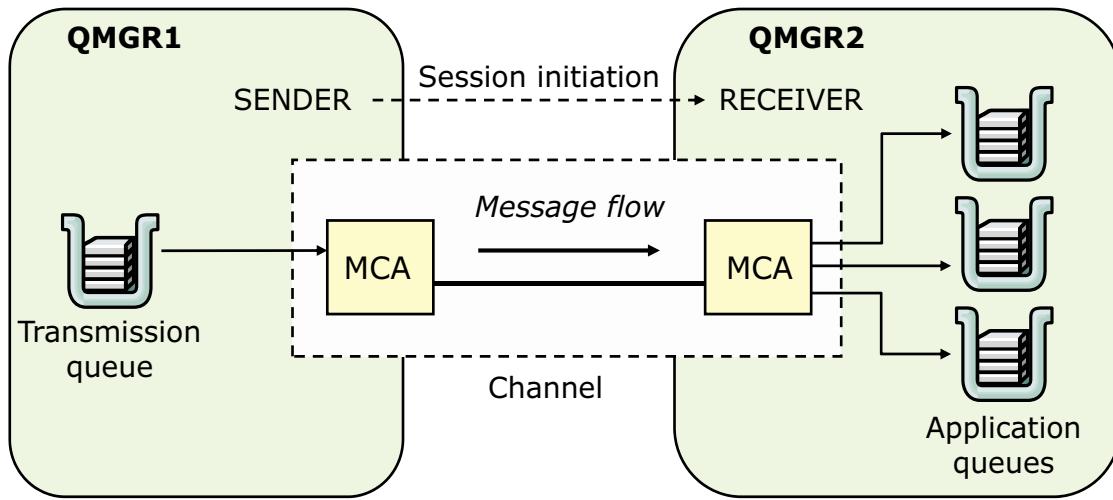
© Copyright IBM Corporation 2017

Figure 6-13. Message channel types

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

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

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

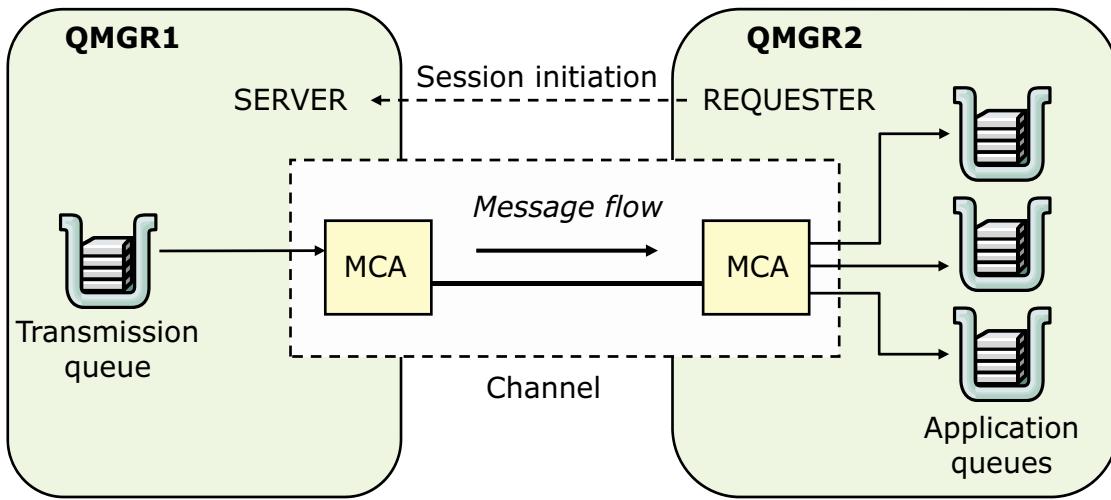
© Copyright IBM Corporation 2017

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.

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

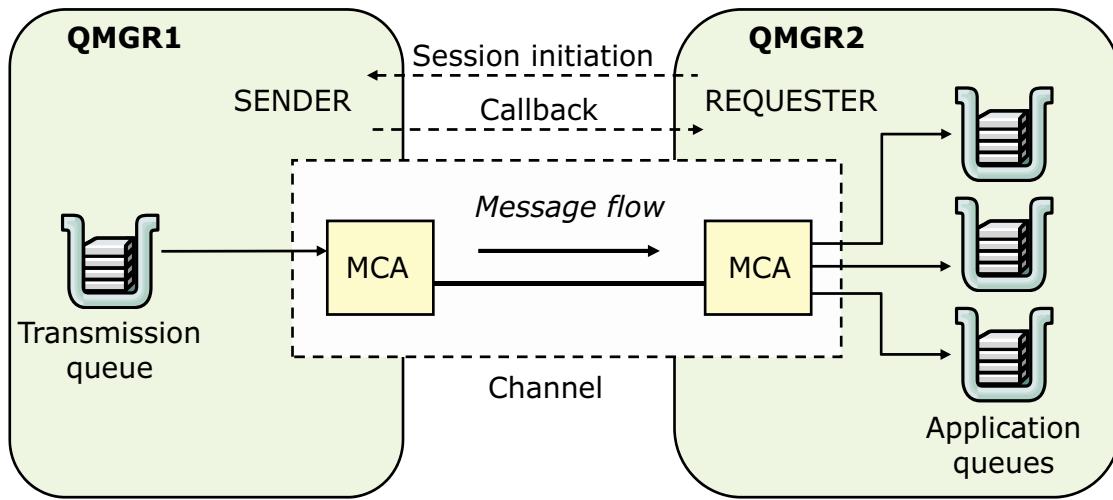
© Copyright IBM Corporation 2017

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.

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.

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
 Sender: **SDR**
 Receiver: **RCVR**
 Server: **SVR**
 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.

Defining channels example

QMGR1 on 9.20.31.5:1414

```
DEF CHL (QMGR1.QMGR2) +
CHLTYPE (SDR) TRPTYPE (TCP) +
CONNNAME ('9.20.31.5(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) +
CONNNAME ('9.84.100.1(1414)') +
XMITQ (QMGR1)
```

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

Implementing distributed queuing

© Copyright IBM Corporation 2017

Figure 6-18. Defining channels example

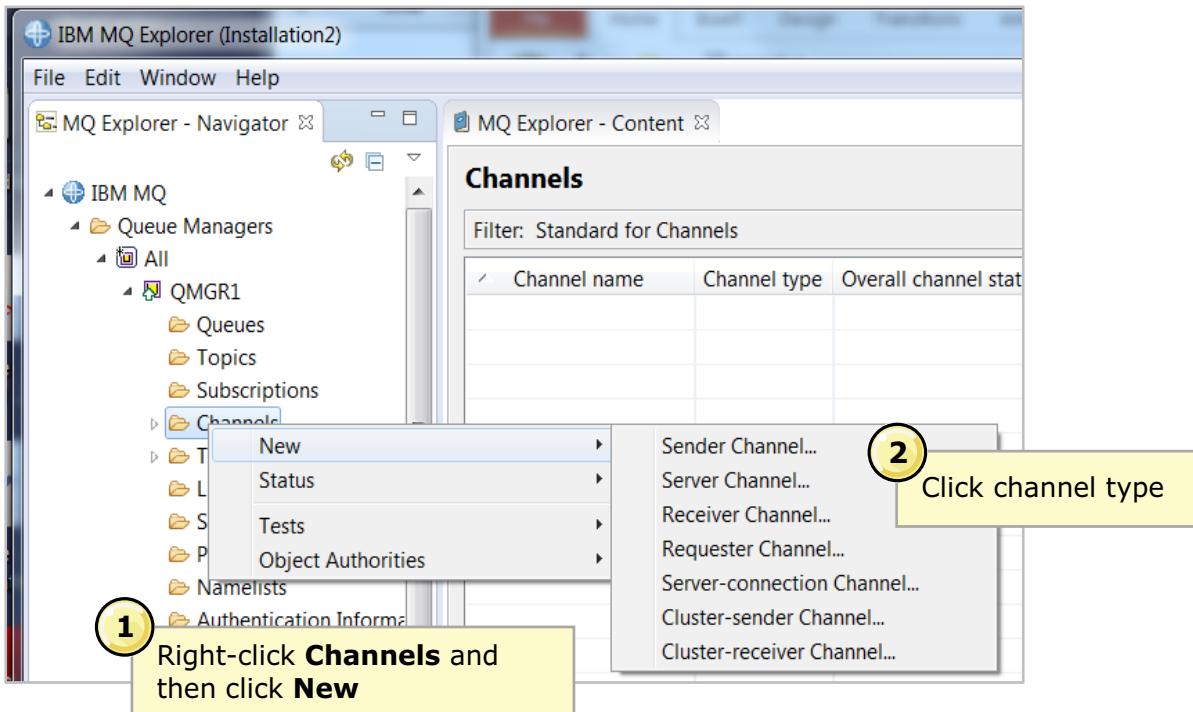
This figure shows the MQSC commands that you would enter on each queue manager to create two-way communication between the queue managers. Each queue manager has two channel definitions. QMGR1 has a sender channel to QMGR2 and receiver channel from QMGR2. QMGR2 has a receiver channel for QMGR1 and a 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 **CONNNAME** parameter. The port number is the port number of the queue manager listener, which must be running.



Defining channels in IBM MQ Explorer (1 of 2)



Implementing distributed queuing

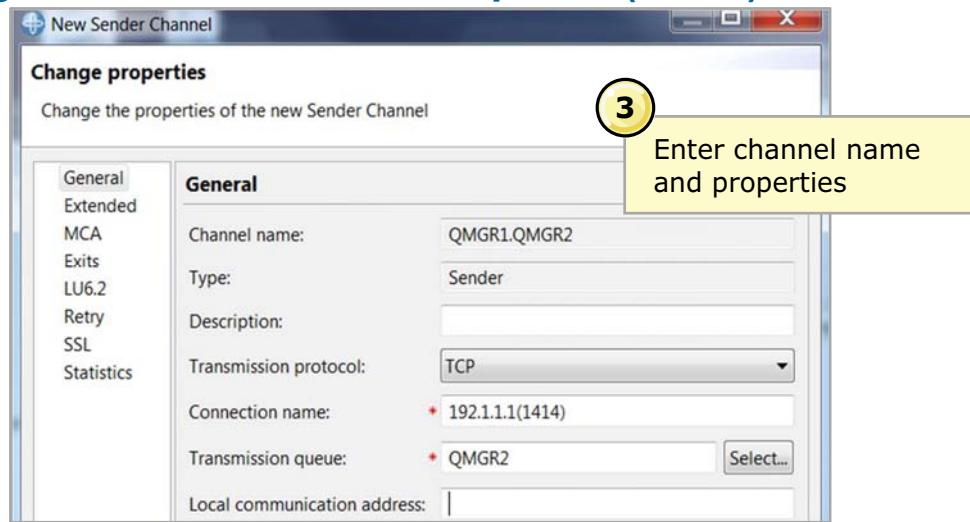
© Copyright IBM Corporation 2017

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.

Defining channels in IBM MQ Explorer (2 of 2)



- **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.

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.

Minimum channel definitions for remote communication

- On source queue manager, channel type of **SENDER**
 - **XMITQ** specifies the name of the transmission queue
 - **CONNNAME** 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 **CONNNAME** attribute)
- The communication protocol type (the **TRPTYPE** attribute)

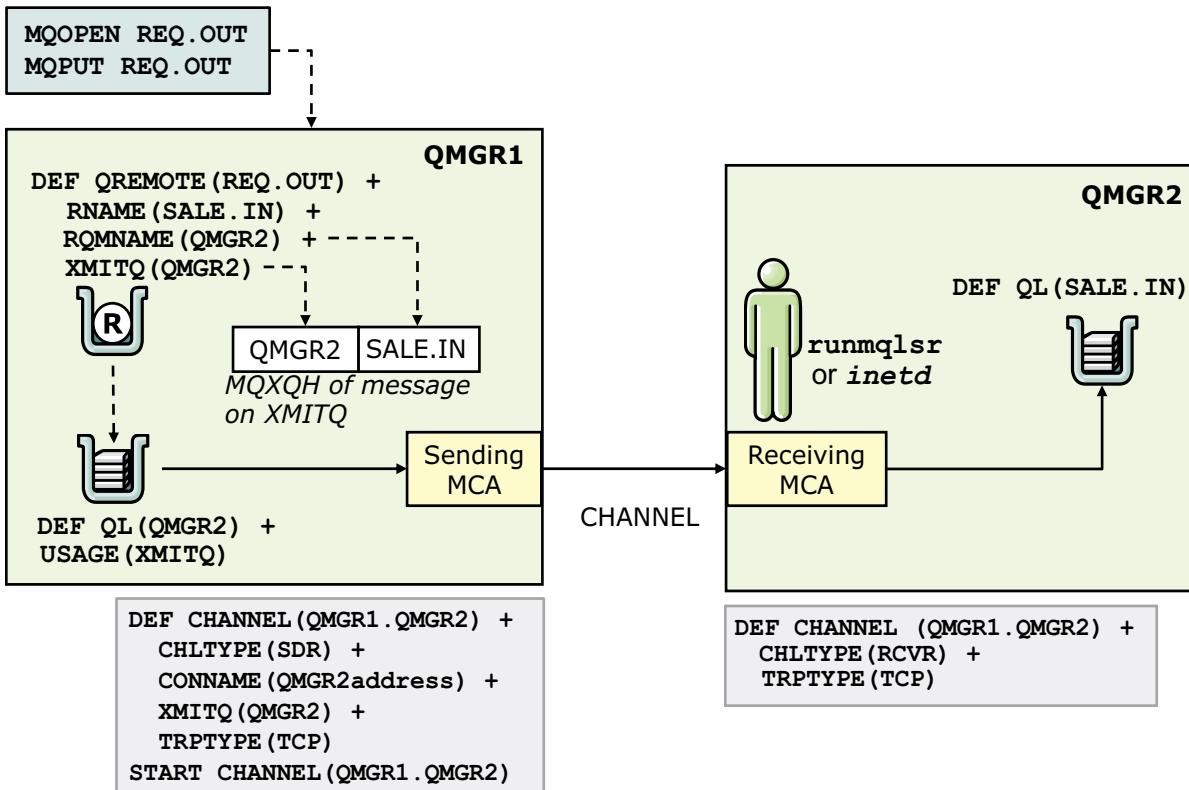
On the target queue manager, define a channel with a channel type of **RECEIVER**, and the same name as the sender channel. Also, specify the communication protocol on the channel (the **TRPTYPE** attribute).

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



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.

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:

MAXMSG Maximum message length

HBINT Heartbeat interval

NPMSSPEED 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:

- **MAXMSG** 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.

- **NPM SPEED** 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.

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.

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...)
LSTMMSGTI(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)
CONNNAME(MQ0A)
SAVED
CHLTYPE(SDR)
INDOUBT(NO)
LSTSEQNO(12)
LSTLUWID(CDB054432EB...)
CURMSGS(0)
CURSEQNO(12)
CURLUWID(CDB054432EB...)
```

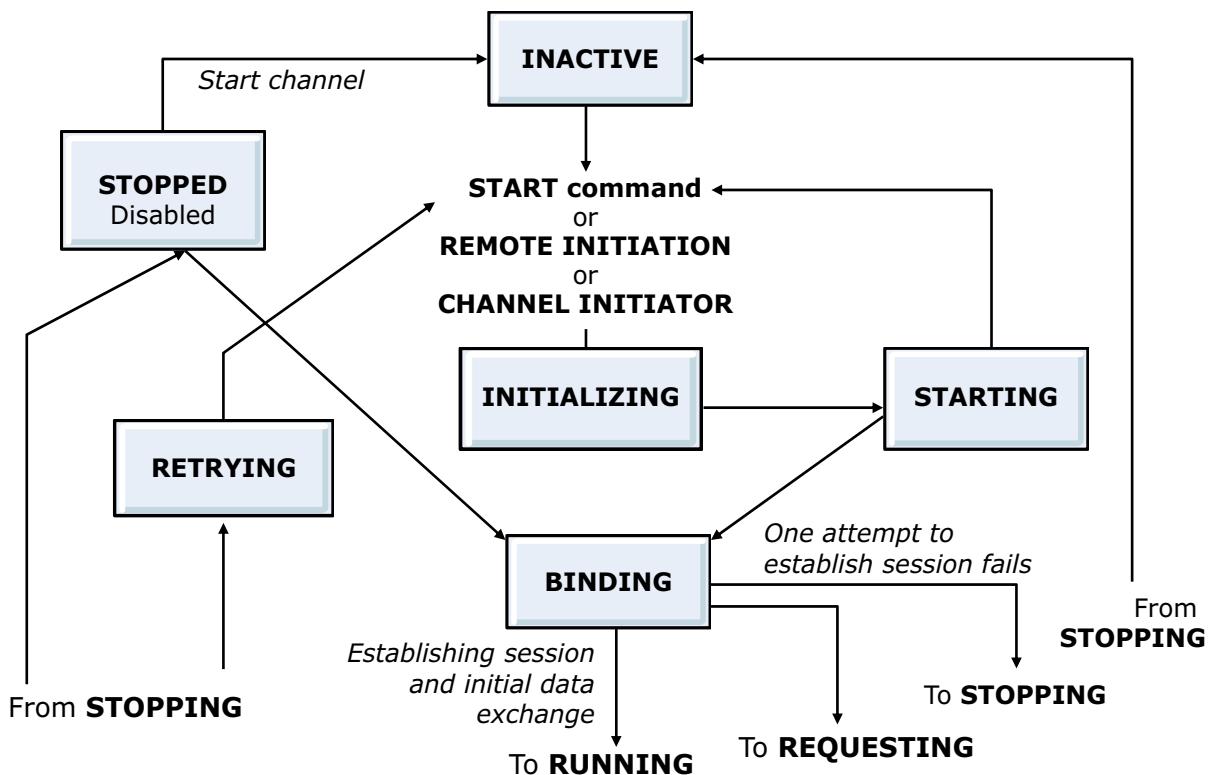
Implementing distributed queuing

© 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.

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.

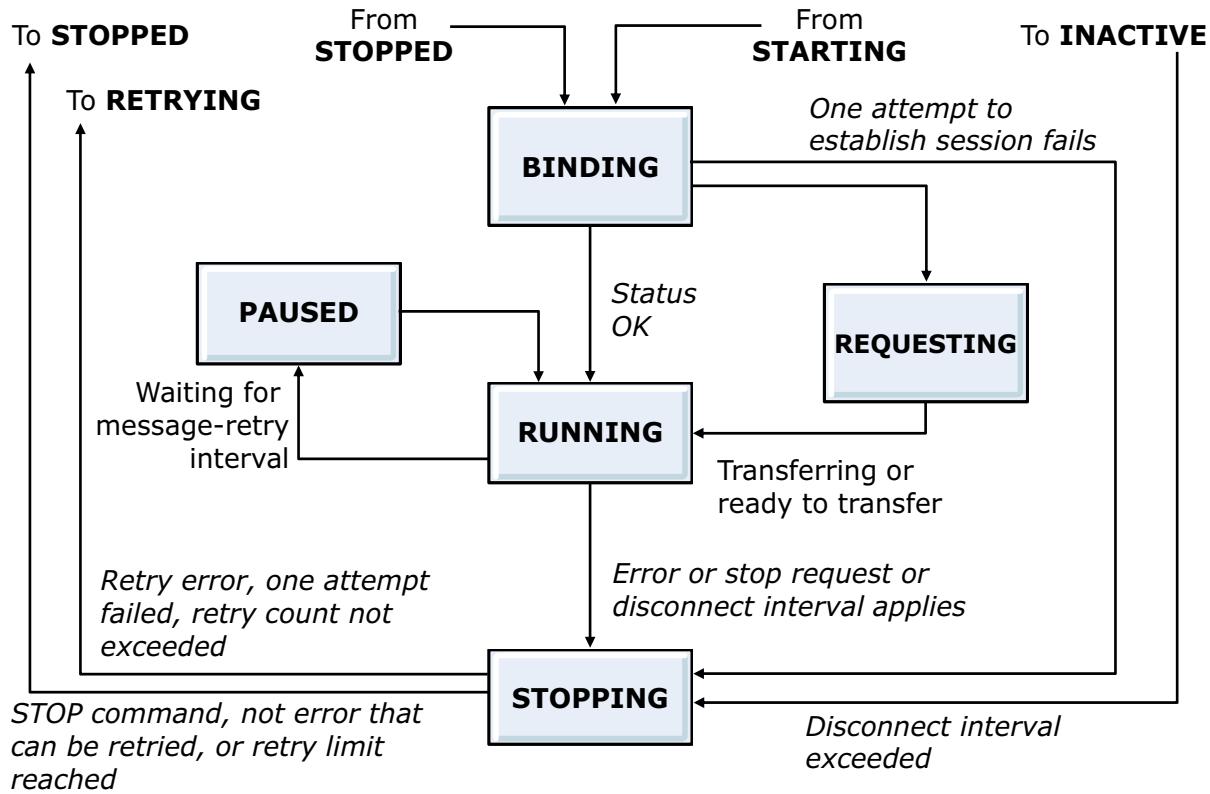
The RETRYING state indicates that the channel is waiting until it is time for the channel initiator to make the next attempt to start the channel. An error can cause a channel to go into the RETRYING state when it is possible that the problem might clear itself. Otherwise, it goes into the STOPPED state.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

Channel states (2 of 2)



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.

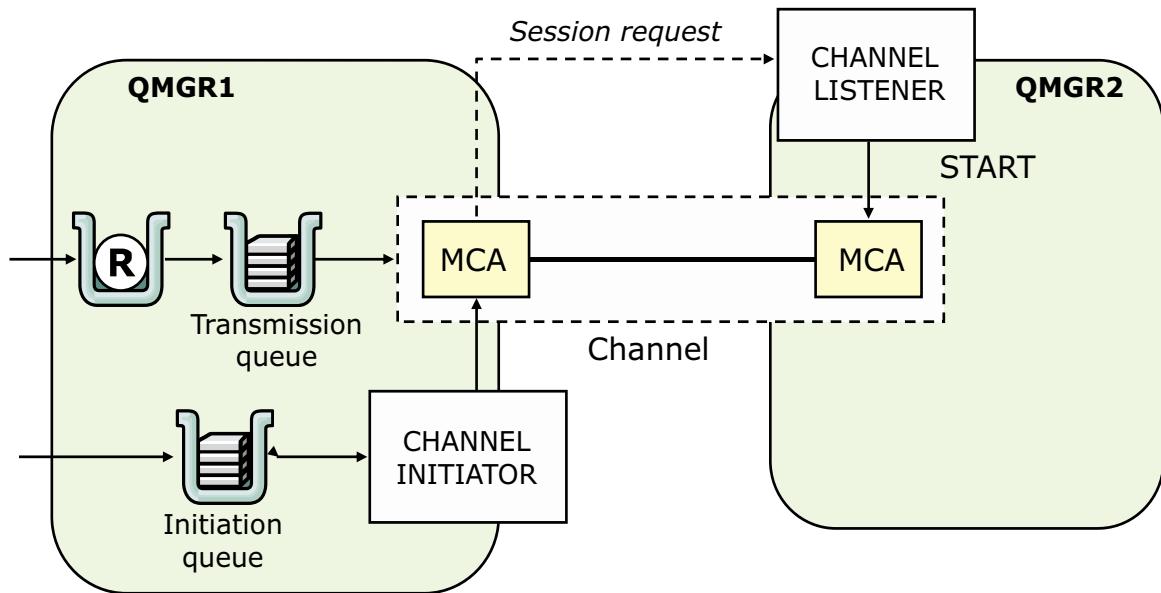
In the **REQUESTING** state, a requester is waiting for callback from a sender.

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.

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

Implementing distributed queuing

© Copyright IBM Corporation 2017

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.

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`**.

Listener object

- Create a listener object
 - Optionally, specify **CONTROL (QMGR)** 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.

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.

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.

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

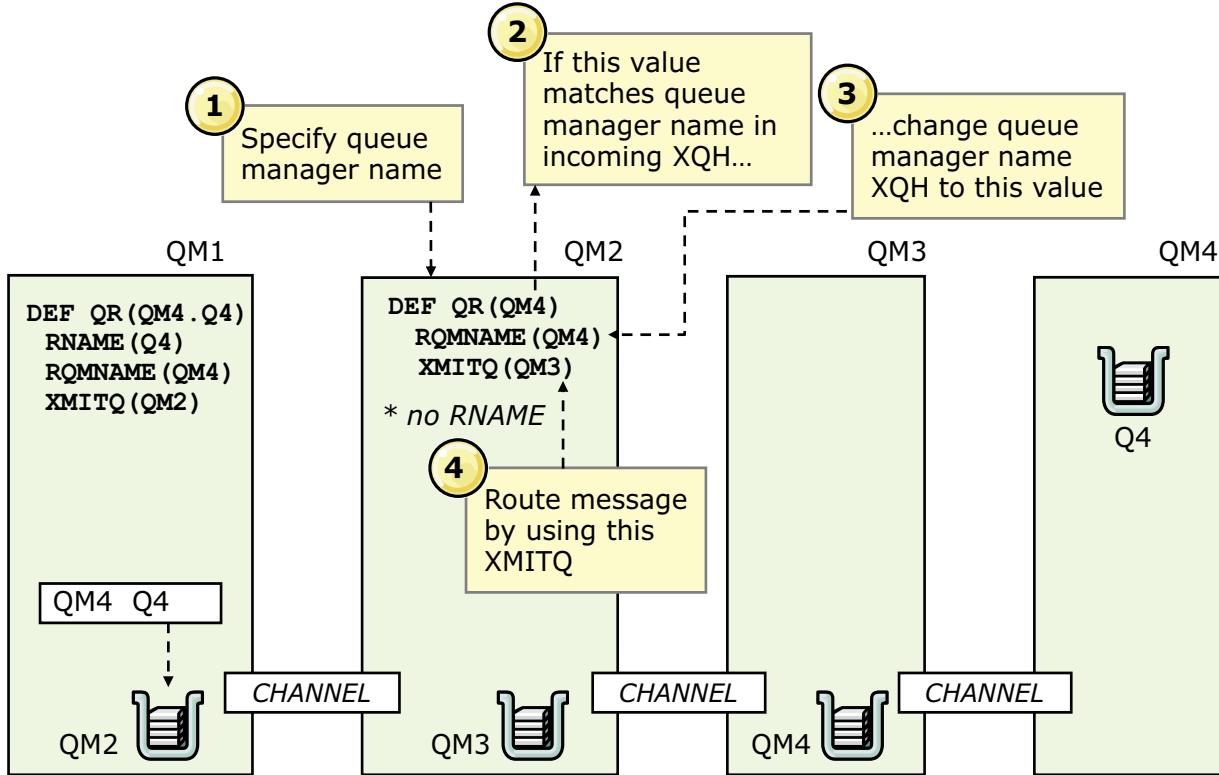
Queue manager alias definitions apply when an application that opens a queue to put a message, specifies the queue name and the queue manager name.

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.

Using a queue manager alias



Implementing distributed queuing

© Copyright IBM Corporation 2017

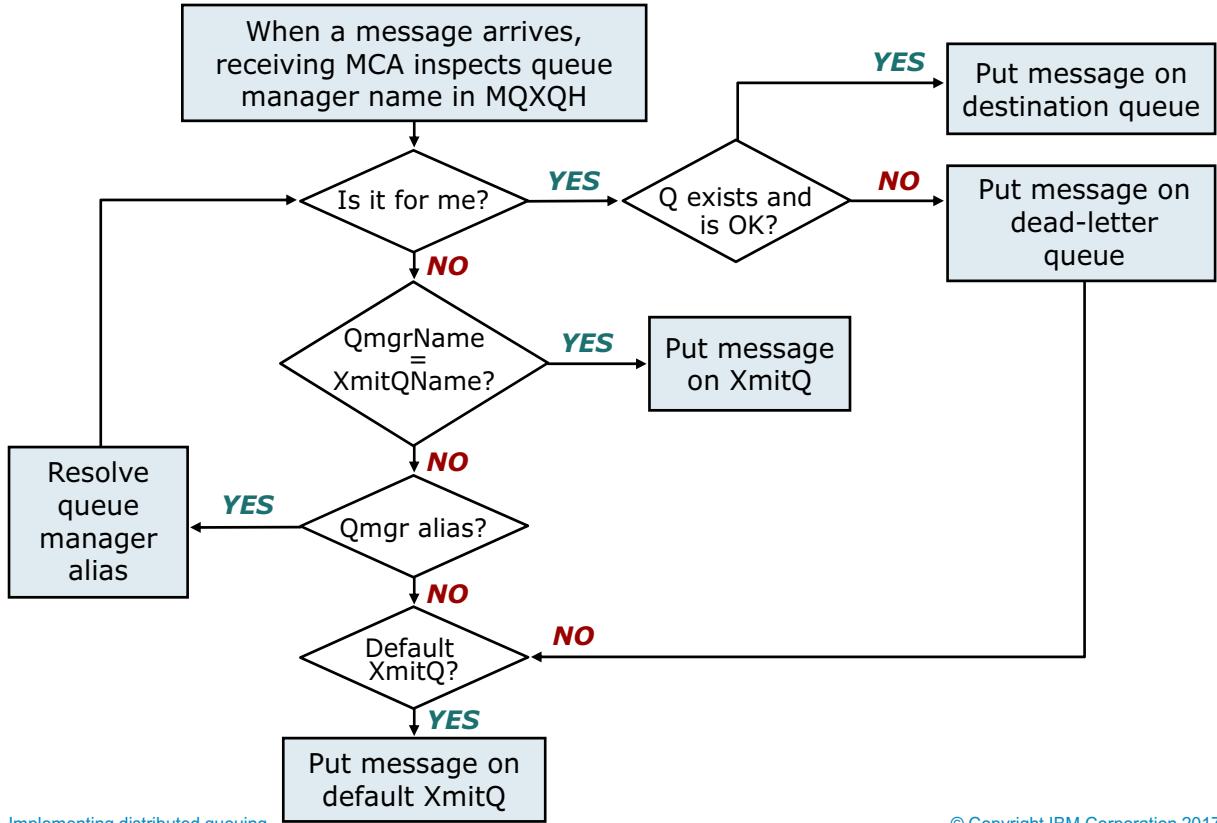
Figure 6-35. Using a queue manager alias

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.

When a message arrives at a queue manager



Implementing distributed queuing

© Copyright IBM Corporation 2017

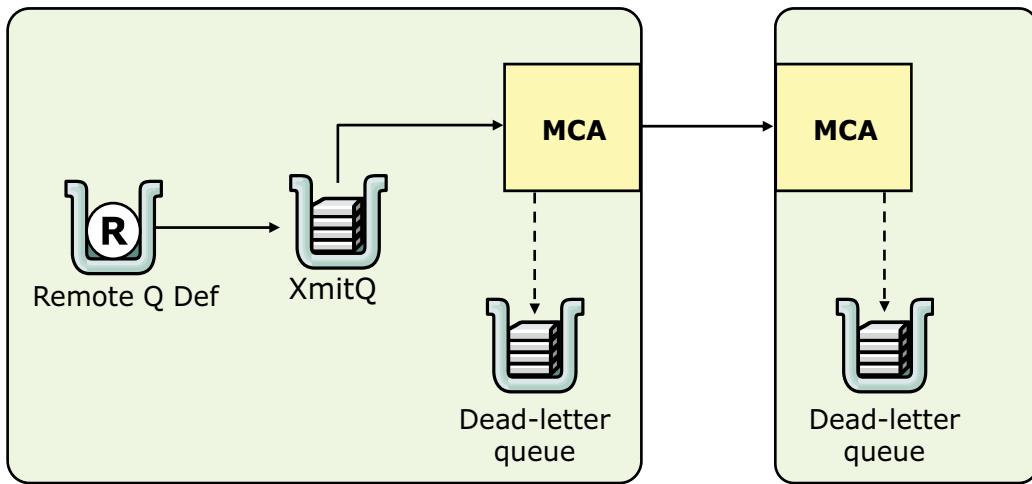
Figure 6-36. When a message arrives at a queue manager

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.

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.

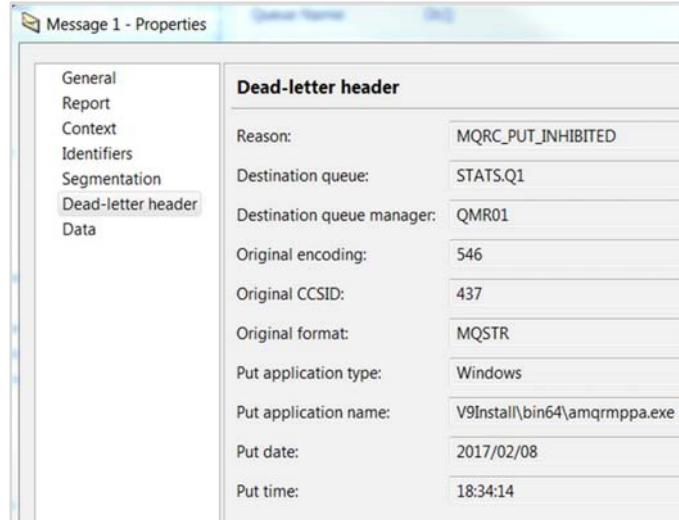
When a problem that relates to a message is detected asynchronously, an exception report is generated if one is requested, and the report is sent to the specified reply-to queue. The *Feedback* field in the message descriptor of the report message indicates the reason for the report. The original message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

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



Implementing distributed queuing

© Copyright IBM Corporation 2017

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

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.

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.

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

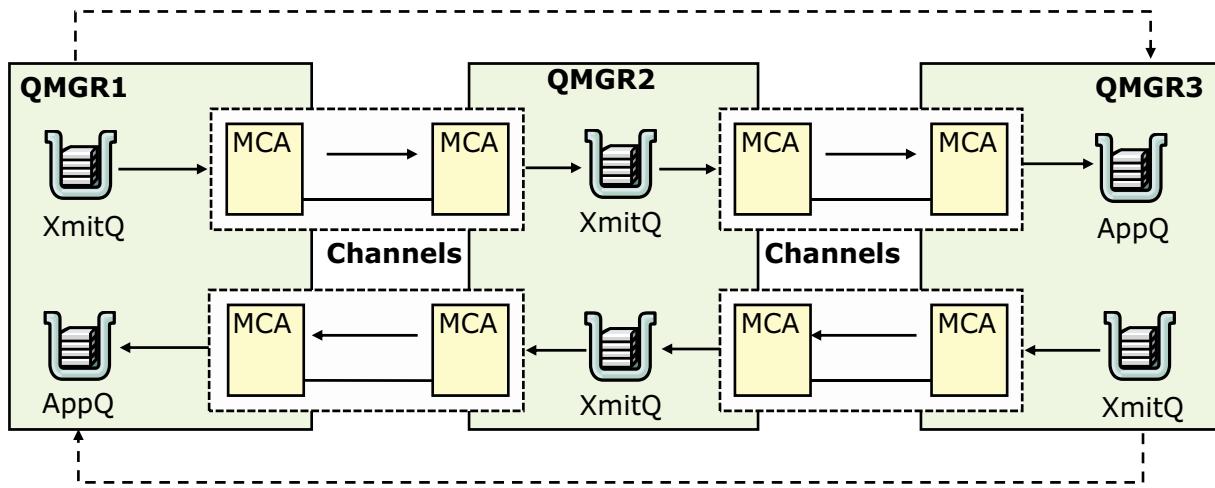
Figure 6-41. Methods for accessing a remote queue manager

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.

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*.

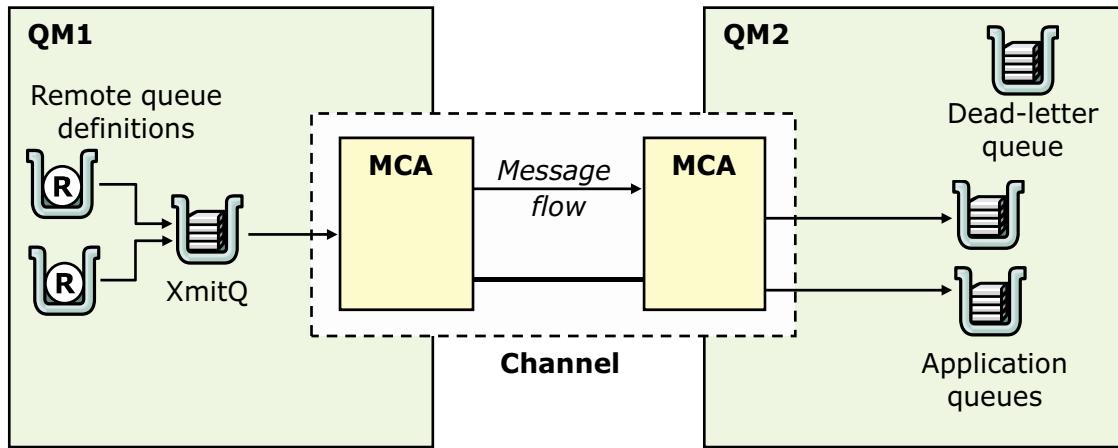
In this configuration, you must define channels between all the queue managers and transmission queues on the intermediate queue managers.

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.

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](#)

© Copyright IBM Corporation 2017

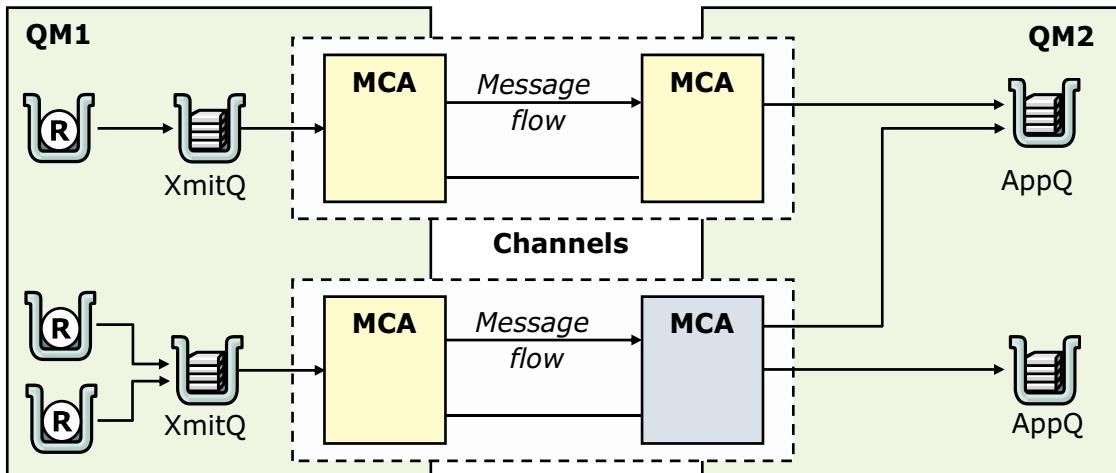
Figure 6-43. Channel sharing

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.

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.

Data representation and conversion

- Messages might require data conversion when sent between 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

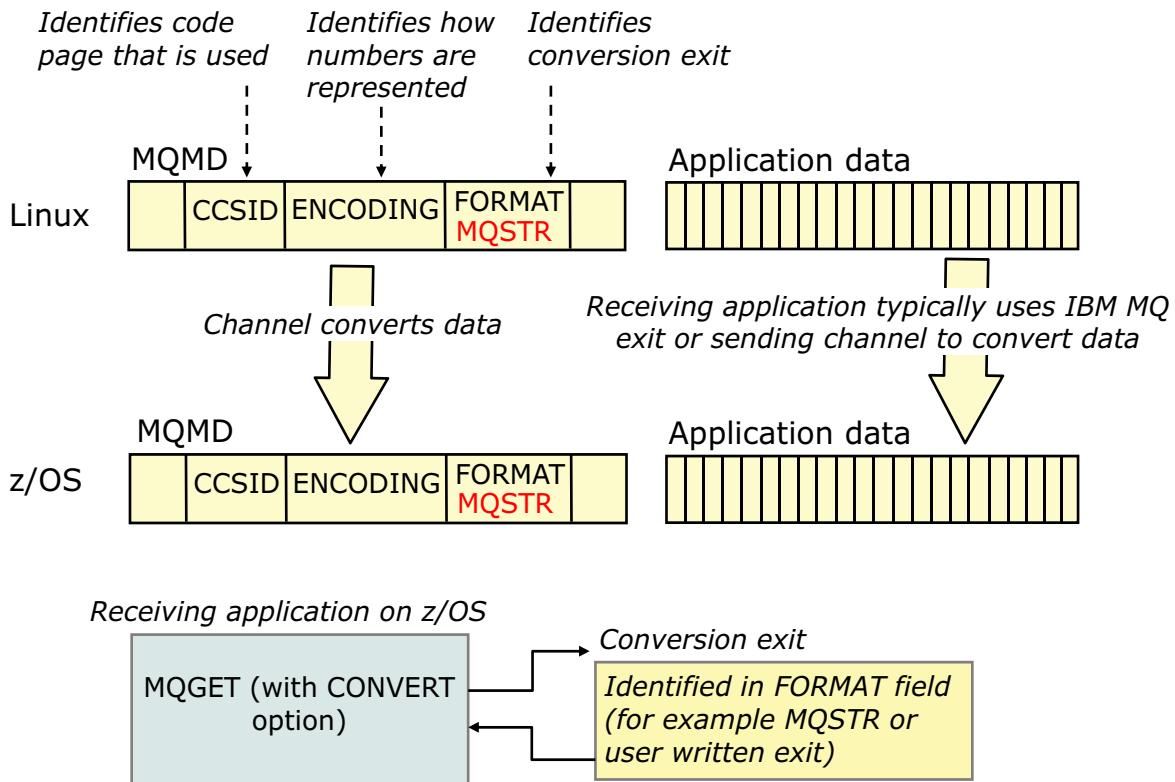
Application data might need to be converted to the character set and the encoding that another application requires when the queue managers or applications are on different operating systems and hardware.

- 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.

Data conversion example



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.

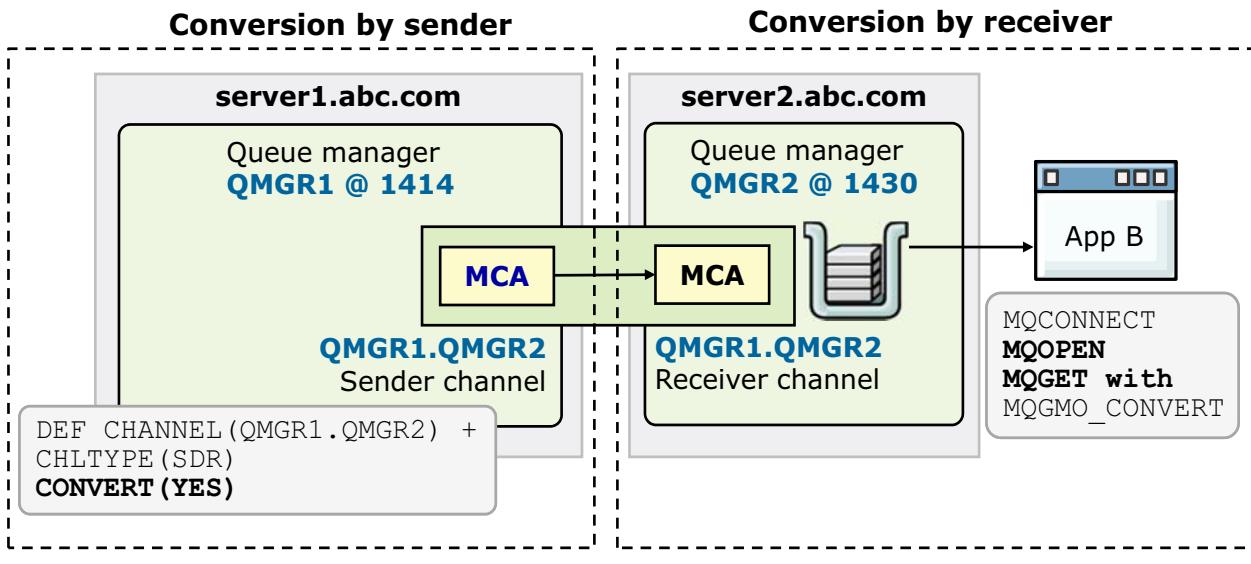
When MQ sends messages between queue managers, you must consider the message code page and encoding, especially when MQ sends data between countries and operating systems.

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.

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



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.

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.

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

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

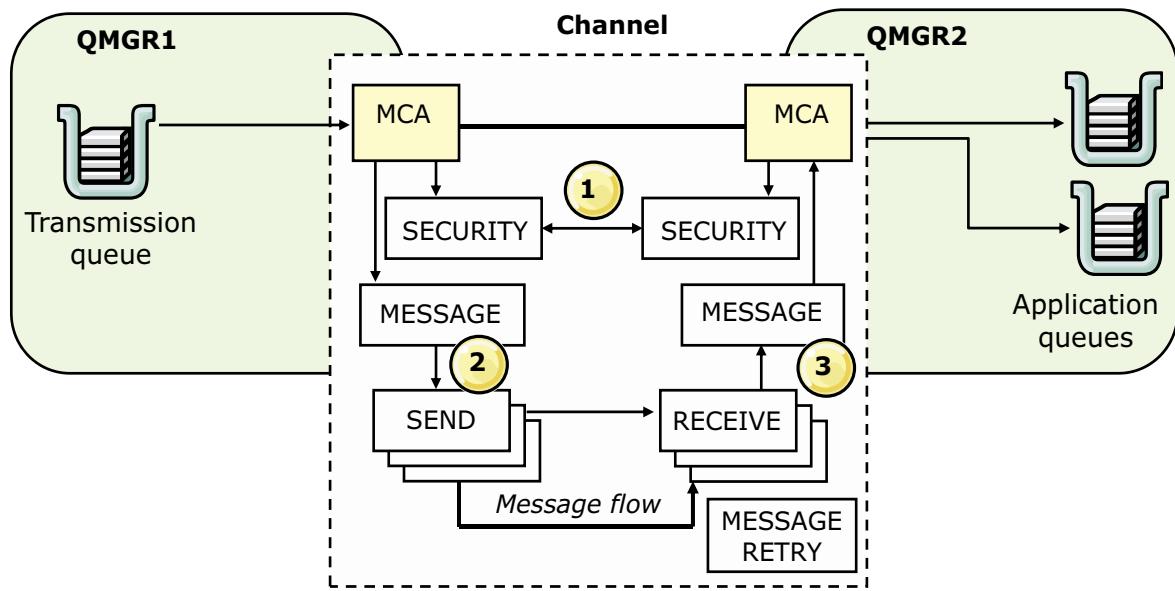
MQ calls channel-exit programs at defined places in the processing carried out by the MCA. Four types of channel exits exist:

- 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*.

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.

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

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 your answers down here:

- 1.
- 2.
- 3.

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.



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.

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.

Unit 7. IBM MQ clients

Estimated time

01:00

Overview

In this unit, you learn how IBM MQ clients can attach to an IBM MQ server with various connection methods.

How you will check your progress

- Review questions
- Hands-on exercise

References

IBM Knowledge Center for IBM MQ V9

Unit objectives

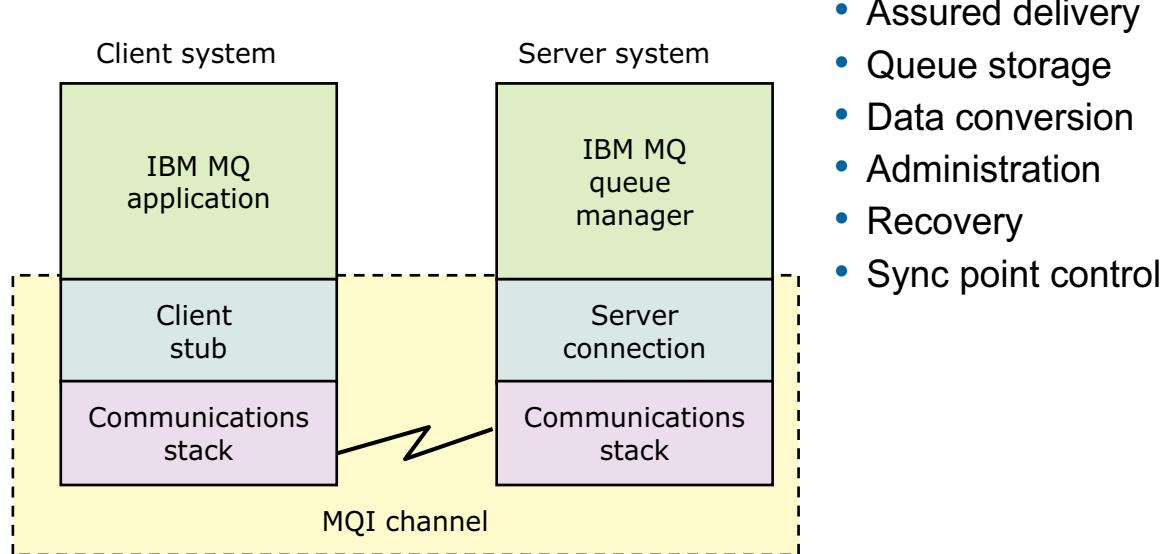
- Describe the various ways to connect a client to a queue manager
- Analyze client connection requirements
- Describe the limitations of various client connection methods
- Propose methods to ensure security with client connections

IBM MQ clients

© Copyright IBM Corporation 2017

Figure 7-1. Unit objectives

IBM MQ client



IBM MQ clients

© Copyright IBM Corporation 2017

Figure 7-2. IBM MQ client

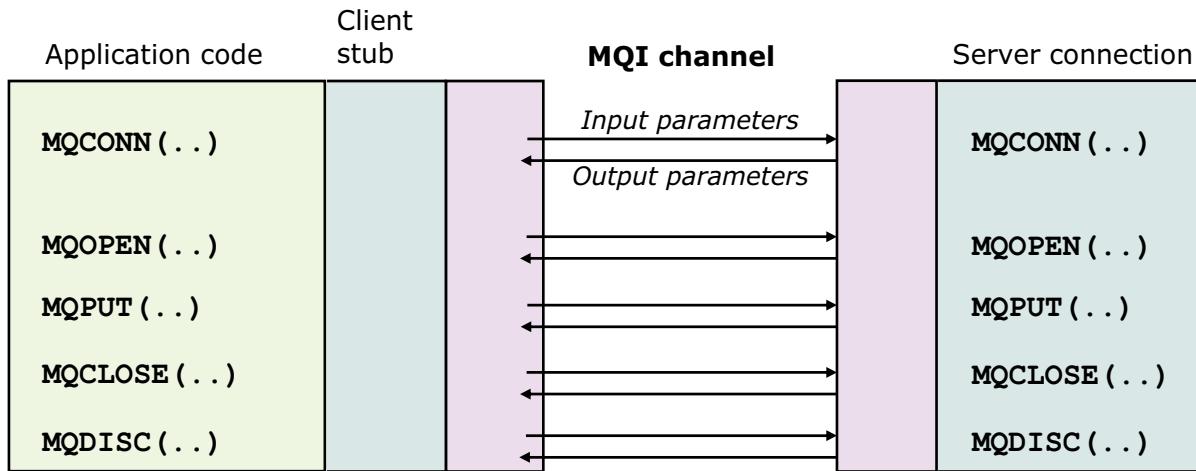
With an MQ client, an application that is running on the same computer as the client can connect to a queue manager that is running on another computer and issue MQI calls to that queue manager. This type of application is referred to as an *MQ client application* and the queue manager is referred to as the *MQ server queue manager*.

An MQ client can be installed on a computer on which no queue manager runs.

It is also possible to configure an MQ client application so that it can connect to a queue manager that is on the same computer, that is, a loopback connection. This configuration would allow a Windows application to connect to a Windows queue manager on the same system, for example.

An MQ client application and a server queue manager communicate with each other by using an MQI channel.

IBM MQ client communication



- MQI channel starts when client application sends an MQCONN or MQCONNX call to connect to server queue manager
- MQI channel ends when client application sends MQDISC call to disconnect from server queue manager
- Input parameters of MQI call flow in one direction on MQI channel, and output parameters flow in opposite direction

IBM MQ clients

© Copyright IBM Corporation 2017

Figure 7-3. IBM MQ client communication

An MQI channel starts when the client application sends an MQCONN or MQCONNX call to connect to the server queue manager. It ends when the client application sends an MQDISC call to disconnect from the server queue manager.

The input parameters of an MQI call flow in one direction on an MQI channel and the output parameters flow in the opposite direction.

A client application can be connected to more than one queue manager server simultaneously. Each MQCONN call to a different queue manager returns a different connection handle. This behavior does not apply if the application is not running as an MQ client.

The MQI stub that is linked with an application that runs as a client is different from the one that is used when the application is not running as a client. An application receives the reason code MQRC_Q_MGR_NOT_AVAILABLE on an MQCONN call if it is linked with the wrong MQI stub.

If a communications failure exists, some delay might be involved in servicing an MQI call. Any failure is reported to the application with a reason code.

Although the full range of MQI calls and options are available to an application that runs as an MQ client, some restrictions might apply to system resources in certain environments, for example, memory constraints.

Units of work and sync point

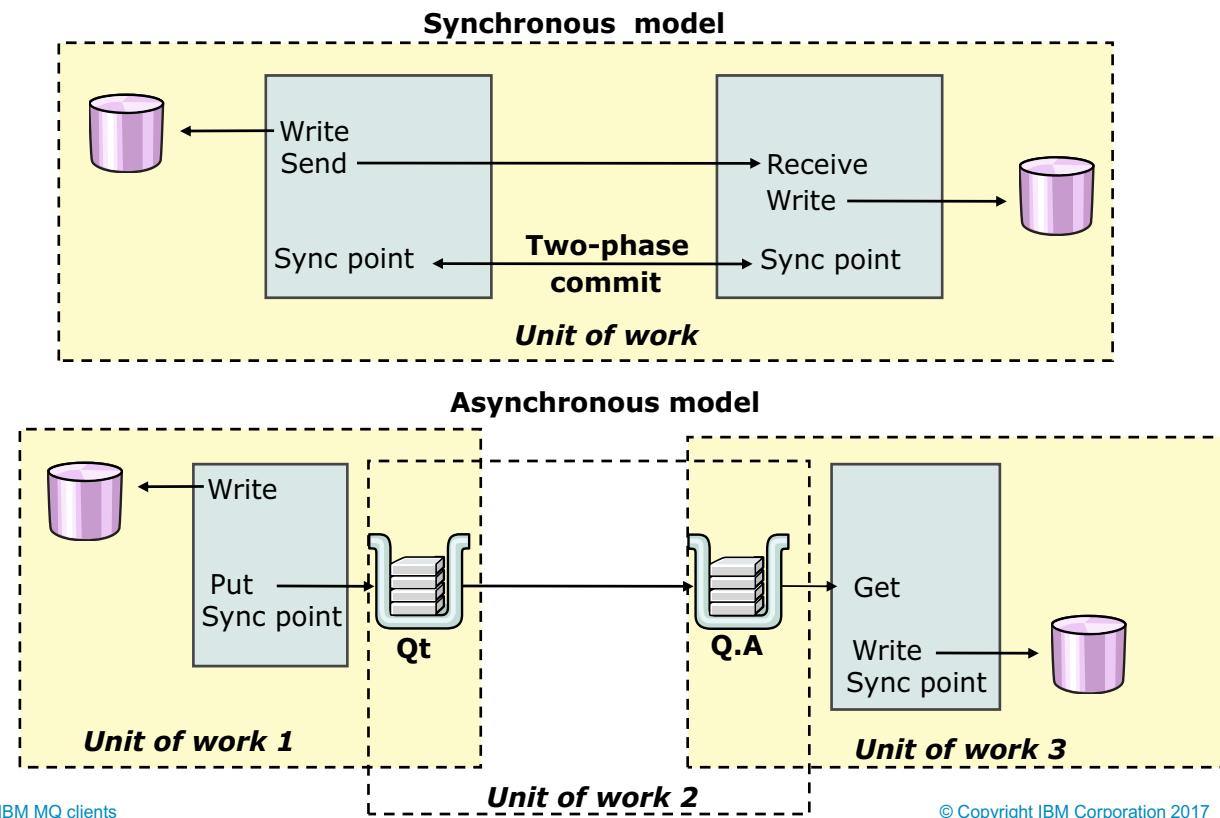


Figure 7-4. Units of work and sync point

Some implementations of the conversational style of program-to-program communication use a two-phase commit protocol to support the implementation of a distributed unit of work. However, this type of function is necessary only when the business has an absolute requirement to always maintain two or more distributed databases in step, down to the last fraction of a second. Such a requirement is encountered only rarely in practice. If the requirement does not really exist, a single, distributed unit of work can be resource-intensive and complex, particularly if many processes are involved. In this case, MQ offers a simple solution that involves multiple units of work that act asynchronously.

If the business transaction is a more complex one, many units of work can be involved. It is the assured delivery property of MQ that ensures the integrity of the complete business transaction.

The importance of the assured, one-time delivery property of MQ in this design can clearly be seen. This property, with the ability to change MQ resources as part of a unit work, means that the separate processes can operate safely and independently without holding complex locks across a network.

Implementing a business transaction as a single distributed unit of work requires a two-phase commit protocol. The MQ implementation uses multiple units of work that act asynchronously, as shown in the lower flow in the figure.

In the first unit of work, the application writes to a database, puts a message on a queue, and then sends a sync point to commit the changes to the two resources. Because the queue is a remote queue, the message gets no further than the transmission queue within this unit of work. When the unit of work is committed, the message becomes available for retrieval by the sending MCA.

In the second unit of work, the sending MCA gets the message from the transmission queue and sends it to the receiving MCA on the system that contains the second database. The receiving MCA puts the message on the destination queue. When this unit of work is committed, the message becomes available for retrieval by the second application.

In the third unit of work, the second application uses the data that is contained in the message to get the message from the destination queue and updates the database.

Units of work and sync point control

- **Local** unit of work is one in which the only resources that are updated are the resources of the queue manager
- **Global** unit of work is one in which the resources of other resource managers are also being updated

- Application can specify sync point on MQPUT and MQGET calls
 - Message added or removed immediately (**NO_SYNCPOINT**)
 - Result of MQPUT or MQGET is visible only when unit of work is committed (**SYNCPOINT**)
 - Gets message within sync point control only if it is persistent on MQGET (**SYNCPOINT_IF_PERSISTENT**)
 - Default depends on the operating system

Figure 7-5. Units of work and sync point control

A local unit of work is one in which the resources that are updated are those resources of the queue manager to which the application is connected.

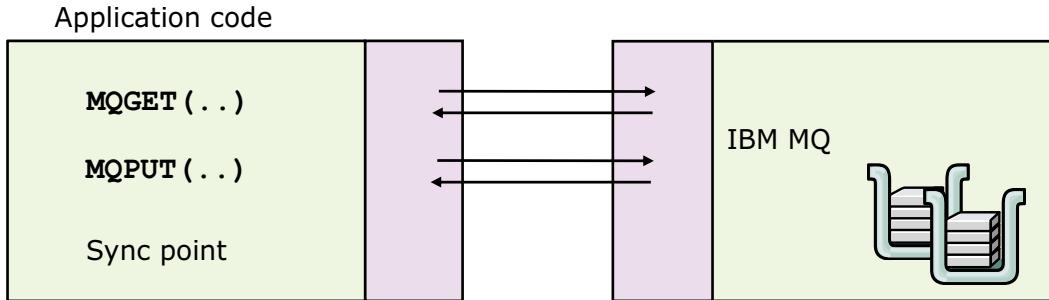
A global unit of work is one in which the resources of other resource managers and the resources of a queue manager are updated. The coordination of global units of work can be internal or external to the queue manager.

A message that is put on a queue within sync point control is not available for another application to get until the unit of work is committed. Similarly, a message got from a queue within sync point control does not cause the message to be removed from the queue until the unit of work is committed.

An application can specify whether an MQPUT call or an MQGET call is within, or outside of, sync point control or leave it as the default. The default option depends on the operating system.

- Using the NO_SYNCPOINT option, a message is added to a queue or removed from a queue immediately.
- Using the SYNCPOINT option, the result of an MQPUT or MQGET call takes effect when the unit of work is committed.
- The SYNCPOINT_IF_PERSISTENT option on an MQGET call causes the request to operate within the normal unit-of-work protocols if the message retrieved is persistent.

Sync point control on a base client



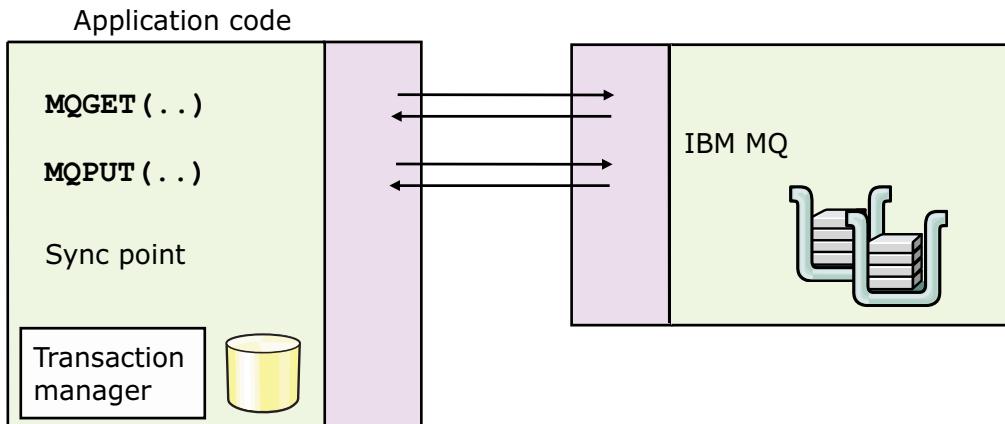
- IBM MQ client application:
 - Can participate in a local unit of work that uses IBM MQ resources
 - Cannot participate in a global unit of work that uses IBM MQ resources

Figure 7-6. Sync point control on a base client

An MQ client application can participate in a local unit of work that uses the resources of the queue manager it is connected to.

An MQ client application cannot participate in a global unit of work. An application can send MQI calls to another resource manager or sync point coordinator, on the same system as the application or on another system. It can participate in a unit of work that is associated with that resource manager or sync point coordinator. At the same time, it can also be participating in a local unit of work that requires the resources of the queue manager it is connected to. In this case, the two units of work are independent of each other.

Extended transactional client



- Can update resources that another resource manager manages under the control of an external transaction manager
 - XA transactional client is supplied with IBM MQ
 - Transaction manager runs on client system
 - Transaction manager provides sync point processing

IBM MQ clients

© Copyright IBM Corporation 2017

Figure 7-7. Extended transactional client

An *MQ extended transactional client* is an MQ MQI client with some additional features.

An MQ extended transactional client application, within the same unit of work, can complete the following tasks:

- Put messages to, and get messages from, server queue manager queues
- Update the resources of a resource manager other than an MQ queue manager

An external transaction manager that is running on the same system as the client application must manage this unit of work. The queue manager to which the client application is connected cannot manage the unit of work; the queue manager can act only as a resource manager, not as a transaction manager. The client application can commit or back out the unit of work by using only the API provided by the external transaction manager. Therefore, the client application cannot use the MQI calls **MQBEGIN**, **MQCMBIT**, and **MQBACK**.

The external transaction manager communicates with the queue manager as a resource manager with the same MQI channel as the client application that is connected to the queue manager. In a recovery after a failure, the transaction manager can use a dedicated MQI channel to recover any incomplete units of work in which the queue manager was participating at the time of the failure. The recovery requires that all applications are stopped.

Configuring connections between the server and client

- Decide on your communication protocol based on the client and server operating system
 - TCP/IP
 - LU 6.2
 - NetBIOS (Windows only)
 - SPX (Windows only)
- Define server-connection channel and client-connection channel that use the same channel name and compatible channel types
 - Option 1: Create one channel definition on the IBM MQ client and the other on the server
 - Option 2: Create both channel definitions on the server and then make the client-connection definition available to the client

Figure 7-8. Configuring connections between the server and client

To configure the communication links between MQ MQI clients and servers, you must select the communication protocol and define the channels for both ends of the link.

Defining an MQI channel

- Use **DEFINE CHANNEL** command with parameters:

CHLTYPE	CLNTCONN or SVRCONN
TRPTYPE	LU62 , NETBIOS , SPX , or TCP
CONNNAME (<i>string</i>)	For client connection only
QMNAME (<i>string</i>)	For client connection only

- No operational involvement on an MQI channel
 - MQI channel starts when a client application sends **MQCONN** (or **MQCONNX**)
 - MQI channel stops when a client application sends **MQDISC**

Figure 7-9. Defining an MQI channel

Similar to a message channel, an MQI channel requires a definition at both ends of the channel, and each end of an MQI channel has a type.

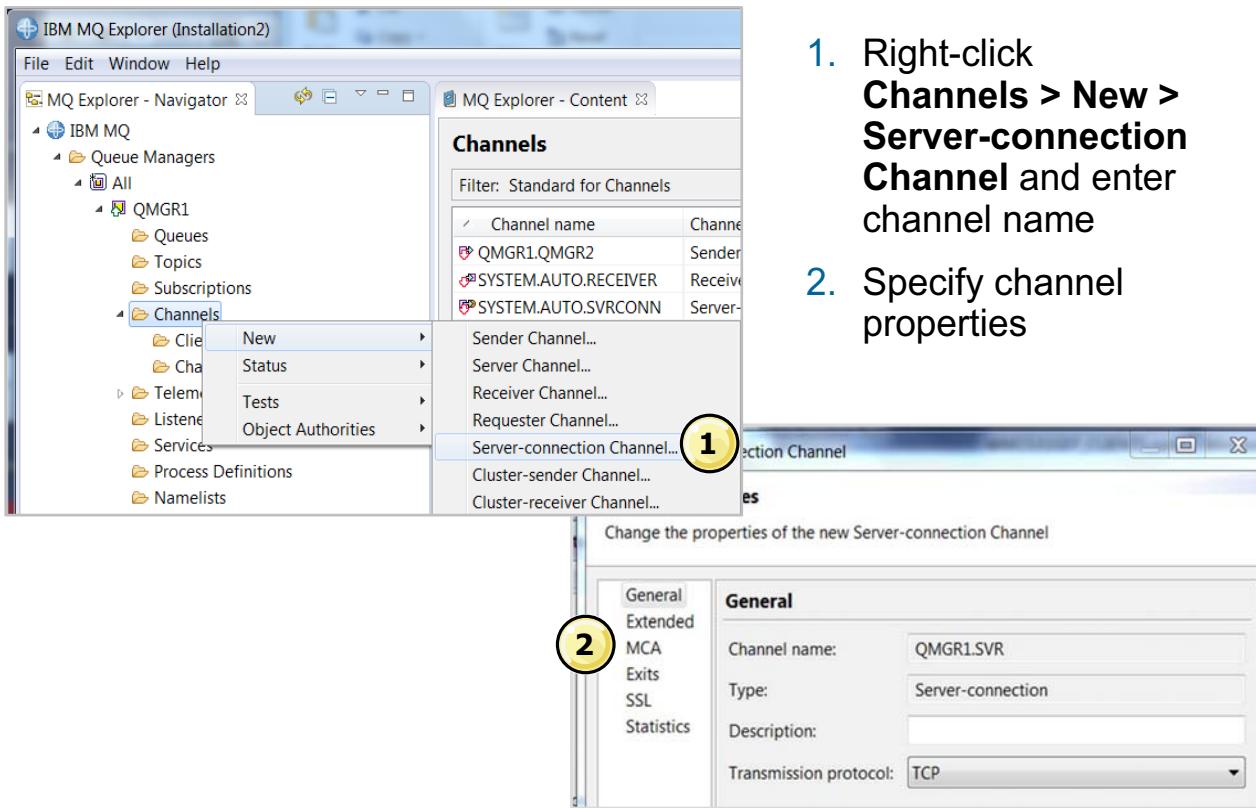
- The client connection (**CLNTCONN**) is for the end of the MQI channel on the client system.
- The server connection (**SVRCONN**) is for the end of the MQI channel on the server system. The MQ listener must be running on the server system.

An MQI channel is bidirectional in terms of the flow of information, that is, the input parameters of an MQI call flow in one direction and the output parameters in the reverse direction. It is not necessary to define two channels, one for each direction.

An MQI channel starts when a client application connects by using an **MQCONN** or **MQCONNX** call. The MQI channel stops when the client application disconnects by using the **MQDISC** call.

An MQI channel can be used to connect a client to a single queue manager, or to a queue manager that is part of a queue-sharing group.

Defining server-connection channels in IBM MQ Explorer



IBM MQ clients

© Copyright IBM Corporation 2017

Figure 7-10. Defining server-connection channels in IBM MQ Explorer

You can define a server-connection channel by using the `DEFINE CHANNEL` command or MQ Explorer. This figure shows the steps for defining a server-connection channel.

Client configuration methods

- Method 1: MQSERVER environment variable to specify a simple definition of a client-connection channel
- Method 2: Configure clients by using attributes in a client configuration file on the client
- Method 3: Use client channel definition table (CCDT) to determine channel definitions and authentication information that client applications use
- Method 4: Encapsulated connection object in Java (application code)
- Method 5: External JNDI lookup for JMS (application code)

IBM MQ clients

© Copyright IBM Corporation 2017

Figure 7-11. Client configuration methods

The figure lists the ways to configure clients connections:

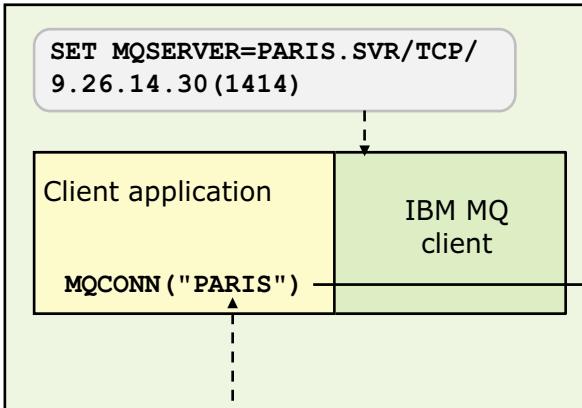
1. Set the MQSERVER environment variable on the client.
2. Modify the client configuration file (`mqclient.ini`).
3. Create a client channel definition table.
4. Use an encapsulated connection object in Java.
5. Create an external JNDI lookup for JMS.

The first three methods are relevant from an administration point of view. These methods are examined in more detail on the next pages.

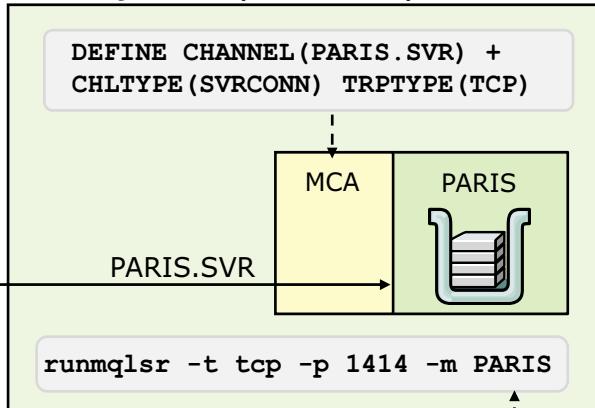
The last two methods require user-defined application code.

Method 1: Using MQSERVER

Windows client



IBM MQ server (9.26.14.30)



----- Must be the same name -----

- Only channel available to the application
- Connection name can be a comma-separated list of connections
- Listener program on IBM MQ server determines the queue manager to which the application connects

IBM MQ clients

© Copyright IBM Corporation 2017

Figure 7-12. Method 1: Using MQSERVER

You can use the MQSERVER environment variable to specify a simple definition of a client-connection channel.

On the client system, you define a simple client connection by setting the environment variable MQSERVER to the following value: *ChannelName/TransportType/ConnectionName*

The **ConnectionName** portion of the environment variable can be a comma-separated list of connections.

The syntax for setting an environment variable varies by operating system. On Windows, you use the **SET** command. On UNIX and Linux, you use the **export** command.

Examples:

- On Windows, type: **SET MQSERVER=PARIS.SVR/TCP/9.26.14.30(1414)**
- On UNIX or Linux, type: **export MQSERVER='PARIS.SVR/TCP/9.26.14.30(1414)'**

On the server in the example, you use the MQSC **DEFINE CHANNEL** command to define a server connection (SVRCONN) and the **runmqqlsr** command to start the listener.

As shown in the example, the connection name in the **MQCONN** call from the application must be the same as the queue manager name in the **runmqqlsr** command.

Method 2: Client configuration file

- Configure clients by using attributes in the `mqclient.ini` text file
 - Default location on UNIX and Linux is `/var/mqm`
 - Default location on Windows is `C:\ProgramData\IBM\MQ`
 - Can specify location in `MQCLNTCF` environment variable
- Configuration applies to all connections that a client application makes to any queue managers
- Environment variables can override attributes in `mqclient.ini` file

Example:

```
## Module Name: mqclient.ini
## Type : IBM MQ MQI client configuration file
# Function : Define the configuration of a client
#####
ClientExitPath:
    ExitsDefaultPath=/var/mqm/exits
    ExitsDefaultPath64=/var/mqm/exits64

CHANNELS:
    DefRecon=YES
    ServerConnectionParms=SALES.SVRCONN/TCP/hostname.x.com(1414)
```

IBM MQ clients

© Copyright IBM Corporation 2017

Figure 7-13. Method 2: Client configuration file

Using the client configuration file method, you configure the MQI clients by using a text file, similar to the queue manager configuration file. The file contains stanzas, each of which contains lines of the format **attribute-name=value**

The **CHANNELS** stanza in the client configuration file specifies information about client channels.

- The **DefRecon** attribute provides an administrative option to enable client programs to automatically reconnect, or to disable the automatic reconnection of a client program that is written to reconnect automatically.
- The **ServerConnectionParms** attribute is equivalent to the `MQSERVER` environment variable. It specifies the location of the MQ server and the communication method. This attribute defines only a simple channel; you cannot use it to define an SSL channel or a channel with channel exits. Similar to the `MQSERVER` environment variable, the string of the format is `ChannelName/TransportType/ConnectionName`. The `ConnectionName` portion must be a fully qualified network name.

This method has an advantage over the previous method: the configuration features apply to all connections a client application makes to any queue manager, rather than being specific to an individual connection to a queue manager.

Method 3: Using a CCDT

- CCDT files allow IBM MQ client applications to specify one or more client channel definitions to use when connecting to a queue manager
 - Can offer choice of queue managers
 - Supports workload balancing and configuration of advanced connections
- Options for creating a CCDT
 - Option 1: Create the CCDT on the queue manager by defining the client-connection channel on the queue manager
 - Option 2: Create the CCDT on the client by using `runmqsc -n` and then define client-connection channel by using the `DEFINE CHANNEL` command
- Options for specifying location of CCDT file
 - `MQCHLLIB` and `MQCHLTAB` environment variables
 - “ChannelDefinitionDirectory” attribute in `mqclient.ini` file
 - URL locations that are specified by `MQCHLLIB` or `MQCCDTURL` environment variables or programmatically by the application

Figure 7-14. Method 3: Using a CCDT

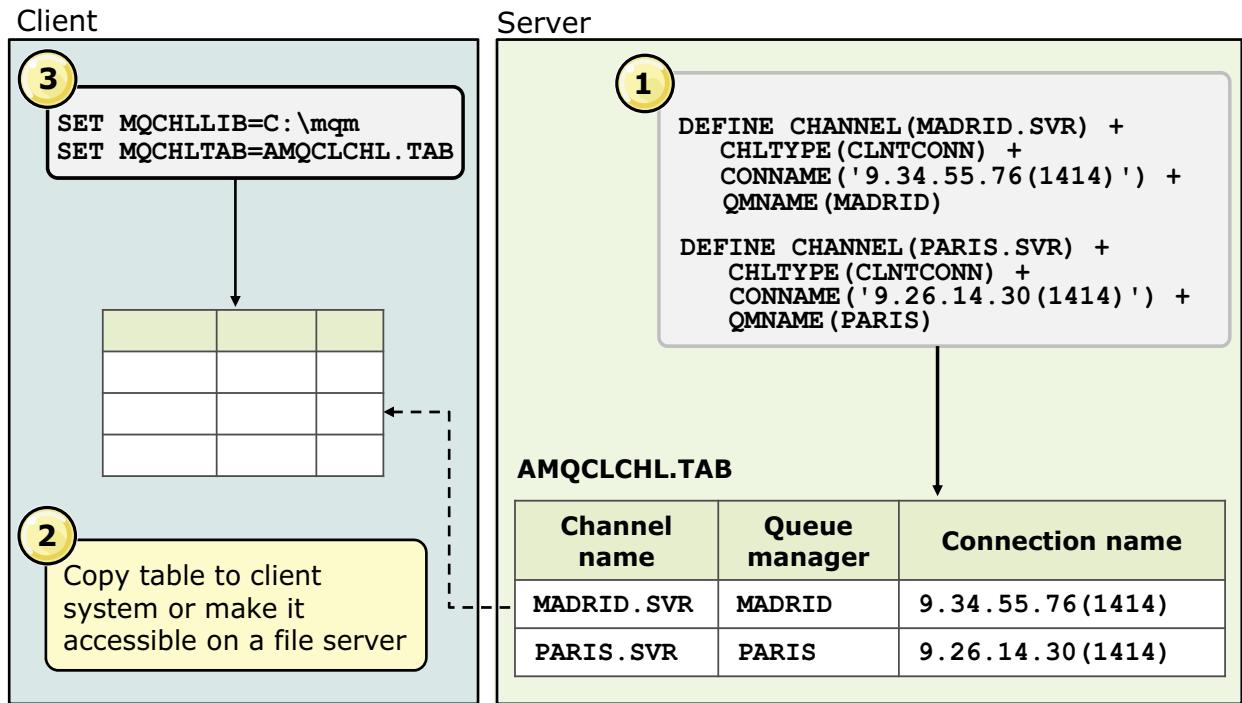
Method 3 is a more complex method that uses the definition of a client connection and a server connection on the server system.

When you define a client connection on the server, it is stored in the client channel definition table (CCDT) in the file name AMQCLCHL.TAB. One table can contain client connection information for many queues managers, which gives the application a choice of client connections and queue managers.

The CCDT file must be accessible from the client system by using one of the following methods.

- Copy the CCDT file to the client system or put it on a file server that the client can access. Then, specify the path and file name of the CCDT in the `MQCHLLIB` and `MQCHLTAB` environment variables.
- Copy the CCDT file to the client system or put it on a file server that the client can access. Then, specify the path to the CCDT file in the MQ client file.
- Locate a CCDT through a URL, either by programming, by using environment variables, or by using `mqclient.ini` file stanzas.

Using a CCDT example



IBM MQ clients

© Copyright IBM Corporation 2017

Figure 7-15. Using a CCDT example

In this example, the CCDT table is copied from the server to the local file system of the client.

On the client system, the environment variables `MQCHLLIB` and `MQCHLTAB` specify the path and the name of the CCDT file.

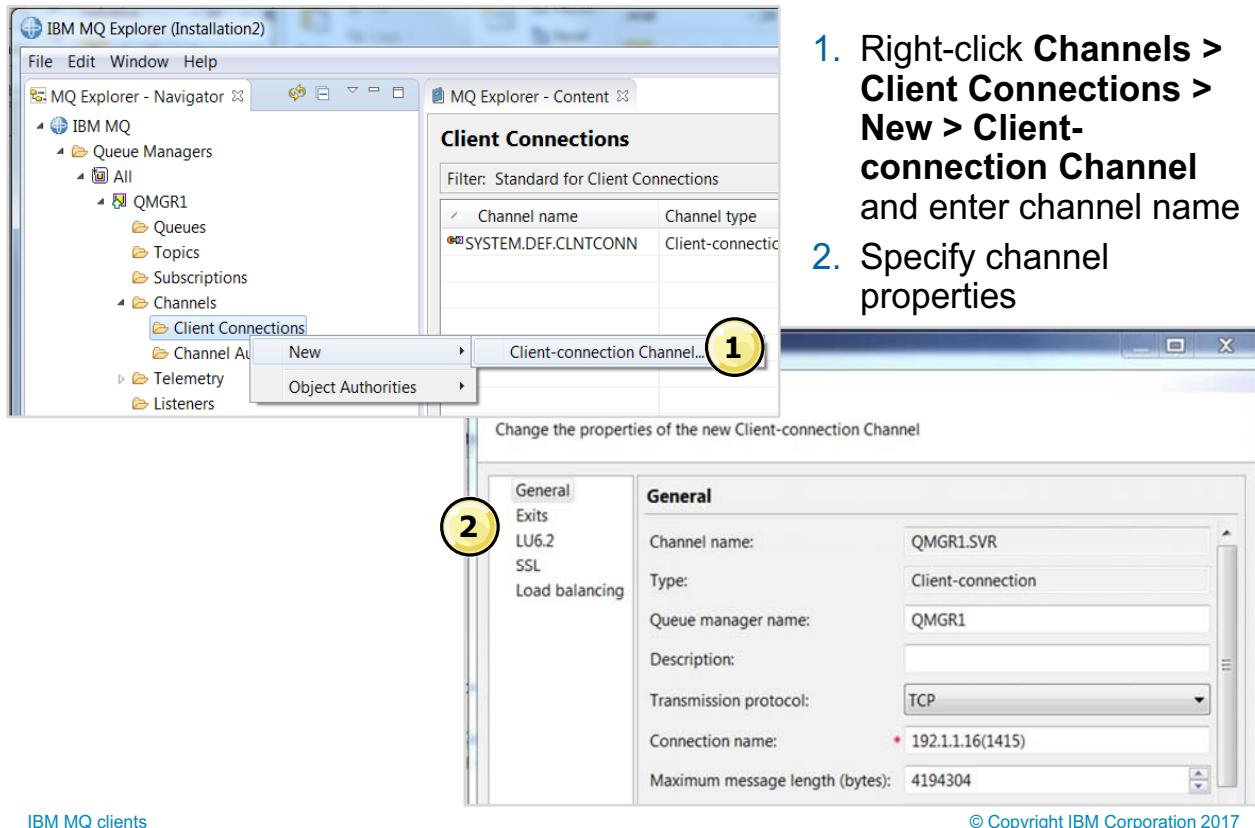
On Windows, the syntax is:

```
SET MQCHLLIB=C:\Program Files\IBM\MQ
SET MQCHLTAB=AMQCLCHL.TAB
```

On UNIX or Linux, the syntax is:

```
export MQCHLLIB=/var/mqm/
export MQCHLTAB=AMQCLCHL.TAB
```

Defining client-connection channels in IBM MQ Explorer



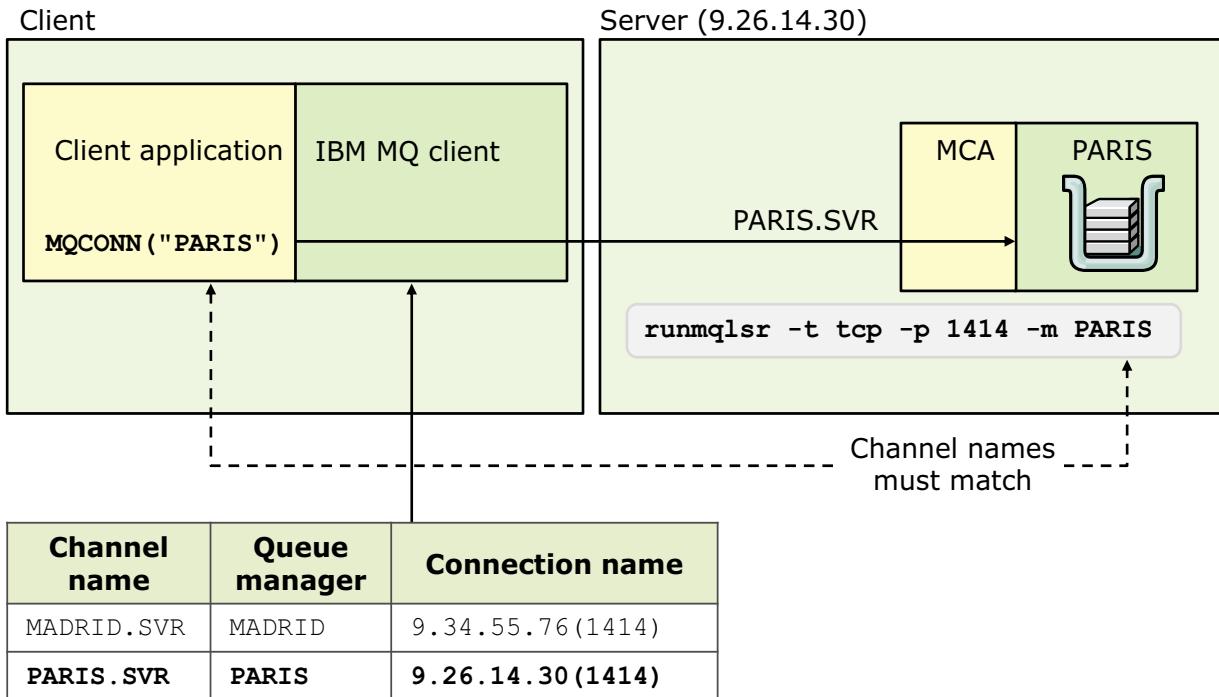
IBM MQ clients

© Copyright IBM Corporation 2017

Figure 7-16. Defining client-connection channels in IBM MQ Explorer

You can define a client-connection channel on the MQ server by using the **DEFINE CHANNEL** command or by using MQ Explorer.

Accessing a CCDT



IBM MQ clients

© Copyright IBM Corporation 2017

Figure 7-17. Accessing a CCDT

As shown in the figure, the connection name in the MQCONN call from the client must match the queue manager name in the `runmqlsr` command on the server. The client uses the CCDT to get the connection information for the server.

URL support for CCDT files

- Supports “`ftp://`”, “`file://`” and “`http://`” protocols
- `MQCCDTURL` environment variable specifies full URL

Examples:

```
MQCCDTURL=file:///var/mqm/qmgrs/QMGR/@ipcc/AMQCLCHL.TAB
MQCCDTURL=ftp://example.com//files/MYCHLTAB.TAB
MQCCDTURL=http://www.example.com/MYFILE.TAB
```

Examples with basic authentication:

```
MQCCDTURL=ftp://user:password@example.com//files/MYCHLTAB.TAB
MQCCDTURL=http://user:password@www.example.com/MYFILE.TAB
```

- Alternatively use `MQCHLLIB` to provide the stem of the URL

Example:

```
MQCHLLIB=ftp://user:password@example.com//files
MQCHLTAB=MYCHLTAB.TAB
```

Figure 7-18. URL support for CCDT files

IBM MQ V9.0 can locate a CCDT through a URL, either by programming, by using the `MQCCDTURL` and `MQCHLLIB` environment variables, or by using the `mqclient.ini` file stanzas.



Note

You can use the environment variable option only for C, COBOL, or C++ applications. The environment variables have no effect for Java, JMS, or managed .NET applications.

The `MQCCDTURL` environment variable specifies a file, FTP, or HTTP URL as a single value from which a CCDT can be obtained.

You can also use the `MQCHLLIB` environment variable or the `ChannelDefinitionDirectory` variable under the `CHANNELS` stanza of the client configuration file to locate a CCDT file. The `MQCHLLIB` value is a directory stem and works in combination with `MQCHLTAB` to derive the fully qualified URL.

Precedence order for finding the CCDT

1. Application specifies connection details in MQCONN call
 2. Application specifies CCDT URL location in MQCONN call
 3. MQSERVER environment variable
 4. MQCCDTURL environment variable
 5. MQCHLLIB and MQCHLTAB environment variables
 6. ChannelDefinitionDirectory attribute in CHANNELS stanza of **mqclient.ini** file
- Errors that retrieve CCDT are reported in error logs

Example:

AMQ9795: The client channel definition could not be retrieved from its URL, error code (16).

EXPLANATION:

The client channel definition location was specified as URL '<http://www.example.com/files/AMQCLCHL.TAB>', however the file could not be retrieved from this location.

The error returned was (16) 'HTTP response code said error'. The protocol specific response code was (404).

ACTION:

Ensure that the URL is reachable and if necessary correct the details provided.

Figure 7-19. Precedence order for finding the CCDT

This figure lists the order of precedence for a client application to find a CCDT.

If the client application cannot find a CCDT, an error is reported in the error logs.

Weighted selection on CLNTCONN channels

Client channel definition table

Group name	Channel name	Queue manager	Connection name	Client weight	Affinity
GRP1	LONDON.SVR	LONDON	9.45.31.124(1414)	20	NONE
GRP1	MADRID.SVR	MADRID	9.34.55.76(1414)	50	PREFERRED
GRP1	PARIS.SVR	PARIS	9.26.14.30(1414)	30	NONE
GRP1	ROME.SVR	ROME	9.14.78.101(1414)	0	PREFERRED

Application 1:
`MQCONN ("*GRP1")`

First attempt: ROME.SVR
Subsequent attempts:
ROME.SVR

Application 2:
`MQCONN ("*GRP1")`

First attempt: ROME
Next attempt: 50%
chance MADRID
then 100% MADRID

Application 3:
`MQCONN ("*GRP1")`

First attempt: ROME
Next attempt: 30%
chance PARIS then 100%
ROME (if up) or MADRID,
PARIS, or LONDON

* Distribution is not guaranteed

IBM MQ clients

© Copyright IBM Corporation 2017

Figure 7-20. Weighted selection on CLNTCONN channels

MQ has two attributes on the client-connection channel definition: client weighting (`CLNTWIGHT`) and affinity (`AFFINITY`). These attributes provide load sharing and selection of a channel that is based on the weights that are assigned to the channel when it is defined. The special channel weighting of 0 means to always use this channel definition. However, if this connection is not available, then further matching channels are tried. If multiple matching channels with a weighting of 0 exist, then these channels are tried first, in alphabetical order.

For nonzero client weightings, weights are summed; then, a random number between 1 and that sum is generated. The sum is used to select the matching channel. In the example in the figure, a weight of 1 – 20 causes LONDON to be selected, 21 – 70 selects MADRID, and 71 – 99 selects PARIS.

The channel `AFFINITY` attribute is used so client applications that connect to the same queue manager multiple times can choose to use the same client channel definition for each connection.

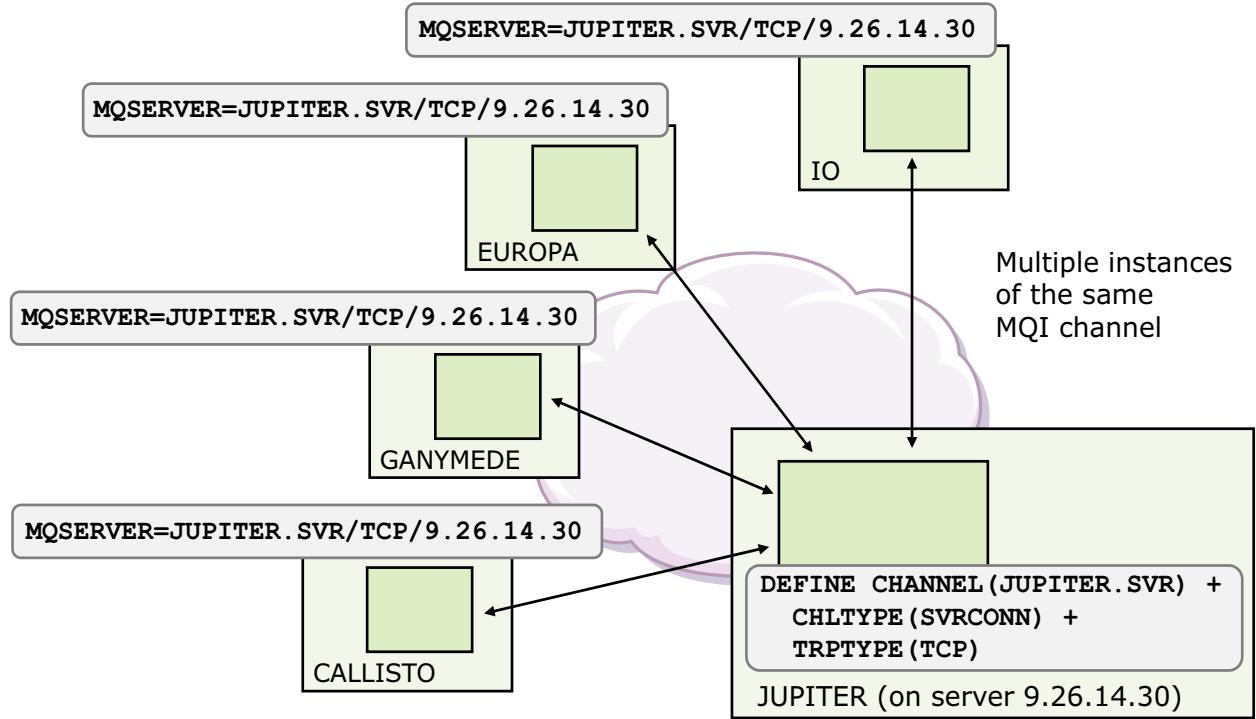
The example shows how these attributes work together. In the example, Application 1 chooses the connection to ROME because the client weighting is zero. The `AFFINITY` value is set to `PREFERRED`, so subsequent connections use this connection when it is available; otherwise, the next matching connection is tried.

Application 2 also tried to connect to ROME, but it is not responding for some reason. In the absence of any other weight 0 channels, the channel selection algorithm chooses MADRID (50% of

the time) which also has an **AFFINITY** of **PREFERRED**. Subsequent connections from this client return to MADRID while it is available.

Application 3 also fails to connect to ROME and connects to PARIS. On PARIS, **AFFINITY** is set to **NONE** so subsequent connections from this client start a new selection process. If ROME becomes available, that connection is reestablished; otherwise, it is distributed among the remaining three channels that are based on their weights.

MQI channel instances



IBM MQ clients

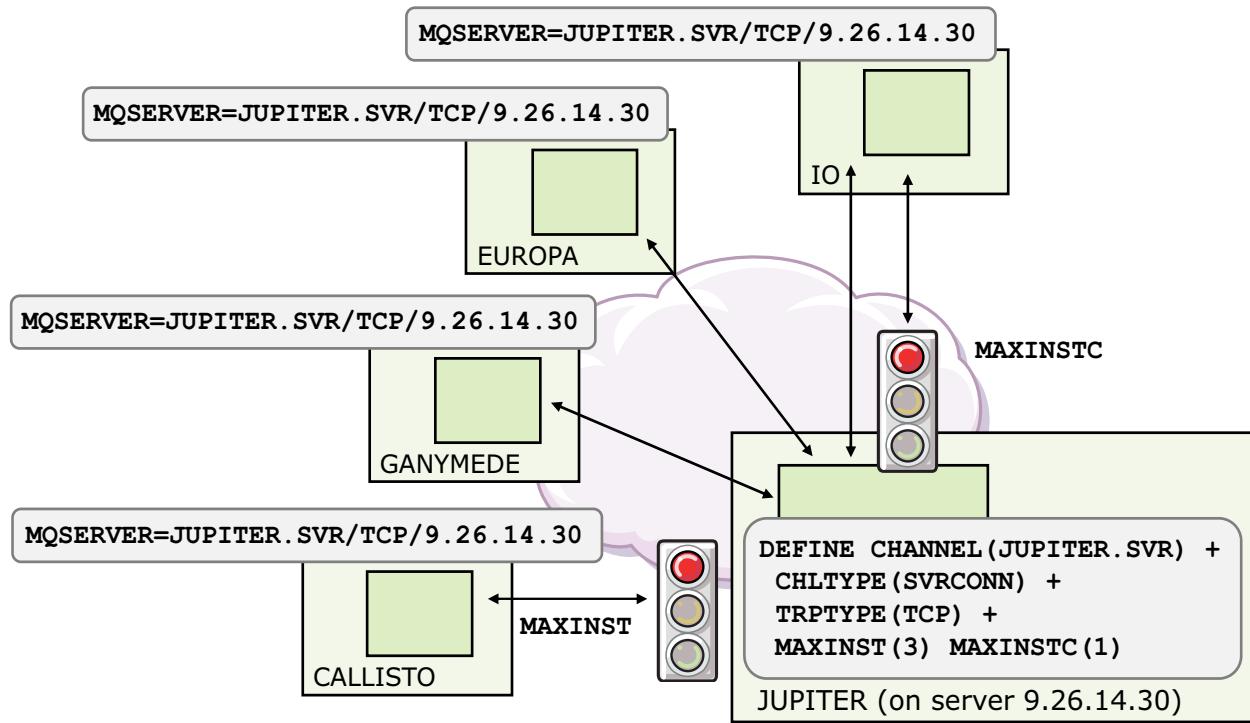
© Copyright IBM Corporation 2017

Figure 7-21. MQI channel instances

It is possible for a server to support multiple instances of a client-connection channel. In the example, each of the four MQI channels that are shown in the figure is an *instance* of the same MQI channel.

In this example, the environment variable `MQSERVER` has an identical value on each of the client systems. On the server system, JUPITER, the queue manager requires just one server connection definition for the MQI channel `JUPITER.SVR`. Each client application is using a different instance of the MQI channel `JUPITER.SVR`.

Instance limits on SVRCONN channels



IBM MQ clients

© Copyright IBM Corporation 2017

Figure 7-22. Instance limits on SVRCONN channels

The **MAXINST** server-connection channel attribute specifies the maximum number of simultaneous instances of a server-connection channel that can be started. A value of zero prevents all client access on this channel.

The **MAXINSTC** server-connection channel attribute specifies the maximum number of simultaneous instances of a server-connection channel that can be started from a single client. This attribute can be set from zero through 999 999 999. A value of zero indicates that no client connections are allowed on this channel.

If the **MAXINST** or **MAXINSTC** value is dynamically reduced, no new channel instances are started from this client, and running channel instances remain running.

The example shows four separate clients, all attempting to connect over the server-connection channel JUPITER.SVR. Three of the clients connect with one instance each: IO, EUROPA, and GANYMEDE. The fourth client CALLISTO is refused connection because of the **MAXINST** value of 3 on the server-connection definition.

IO tries to start a second client connection over the same JUPITER.SVR channel. The connection is refused also because it exceeds the **MAXINSTC** value of JUPITER.SVR. The refusal would happen irrespective of how many other clients are connected. That is, it is not dependent on **MAXINST**.

Auto-definition of channels

- Applies only to the end of a channel with type:
 - Receiver
 - Server connection
- Function that is started when an incoming request is received to start a channel but no channel definition exists
- Channel definition is created automatically by using the model:
 - SYSTEM.AUTO.RECEIVER
 - SYSTEM.AUTO.SVRCONN
- Connection partner values are used for:
 - Channel name
 - Sequence number wrap value

Figure 7-23. Auto-definition of channels

In MQ, you can enable the automatic definition of a channel if no appropriate channel definition for a receiver or server-connection channel exists. The function is started if an incoming request to start a message channel or MQI channel exists but no channel definition exists for the specific channel.

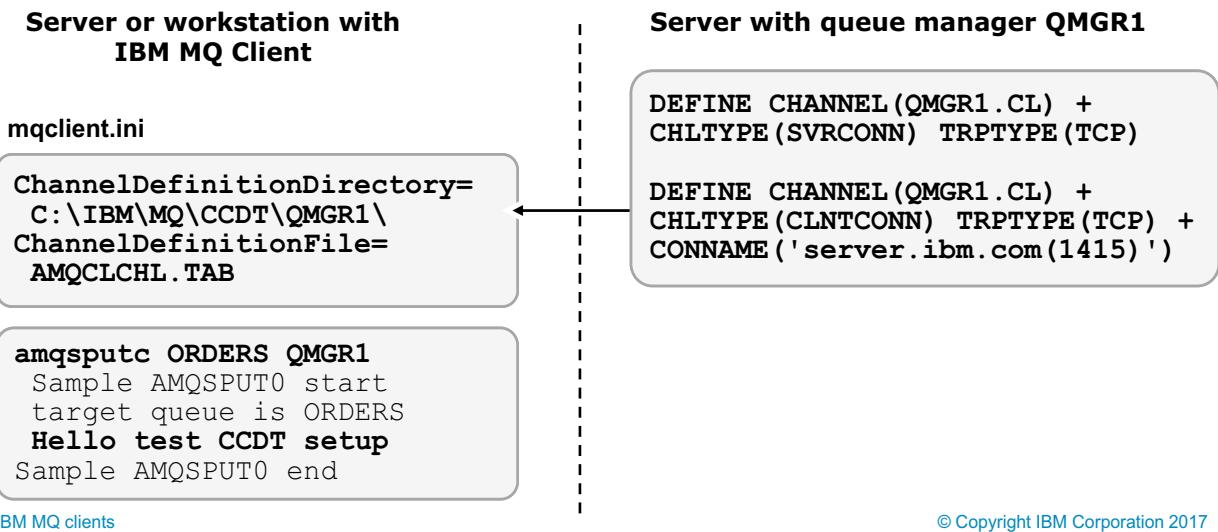
To enable the automatic definition of channels, the attribute **ChannelAutoDef** of the queue manager object must be set to MQCHAD_ENABLED. The corresponding parameter on the **ALTER QMGR** command is **CHAD(ENABLED)**.

The definition is created by using the appropriate model channel definition (SYSTEM.AUTO.RECEIVER or SYSTEM.AUTO.SVRCONN) and the channel name and sequence wrap values from the client. Other attributes, such as the batch size and maximum message length for a message channel, are negotiated with the client as normal.

After a channel definition is created automatically, the definition can be used as though it always existed.

Testing client connectivity

- Requirements:
 - Define SVRCONN, CLNTCONN, and local queue on IBM MQ queue manager
 - Copy CCDT from server to client
 - Reference CCDT on client
 - On client, run IBM MQ sample client PUT program `amqsputc`



IBM MQ clients

© Copyright IBM Corporation 2017

Figure 7-24. Testing client connectivity

You can test client connectivity by using the MQ `amqsputc` sample program. The figure summarizes the steps for defining and testing an MQ client connection to an MQ server.

You configure and test a client connection in the lab exercise for this unit.



Attention

By default, channel authentication is enabled when a queue manager is created. Channel authentication prevents privileged users from accessing a queue manager as an IBM MQ MQI client. For verifying the installation, you can either change the MCA user ID to a non-privileged user, or disable channel authentication.

To disable channel authentication, run the following MQSC command:

```
ALTER QMGR CHLAUTH(DISABLED)
```

When you finish the test, if you do not delete the queue manager, reenable channel authentication:

```
ALTER QMGR CHLAUTH(ENABLED)
```

Testing client-to-server connectivity with MQCONN

- Requirements:
 - SVRCONN on the IBM MQ server
 - IBM MQ client is installed on client computer
 - IBM MQ sample client program to test connectivity: `amqscnxc`

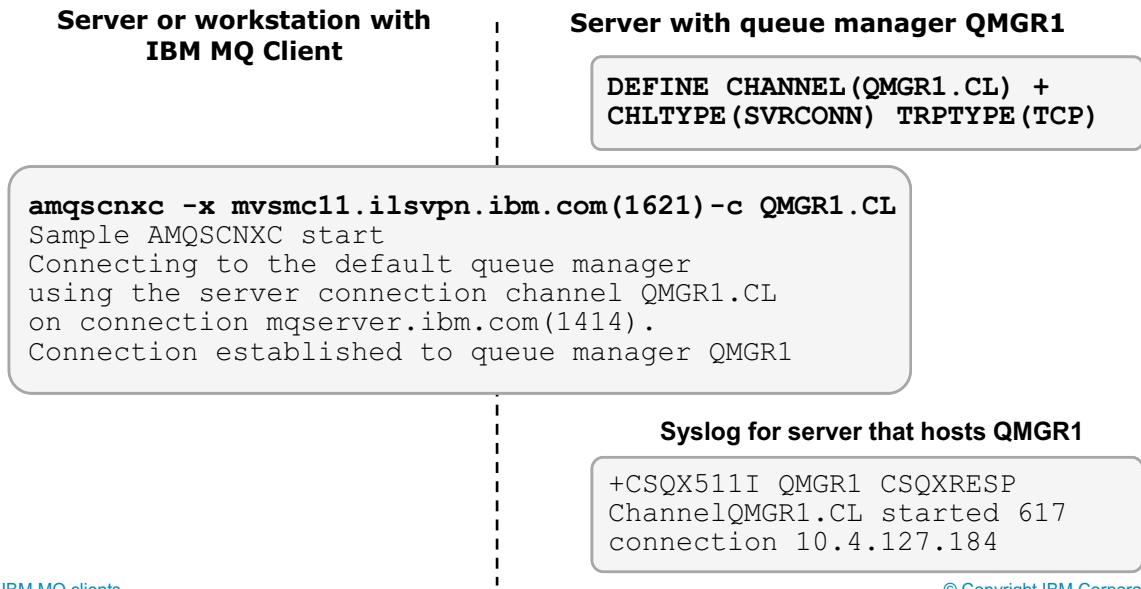


Figure 7-25. Testing client-to-server connectivity with MQCONN

You can also use a sample program to simulate a connection by using an MQCONN call that is sent from the MQ sample application `amqscnxc`.

In the example, the `amqscnxc` sample program connects to the server-connection channel MQ0A.CL that is defined on queue manager MQ0A.

On the client side, the MQCONN call contains all the information necessary to connect to the queue manager server-connection channel.

IBM MQ client limitations

- Remote connection to IBM MQ server relies on network connectivity
- Distribution of IBM MQ maintenance updates for clients
- Control of client applications by IBM MQ administrators

IBM MQ clients

© Copyright IBM Corporation 2017

Figure 7-26. IBM MQ client limitations

Some limitations exist when you use MQ clients:

- Clients use network connections. This limitation introduces a dependency on network availability. Network availability also applies when the MQ client and server are on the same server.
- Distribution of MQ client maintenance can be a problem, particularly when control of the client computer is with another organization. As with any distributed component, keeping the MQ client software up-to-date becomes a challenge, particularly if control of the client computer is the responsibility of another organization.
- The MQ administrator might not have access to control the client application.

IBM MQ client security

- When running a client application, do not run the application by using a user ID that has more access rights than necessary
- Two options for authenticating users to ensure that the client application user ID is genuine
 - Connection authentication
 - Using mutual authentication within TLS
- Use access control to give or remove access rights for a specific user or group of users
- Configuration support in IBM MQ Explorer
 - Setting the default security exit for all client connections
 - Enabling the default TLS/SSL options for all client connections
 - Enabling the default TLS/SSL stores
 - Specifying the user ID and password server security exit users use to establish the identity of the IBM MQ client

IBM MQ clients

© Copyright IBM Corporation 2017

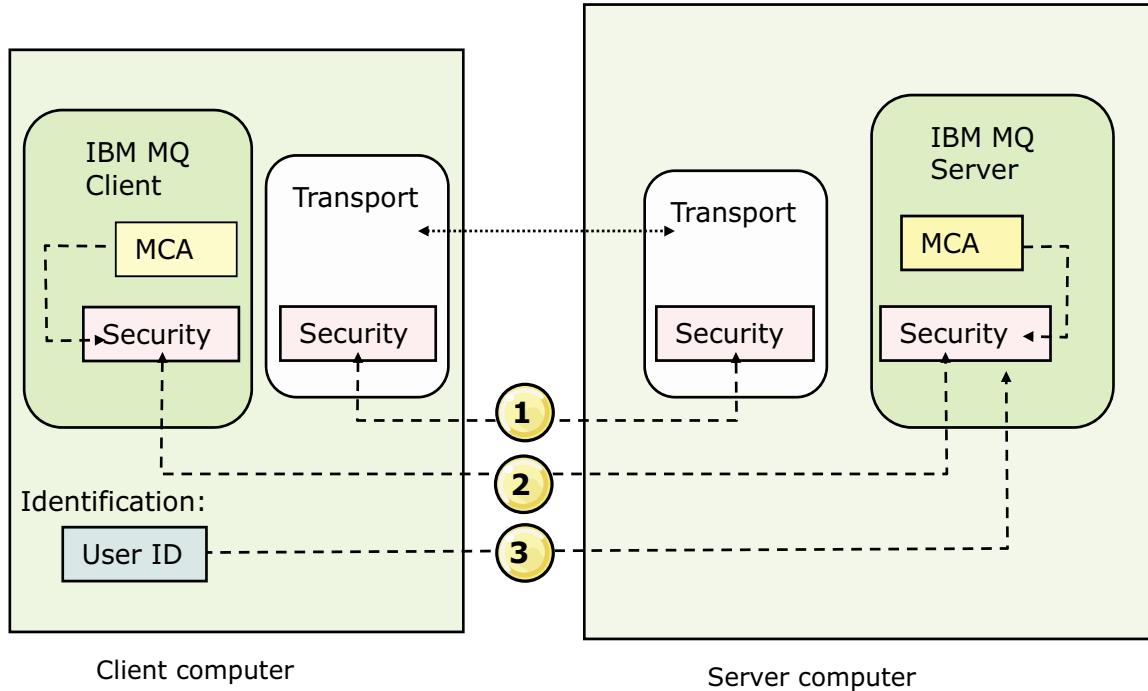
Figure 7-27. IBM MQ client security

You must consider MQ client security so that the client applications do not have unrestricted access to resources on the server.

Two aspects to security between a client application and its queue manager server exist: authentication and access control. Client access control is based on user IDs.

Use the client security configuration support in MQ Explorer to set default security options on all new client connections. These default values can be overridden when a new queue manager is added.

IBM MQ client security checks



IBM MQ clients

© Copyright IBM Corporation 2017

Figure 7-28. IBM MQ client security checks

Three levels of security to consider for MQ clients are:

1. Transport level checks, such as IP filtering.
2. Custom channel exits, such as to check security to a custom specification.
3. Default user ID propagation such as allowing MQ to pass the currently logged-on user ID that a server-side security exit can use to authenticate.

IBM MQ redistributable client

- IBM MQ client (C, C++, COBOL), Java and JMS, and .NET runtimes are available with redistributable license terms
 - Zip and Tar.gz images that contain redistributable runtime libraries
 - 64-bit Windows and Linux x86-64
- Applications can be rolled out with specific versions of IBM MQ client runtime
 - Can be shared on networked file system
 - Supports coexistence with other redistributable runtimes or with full IBM MQ client/server installations
- IBM MQ license agreement is extended to allow redistributable license usage of files with `Redist` in the name

IBM MQ clients

© Copyright IBM Corporation 2017

Figure 7-29. IBM MQ redistributable client

IBM MQ provides redistributable client runtimes.

A redistributable client implies distributing the required runtime with an application both inside and outside of your environment.

Unit summary

- Describe the various ways to connect a client to a queue manager
- Analyze client connection requirements
- Describe the limitations of various client connection methods
- Propose methods to ensure security with client connections

IBM MQ clients

© Copyright IBM Corporation 2017

Figure 7-30. Unit summary

Review questions

1. Which of the following options best describes an IBM MQ client?
 - A. A channel to an IBM MQ server
 - B. An API for IBM MQ
 - C. A customer who purchases IBM MQ
 - D. Libraries and communications that allow an application to use a remote IBM MQ server
2. True or False: An IBM MQ client can participate in global (XA) units of work.
3. Which of the following statements are true?
 - A. MAXINST limits the total number of client connections for a specific channel.
 - B. MAXINSTC limits the total number of connections for a specific channel from an individual client.
 - C. Affinity causes subsequent client connections from a specific client to return to the same channel.
 - D. All of the above.



IBM MQ clients

© Copyright IBM Corporation 2017

Figure 7-31. Review questions

Write your down answers here:

- 1.
- 2.
- 3.

Review answers

1. Which of the following options best describes an IBM MQ client?
 - A. A channel to an IBM MQ server
 - B. An API for IBM MQ
 - C. A customer who purchases IBM MQ
 - D. Libraries and communications that allow an application to use a remote IBM MQ server

The answer is C and D.
2. True or False: An IBM MQ client can participate in global (XA) units of work.
 The answer is False. The IBM MQ extended transactional client is required to participate in global transactions.
3. Which of the following statements are true?
 - A. MAXINST limits the total number of client connections for a specific channel.
 - B. MAXINSTC limits the total number of connections for a specific channel from an individual client.
 - C. Affinity causes subsequent client connections from a specific client to return to the same channel.
 - D. All of the above.

The answer is D.

IBM MQ clients

© Copyright IBM Corporation 2017

Figure 7-32. Review answers

Exercise: Connecting an IBM MQ client

IBM MQ clients

© Copyright IBM Corporation 2017

Figure 7-33. Exercise: Connecting an IBM MQ client

In this exercise, you configure your client connection to an IBM MQ server. You use different methods to gain experience with the client connectivity methods that are available in IBM MQ. In this exercise, you also encounter client connectivity security problems and use the MCAUSER attribute to solve this problem.

Exercise objectives

- Create a server connection channel to support client connections
- Use a URL to specify the location of the client connection definition table
- Use the MQSERVER environment variable to specify a client connection channel
- Use the client configuration file to specify a client connection channel



IBM MQ clients

© Copyright IBM Corporation 2017

Figure 7-34. Exercise objectives

See the *Course Exercises Guide* for detailed instructions.

Unit 8. Implementing trigger messages and monitors

Estimated time

00:30

Overview

IBM MQ can be configured to trigger an application automatically when certain conditions on a queue are met. In this unit, you learn how to configure IBM MQ to trigger an application.

How you will check your progress

- Review questions
- Hands-on lab exercise

References

IBM Knowledge Center for IBM MQ V9

Unit objectives

- Configure IBM MQ to enable a trigger monitor
- Run the trigger monitors on distributed operating systems
- Determine the cause of a triggering failure

Implementing trigger messages and monitors

© Copyright IBM Corporation 2017

Figure 8-1. Unit objectives

One of the most common functions that are used in MQ is triggering. Triggering provides message-driven, or event-driven processing. In this unit, the requirements for triggering are described so that you can gain a clear understanding of the advantages. This unit also describes some of the common problems that you might encounter with triggering.

Using triggering to start IBM MQ applications

- *Triggering* is an IBM MQ function that starts an application automatically when messages are put on a queue
 - Supported by IBM MQ clients that run on UNIX, Linux, and Windows
- Provides:
 - Instantiation as required
 - Conservation of system resources
 - Automation of message flow

Implementing trigger messages and monitors

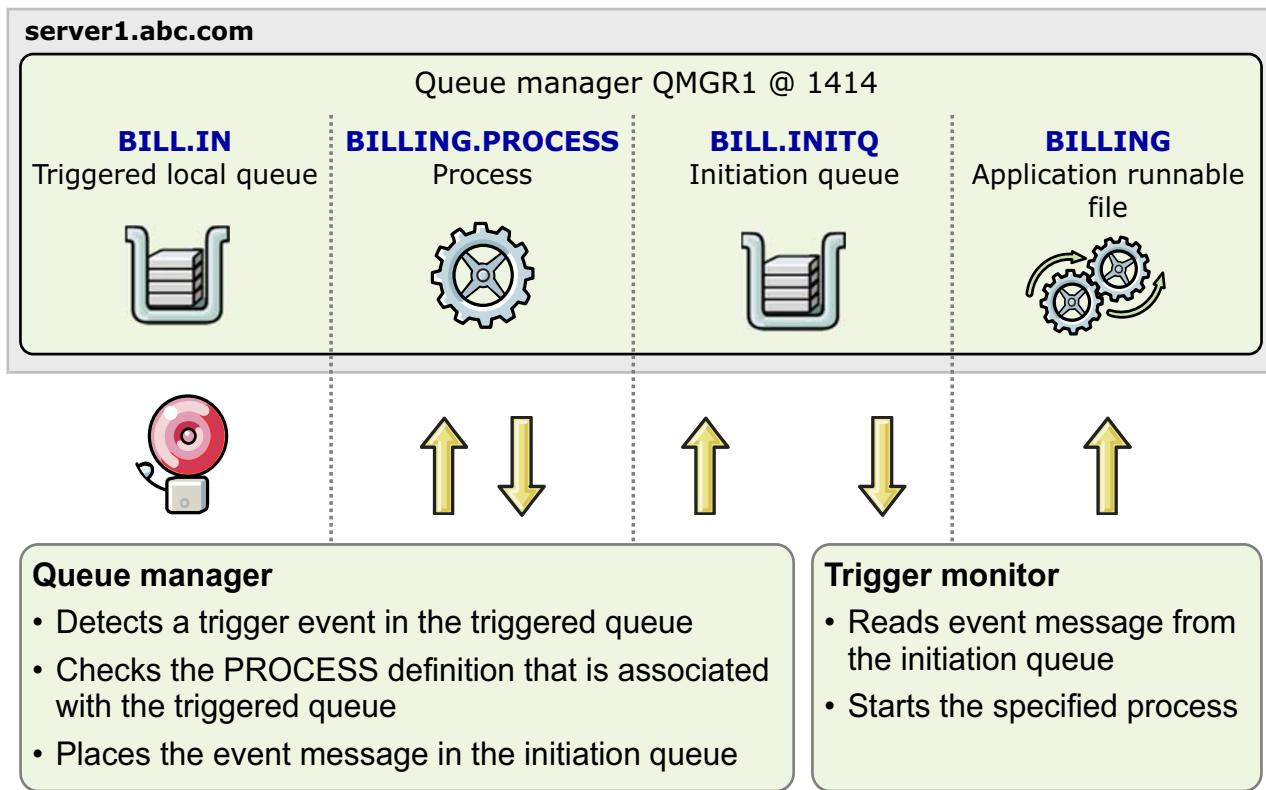
© Copyright IBM Corporation 2017

Figure 8-2. Using triggering to start IBM MQ applications

Some IBM MQ applications that serve queues run continuously, so they are always available to retrieve messages that arrive on the queues. However, you might not want this behavior when the number of messages that arrive on the queues is unpredictable. In this case, applications might be using system resources even when the queues are empty.

MQ application triggering provides an enhancement to the implementation of time-independent processing. The arrival of a message on an application queue might indicate that it is an appropriate time for another application to start to process the messages on the application queue. When the queue manager detects the right conditions, the triggering facility starts the application to service the application queue.

Application triggering



Implementing trigger messages and monitors

© Copyright IBM Corporation 2017

Figure 8-3. Application triggering

To trigger a queue to start a process to get messages, you must define a process. The process definition contains details on where to find the program or script to start.

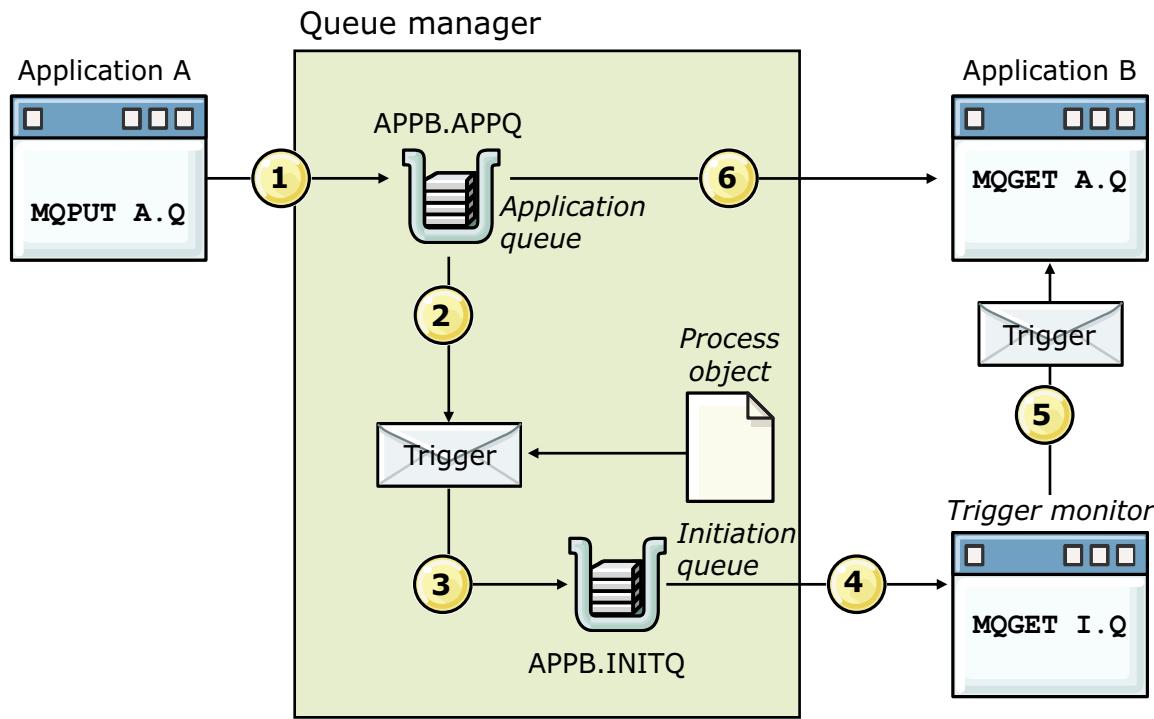
A local queue is the object that is triggered. Triggering attributes are set on the queue to indicate that triggering is to occur, and to specify the trigger conditions.

Another trigger component is the initiation queue. The initiation queue is another queue where the queue manager puts specially formatted trigger messages.

The two other triggering components are the queue manager and a process that is called the trigger monitor, which must be running for triggering to occur. You can also configure the trigger monitor to start when the queue manager starts.

Triggering requires coordination between the queue manager and the trigger monitor. When the defined triggering conditions are met, the queue manager checks the process definition and then places a specially formatted message in the initiation queue. The initiation queue is under the watch of the trigger monitor, which gets the message from the initiation queue and starts the appropriate process.

Components of triggering



Implementing trigger messages and monitors

© Copyright IBM Corporation 2017

Figure 8-4. Components of triggering

The figure shows triggering components and the sequence of actions.

1. Application A puts a message on an application queue that is enabled for triggering.
2. If the conditions for triggering are met, a *trigger event* occurs, and the queue manager examines the process object that the application queue references. The process object identifies the application to start, Application B.
3. The queue manager creates a *trigger message* whose fields contain information that is copied from certain attributes of the process object and the application queue. The queue manager puts the trigger message on an *initiation queue*.
4. A long-running program that is called a *trigger monitor* gets the trigger message and examines its contents. A trigger monitor can start the application synchronously within its own unit of execution, or asynchronously as a separate unit of execution. Trigger monitors are supplied with MQ, but users can write their own.
5. The trigger monitor starts Application B, passing the entire trigger message as a parameter.
6. Application B opens the application queue and gets messages from it.

Queue attributes that control triggering

Attribute	Description
TRIGGER	Specifies whether trigger messages are written to the initiation queue
NOTRIGGER	
TRIGMPRI (integer)	Priority of message that is required to generate a trigger event
TRIGTYPE (FIRST)	Specifies whether and under what conditions a trigger message is written to the initiation queue
TRIGTYPE (DEPTH)	
TRIGTYPE (EVERY)	
TRIGTYPE (NONE)	
TRIGDPTH (integer)	Number of messages that are required to generate a trigger event
TRIGDATA (string)	Data that is inserted in the trigger message
PROCESS (string)	Identifies application that can service the application queue
INITQ (Qname)	Name of initiation queue

Example

```
DEFINE QLOCAL(MY_SERVER) TRIGGER +
PROCESS(ECHO) TRIGTYPE(FIRST) +
INITQ(SALES.INITQ)
```

Figure 8-5. Queue attributes that control triggering

To define an application queue for triggering, the MQSC command **DEFINE QLOCAL** must contain the following parameters:

- **PROCESS(string)**: The name of the process object that identifies the application that can service the application queue.
- **INITQ(string)**: The name of the initiation queue.

The figure lists queue attributes that control triggering.

The **TRIGTYPE** parameter specifies whether and under what conditions a trigger message is written to the initiation queue.

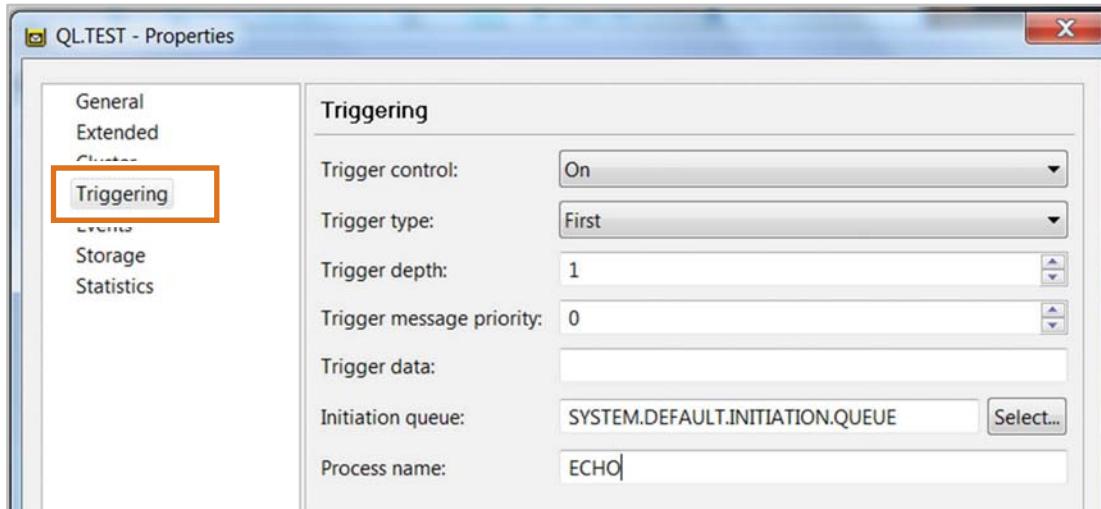
- When **TRIGTYPE** is set to **FIRST**, a trigger event occurs when the queue changes from empty to having one message. For most purposes, trigger type **FIRST** is the most appropriate choice.
- When **TRIGTYPE** is set to **DEPTH**, a trigger event occurs when the number of messages on the queue reaches the **TRIGDPTH** parameter value. When triggering by depth, the queue manager disables triggering by setting the application queue to **NOTRIGGER** after it creates a trigger message. It is the responsibility of the application to reenable triggering by using the MQSET call.

One example where trigger type **DEPTH** would be more appropriate is when an application is expecting a fixed number of related reply messages and creates a dynamic queue to receive them. The dynamic queue can be enabled for triggering by depth with **TRIGDEPTH** set to the number of reply messages expected.

- When **TRIGTYPE** is set to **EVERY**, a trigger event occurs for every message that is put on the application queue.

The action of disabling triggering is not under sync point control, so triggering cannot be enabled again by backing out a unit of work. If a program backs out a PUT request or if a program that caused the triggered event abends, you must reenable triggering by using the MQSET call or the **ALTER QL** command.

Defining triggering properties in IBM MQ Explorer



1. Right-click a local queue in the **Queue Contents** view and then click **Properties**
2. Click **Triggering**
3. Specify triggering properties for the queue and then click **Apply**

Implementing trigger messages and monitors

© Copyright IBM Corporation 2017

Figure 8-6. 'Defining triggering properties in IBM MQ Explorer'

You can also enable triggering and define the trigger properties for a queue in MQ Explorer, as shown in the figure.

Initiation queues

- Create an initiation queue for the application

Example:

```
DEFINE QLOCAL(SALES.INITQ) REPLACE +
LIKE(SYSTEM.DEFAULT.INITIATION.QUEUE) +
DESCR('initiation queue description')
```

- Associate the initiation queue with the application queue and specify the triggering properties

Example:

```
DEFINE QLOCAL(APP1Q) REPLACE +
LIKE(SYSTEM.DEFAULT.LOCAL.QUEUE) +
DESCR('APP1 queue description') +
INITQ(SALES.INITQ) +
PROCESS('APP1') +
TRIGGER TRIGTYPE(FIRST)
```

[Implementing trigger messages and monitors](#)

© Copyright IBM Corporation 2017

Figure 8-7. Initiation queues

Before your application can take advantage of triggering, create an initiation queue for the application queue.

After you define an initiation queue, you must associate the initiation queue with the application queue.

A queue manager can own more than one initiation queue. For example, you might want different programs to serve the application queues. In this case, you can use one initiation queue for each serving program.

Defining an IBM MQ process

- Use the MQSC command **DEFINE PROCESS** to define an IBM MQ process and set its attributes

Attribute	Description
APPLICID (string)	Application type to start
APPLTYPE (string)	Type of application to start
ENVRDATA (string)	Environment information that the trigger monitor passes to the application
USERDATA (string)	User information that the trigger monitor passes as part of a parameter list

Example

```
DEFINE PROCESS (ECHO) +
APPLICID ('/opt/mqm/samp/bin/amqsech')
```

Figure 8-8. Defining an IBM MQ process

The trigger definition requires the name of the process object that identifies the application that can service the application queue. This figure shows the MQSC command and options that define a process object.

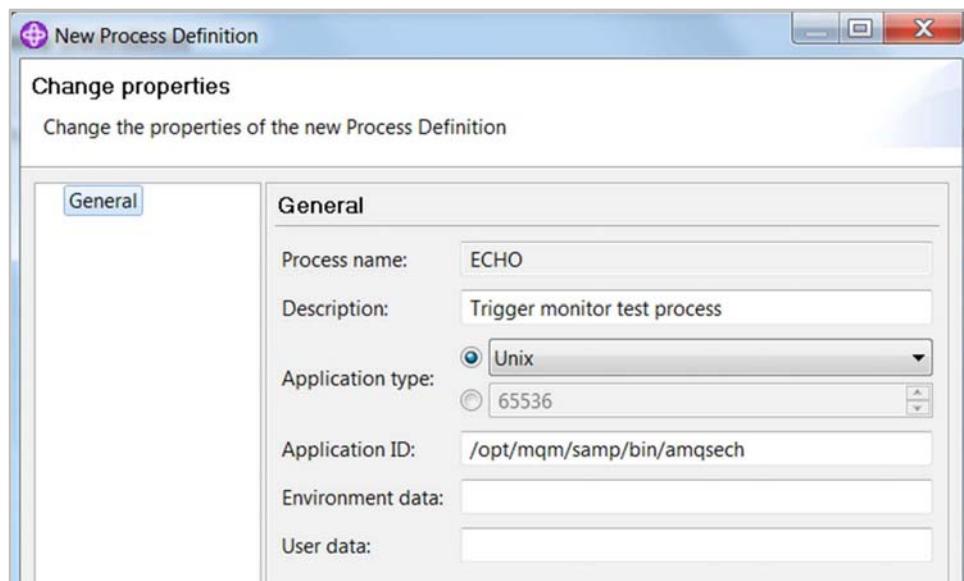
A process and a queue are allowed to have the same name. The name of an object must be unique within an object type.

The **APPLICID** value depends on the **APPLTYPE** parameter. For example, if the application type is UNIX, the **APPLICID** is a fully qualified path of a runnable object, as shown in the example in the figure.

If you want to trigger the start of a channel, it is not necessary to define a process definition object. The transmission queue definition can determine the channel to trigger.

The example in the figure is a UNIX example. It might not always be necessary to provide the full path as shown.

Defining a process in IBM MQ Explorer



1. Right-click **Process Definitions** under queue manager and then click **New > Process Definition**
2. Enter a process name and then click **Next**
3. Specify process definition properties and then click **Finish**

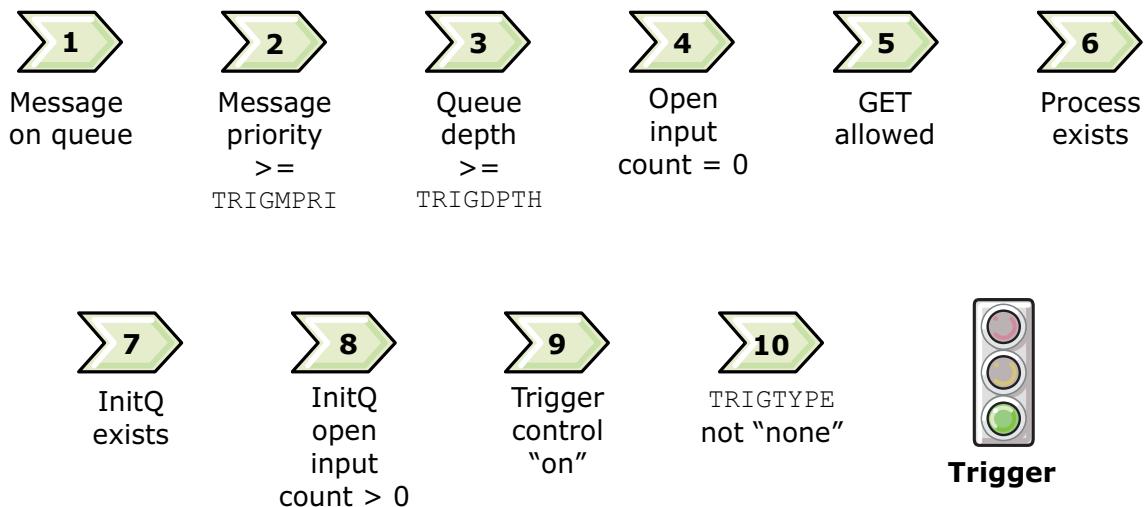
Implementing trigger messages and monitors

© Copyright IBM Corporation 2017

Figure 8-9. Defining a process in IBM MQ Explorer

As shown in the figure, you can also define a process in IBM MQ Explorer.

Application triggering conditions



Implementing trigger messages and monitors

© Copyright IBM Corporation 2017

Figure 8-10. Application triggering conditions

This figure shows all of the conditions that must be satisfied for a trigger event to occur.

1. A message is put on a queue.
2. The priority of the message must be greater than or equal to the value of the trigger message priority attribute of the queue.
3. The number of messages on the queue with a priority greater than or equal to the trigger message priority attribute was previously:
 - No messages, for trigger type FIRST
 - Any number of messages, for trigger type EVERY
 - Trigger depth minus one message, for trigger type DEPTH
4. For triggering of type FIRST or DEPTH, no program has the application queue open for removing messages (the Open Input Count of the local queue attribute is zero).
5. The queue is enabled for GET requests.
6. The process object that the queue **Process Name** attribute identifies exists.
7. The initiation queue that is identified by the **Initiation Queue** queue attribute exists, and is enabled for PUT and GET requests.

8. A trigger monitor currently has the initiation queue open for removing messages (the OpenInputCount of the local queue attribute is greater than zero).
 9. Trigger control for the application queue is enabled; that is, the attribute for *trigger control* is set to ON.
10. The value of the **Trigger Type** attribute of the application queue is not NONE.

If a trigger event does not occur when it is expected, check this list.

Situations can occur in which enough messages exist for the trigger type on an application queue but not **all** the trigger conditions are satisfied. Therefore, without the additional conditions that are shown on the visual, a trigger event might never occur. For example, when a trigger monitor opens an initiation queue for input, the queue manager generates a trigger message for each application queue whose definition references the initiation queue. This process assumes that an application queue already has enough messages on it for its respective trigger type and any other relevant trigger conditions are also satisfied. Such a situation might occur following a queue manager restart, and persistent messages still exist on the application queues.

It might happen that the required conditions are met, and the message that caused the trigger condition is part of a unit of work. If so, the trigger message is not available for retrieval by the trigger monitor application unit until the unit of work completes. The result is the same whether the unit of work is committed or the trigger type FIRST or DEPTH is backed out.

Information in the trigger message

- Trigger message contains information about the application to start
- Information that is copied from the application queue
 - Qname
 - ProcessName
 - TriggerData
- Information that is copied from process object
 - ApplType
 - ApplId
 - EnvData
 - UserData

[Implementing trigger messages and monitors](#)

© Copyright IBM Corporation 2017

Figure 8-11. Information in the trigger message

A trigger message is put on an initiation queue when a trigger event occurs. From the point of view of the trigger monitor, the main purpose of a trigger message is to identify the application to start.

The MQ trigger message 2 character format (MQTMC2) defines the trigger message structure. Some fields in the trigger message are derived from attributes of the application queue, and some are derived from attributes of the process object that the queue definition references.

The figure lists the fields that are contained in the trigger message.

- **QName** is the name of the application queue.
- **ProcessName** is the name of the process object that identifies the application that can service the application queue.
- **TriggerData** is the trigger data that the started application uses. This field can be used to specify the name of the channel to trigger.
- **ApplType** is the application operating system, such as UNIX and Windows.
- **ApplId** is the application identifier of the application to start.
- **EnvData** is the environment data that the trigger monitor uses.
- **UserData** is the user data for use by the trigger monitor or by the started application.

Trigger monitor

- *Trigger monitor* is a continuously running program that serves one or more initiation queues
- To start a trigger monitor: `runmqtrm -m QMgrName -q InitQName`
 - If initiation queue not specified, trigger monitor uses SYSTEM.DEFAULT.INITIATION.QUEUE
- Trigger monitor uses information in trigger message to create an operating system specific command to start the target application

Command string: `AppId "MQTMC2" EnvData`

Where:

- **AppId** is the name of the program to run
- **MQTMC2** is a trigger message, which is enclosed in quotation marks, from the initiation queue and formatted in IBM MQ trigger message character format
- **EnvData** is the environment data from the relevant process definition

Figure 8-12. Trigger monitor

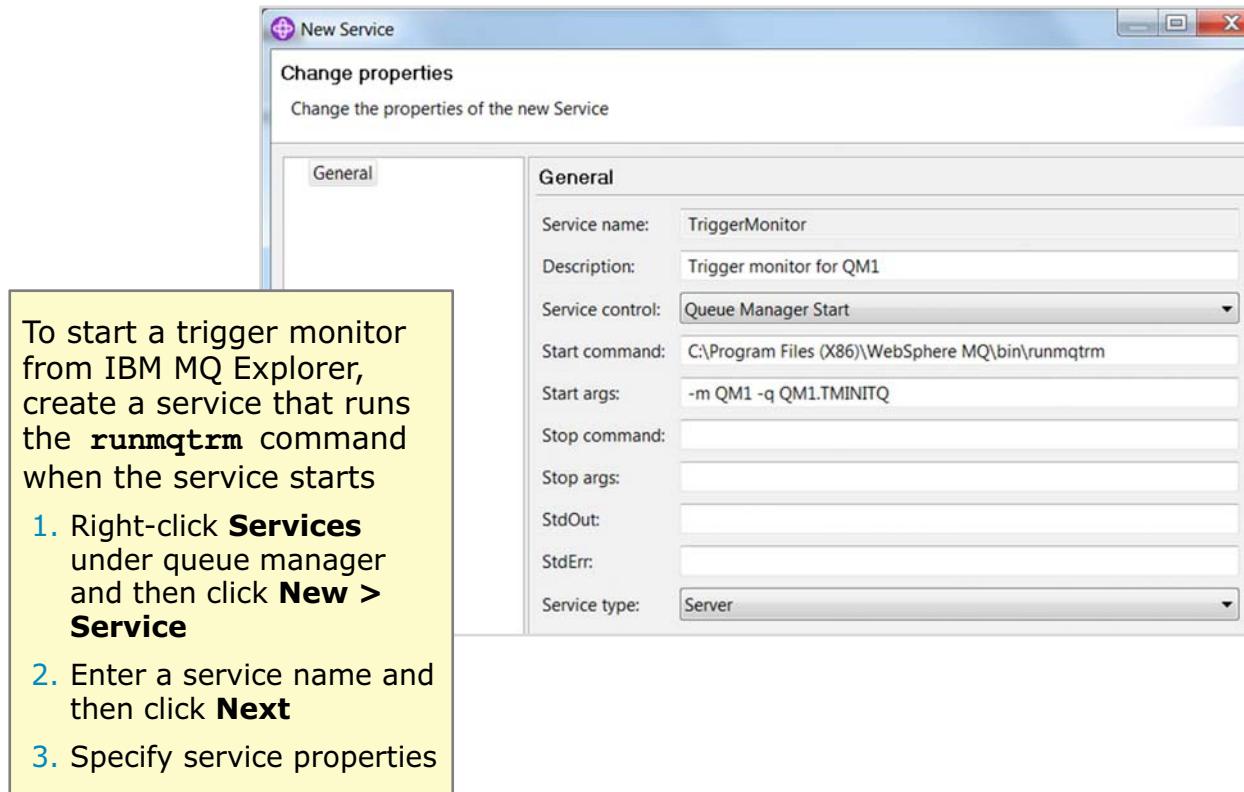
A trigger monitor is a long-running program that gets a trigger message from an initiation queue and starts an application to service the application queue.

To start a trigger monitor on distributed operating systems, enter the `runmqtrm` control command with the queue manager name and the initiation queue name. If the name of an initiation queue is not explicitly specified on the `runmqtrm` command, the queue `SYSTEM.DEFAULT.INITIATION.QUEUE` is used.

The trigger monitor sends a command string that uses the MQTMC2 structure as a parameter. This structure supplies the started application with the name of the queue that caused the trigger event and the name of the queue manager that owns it. The started application can connect to the queue manager and open the queue.



Defining a trigger monitor in IBM MQ Explorer



Implementing trigger messages and monitors

© Copyright IBM Corporation 2017

Figure 8-13. Defining a trigger monitor in IBM MQ Explorer

This figure shows the steps for defining a trigger monitor as a service in MQ Explorer.

The next figure describes the service options.

Trigger monitor properties in IBM MQ Explorer

- **Service control** specifies how service starts and stops
 - **Queue Manager** starts and stops service when queue manager starts and stops
 - **Queue Manager Start** starts service when queue manager starts but does not stop when queue manager stops
 - **Manual** requires a manual start and stop
- **Start command** is full path to the `runmqtrm` command
- **Start args** specifies `runmqtrm` command arguments
 - If queue manager is not default queue manager, type: `-m QMgrName`
 - To use an initiation queue other than `SYSTEM.DEFAULT.INITIATION.QUEUE`, type: `-q InitQName`
- **Service type** selects type of service to run
 - **Command** allows multiple instances of service but cannot view the status of service in IBM MQ Explorer
 - **Server** allows one instance of service but can view the status of service in IBM MQ Explorer

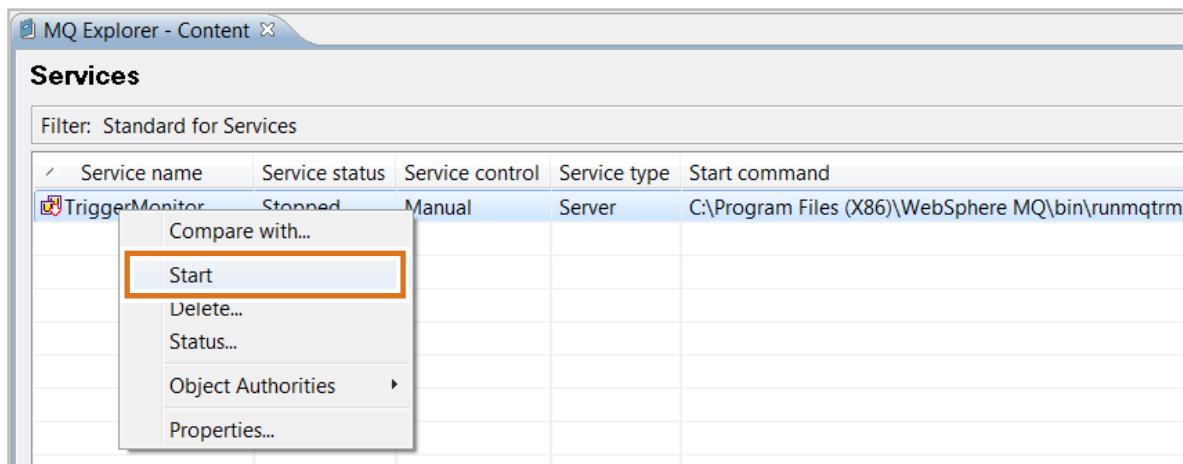
[Implementing trigger messages and monitors](#)

© Copyright IBM Corporation 2017

Figure 8-14. Trigger monitor service properties in IBM MQ Explorer

This figure describes the properties on the **New Service** window in MQ Explorer and how to configure them for a trigger monitor.

Starting the trigger monitor in IBM MQ Explorer



1. In **MQ Explorer - Navigator** view, click **Services** under queue manager name to display **Services** view
2. In **Services** view, right-click the service and then click **Start**
 - The icon next to the service changes to show whether the service is running

Implementing trigger messages and monitors

© Copyright IBM Corporation 2017

Figure 8-15. Starting the trigger monitor in IBM MQ Explorer

After you define the trigger monitor service in the **New Service** window, it is listed in the **Services** view.

You must start the trigger monitor from the IBM MQ Explorer by right-clicking the service and then clicking **Start**, as shown in this figure.

Testing a trigger monitor with an IBM MQ sample program

- **amqsreq ReqQ QMgrName ReplyToQ**

- Reads text from standard input device, converts it to request messages, and puts messages on request queue
- When a null line is entered, program gets messages from the reply-to queue and displays messages on the standard output device

- **amqsech**

- Started by a trigger monitor
- Gets a message from request queue
- Creates a reply message with same application data as original message
- Puts new reply message on the reply-to queue

- **amqsinq**

- Started by a trigger monitor
- Gets a message from request queue
- Creates a reply message with queue attribute values
- Puts message on reply-to queue

Implementing trigger messages and monitors

© Copyright IBM Corporation 2017

Figure 8-16. Testing a trigger monitor with an IBM MQ sample program

You can use some MQ sample programs to verify that your trigger monitor is working.

The **amqsreq** sample program is started from a command line. The first parameter is the name of the request queue. The second parameter is the queue manager name. The third parameter is the reply-to queue name. The program reads lines of text from the standard input device, converts them to request messages, and puts the messages on the named queue. When a blank line is entered, the program gets the messages from the reply-to queue and displays those messages on the standard output device.

A trigger monitor starts the **amqsech** sample program. The program connects to the queue manager that is named in the structure that is passed to it by the trigger monitor. The **amqsech** sample program then opens the request queue that is also named in the structure.

A trigger monitor starts the **amqsinq** sample program. The program connects to the queue manager, opens the request queue, gets a message from the queue, and interprets the application data as the name of a queue. The **amqsinq** program constructs a reply message and puts the message on the reply-to queue. The program then gets each of the remaining messages on the request queue in turn and generates a reply in the same way. When the request queue is empty, the program closes the queue and disconnects from the queue manager.

Monitoring trigger monitor status

Status message can be written to standard output device or error log

- Information status messages

Examples:

- *Trigger monitor started*
- *Waiting for a trigger message*
- *Trigger monitor ended*

- Error condition status messages

Examples:

- *Initiation queue cannot be opened*
- *Use of trigger monitor not authorized*
- *Error starting triggered application*

Trigger event message can be written to dead-letter queue

- Examples:

- *Trigger message structure is not valid*
- *Trigger message specifies an unsupported application type*
- *Trigger monitor cannot start the specified application*

[Implementing trigger messages and monitors](#)

© Copyright IBM Corporation 2017

Figure 8-17. Monitoring trigger monitor status

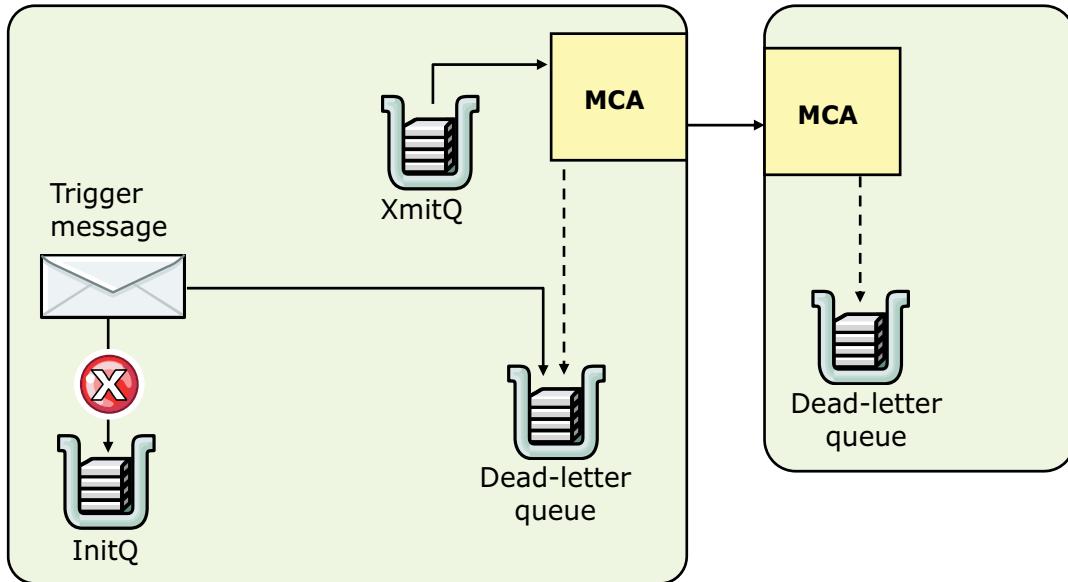
The trigger status messages are produced for two reasons.

- Trigger status messages report on normal activities, such as when a trigger monitor starts and when it ends. These messages do not normally require any user action.
- Trigger status messages report on abnormal conditions, such as when a trigger monitor fails to open the initiation queue or when it fails to start the specified application. These messages normally indicate that user action is required to correct the condition.

Examples of both types of messages are provided in the figure. Trigger monitor status messages can be written to a standard output device to an error log.

If the trigger monitor detects an error, it can put a trigger message on the dead-letter queue. The trigger monitor adds a dead-letter header structure to the message. It uses a feedback code in the **Reason** field of the structure to explain why it put the message on the dead-letter queue.

Triggering and the dead-letter queue



If a trigger message is created but cannot be put on the initiation queue, the trigger message is put on the dead-letter queue

Implementing trigger messages and monitors

© Copyright IBM Corporation 2017

Figure 8-18. Triggering and the dead-letter queue

If the trigger monitor detects an error, it can put a trigger message on the dead-letter queue. The trigger monitor adds a dead-letter header structure to the message. It uses a feedback code in the **Reason** field of the structure to explain why it put the message on the dead-letter queue.

Application triggering program problems

- Make sure that the trigger monitor is running
- Check that the trigger monitor is monitoring initiation queue, not trigger queue
- Verify that applications are putting messages to trigger queue, not the initiation queue
- Try to start trigger program manually by using the string that is specified in the **AppId** property of process definition
- Verify that user ID that is used to start trigger monitor is authorized to access entire path to a runnable file

Implementing trigger messages and monitors

© Copyright IBM Corporation 2017

Figure 8-19. Application triggering program problems

If you are having problems with triggering, the figure lists diagnostic tips.

First, make sure that the trigger monitor is running.

- On Windows, use the Task Manager and look for the **runmqtrm** program.
- On UNIX or Linux, type: **ps -ef | grep runmqtrm**

It is common to confuse the trigger queue with the initiation queue. Verify that the trigger monitor is monitoring the initiation queue and putting messages on the trigger queue.

You must also ensure that the user ID that is used to start the trigger monitor can access the **runmqtrm** program.

Unit summary

- Configure IBM MQ to enable a trigger monitor
- Run the trigger monitors on distributed operating systems
- Determine the cause of a triggering failure

Implementing trigger messages and monitors

© Copyright IBM Corporation 2017

Figure 8-20. Unit summary

Review questions

1. Which of the following messages from the trigger monitor are reporting on normal activities?
 - A. Waiting for a trigger message
 - B. Initiation queue cannot be opened
 - C. Use of trigger monitor not authorized
 - D. Trigger monitor ended
2. True or False: A trigger monitor requires a process object.



Implementing trigger messages and monitors

© Copyright IBM Corporation 2017

Figure 8-21. Review questions

Write your answers down here:

- 1.
- 2.

Review answers

1. Which of the following messages from the trigger monitor are reporting on normal activities?
 - A. Waiting for a trigger message
 - B. Initiation queue cannot be opened
 - C. Use of trigger monitor not authorized
 - D. Trigger monitor ended

The answer is A and D.
2. True or False: A trigger monitor requires a process object.
The answer is True.



Implementing trigger messages and monitors

© Copyright IBM Corporation 2017

Figure 8-22. Review answers

Exercise: Implementing a trigger monitor

Implementing trigger messages and monitors

© Copyright IBM Corporation 2017

Figure 8-23. Exercise: Implementing a trigger monitor

In this exercise, you modify queue parameters to implement triggering. You define a process object that identifies the IBM MQ “echo” sample program and triggers the program when a message arrives on a queue. You use the “request” sample program to send messages to the queue and retrieve the responses.

Exercise objectives

- Apply triggering parameters to queues
- Start a trigger monitor
- Test triggering by using IBM MQ sample programs



Implementing trigger messages and monitors

© Copyright IBM Corporation 2017

Figure 8-24. Exercise objectives

See the *Course Exercises Guide* for detailed instructions.

Unit 9. Diagnosing problems

Estimated time

01:00

Overview

In this unit, you learn about the IBM MQ tools and utilities that you can use to help you diagnose problems in the IBM MQ network. The unit describes the IBM MQ trace mechanism, explains the contents of the `AMQERR01.LOG` file, and describes the First Failure Support Technology (FFST). It also provides problem determination hints and tips for some of the more common types of problems.

How you will check your progress

- Review questions
- Hands-on exercise

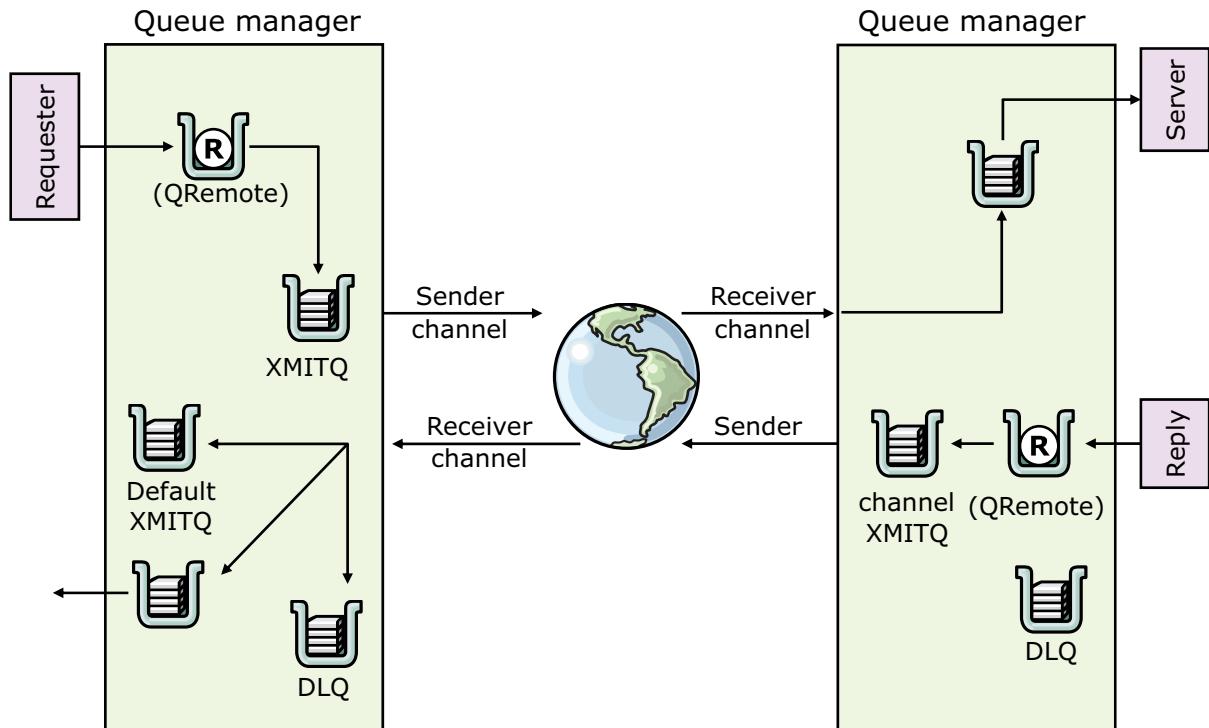
References

IBM Knowledge Center for IBM MQ V9

Unit objectives

- Determine the possible causes and locations of a missing message
- Analyze the error logs that IBM MQ generates
- Locate First Failure Support Technology (FFST) files on a system
- Use an IBM MQ trace to collect detailed information about IBM MQ operation
- Describe some of the more common problem types and how to approach initial problem determination
- Stop and remove a queue manager manually

Where is the message?



Diagnosing problems

© Copyright IBM Corporation 2017

Figure 9-2. Where is the message?

The figure shows a typical MQ request-reply application with two queue managers and queues. In this scenario, a message might end up in one of many places after a successful MQPUT, especially when it must cross systems and networks.

MQ does not lose messages by design, but messages can be lost for various reasons.

If the message is placed on the intended queue, no error message is apparent, but what if the message did not arrive at its intended target? Where is the message? If the message was a financial transfer, its loss no doubt causes worries, but you can check for messages in some standard places.

Missing message checklist

1. Is the message on the dead-letter queue?
2. Is the IBM MQ completion code “OK”?
3. Are there problems within the IBM MQ network?
4. Is the message persistent or not persistent?
5. Did the message expire?
6. Was the message committed?

Diagnosing problems

© Copyright IBM Corporation 2017

Figure 9-3. Missing message checklist

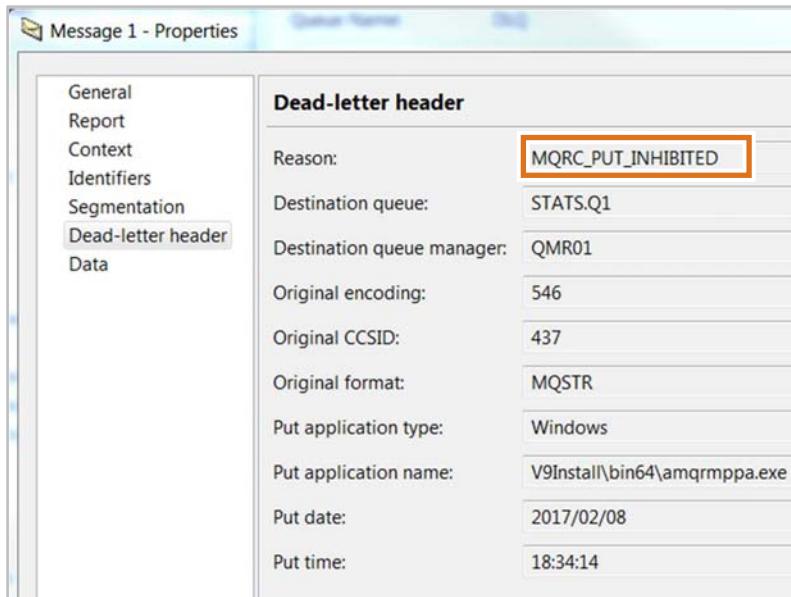
What if the message is not on the dead-letter queue and the message seems to be lost? What can you do next?

This figure lists the items to check when you investigate the possible loss of a message. Each of these items is described in more detail in this unit.

Finding the dead-letter reason (1 of 2)

Option 1:

Browse the message on the dead-letter queue in IBM MQ Explorer and view the dead-letter header in the **Dead-letter header** message properties



Diagnosing problems

© Copyright IBM Corporation 2017

Figure 9-4. Finding the dead-letter reason (1 of 2)

One of the first places to look for a message is the queue manager dead-letter queue.

The easiest way to find the dead-letter reason is by using the IBM MQ Explorer message browser.

Finding the dead-letter reason (2 of 2)

Option 2:

1. Use the `amqsbcg` sample program or RFHUtil SupportPac to browse the message on the dead-letter queue
2. Locate the dead-letter header structure ID (**DLH**)
3. Find the dead-letter reason code (third word in dead-letter header)

The screenshot shows a table of message data bytes. The first column is 'Message data bytes' showing hex values. The second column is 'Reason code (in hex)' with a yellow background. The third column is 'Structure ID' with a yellow background. Arrows point from the labels to their respective columns. The 'Reason code (in hex)' row has a circled value '03 08'. The 'Structure ID' row has a circled value 'DLH'.

Message data:	Reason code (in hex)	Structure ID
Message data bytes:	DH	
00000	44 4C 48 20 01 00 00 00- 03 08 0 00 53 54 41 54	
00010	53 2E 51 31 20 20 20 20-20 20 20 20 20 20 20 20	DLH
00020	20 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20	STAT
00030	20 20 20 20 20 20 20 20-20 20 20 20 51 4D 52 30	S.Q1
00040	31 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20	Queue
00050	20 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20	QMR0
00060	20 20 20 20 20 20 20 20-20 20 20 20 22 02 00 00	1 Queue manager
00070	B5 01 00 00 4D 51 53 54--52 20 20 20 0B 00 00 00	"....
00080	56 39 49 6E 73 74 61 6C--6C 5C 62 69 6E 36 34 5C	...MQSTR \....\
00090	61 6D 71 72 6D 70 70 61--2E 65 78 65 32 30 31 37	V9Install\bina64\ amqrmpaa.exe2017

4. Convert the reason code from hex to decimal
5. Use the `mqrc` command dead-letter reason code to determine why the message was placed on the dead-letter queue

Figure 9-5. Finding the dead-letter reason (2 of 2)

If you do not have access to IBM MQ Explorer, you might have to decode the dead-letter header (DLH) manually to determine the reason code. The figure lists the steps for determining why a message was put on the dead-letter queue.

The dead-letter reason provides a code that identifies the reason that the message was put on the dead-letter queue. It reveals a reason code (MQRC_*) or a feedback code (MQFB_*).

After you determine the dead-letter reason code, you can use it to determine the reason why the message was placed on the dead-letter queue.

Checking IBM MQ reason codes

- Queue manager or exit program returns completion code and reason code to indicate success or failure of MQI call
- Use `mqrc` command to display information about return codes if completion code is `MQCC_WARNING` or `MQCC_FAILED`

Example:

```
C:\> mqrc 2051
2051 0x00000803 MQRC_PUT_INHIBITED
```

- Developers should include logic in applications to ensure that the MQI call is successful

Example:

```
IF REASON IS NOT EQUAL TO MQRC-NONE
  IF REASON IS EQUAL TO MQRC_Q_FULL
    DISPLAY 'Queue contains max number of messages'
  ELSE
    DISPLAY 'MQPUT ended with reason code ' REASON
  END-IF
END-IF
```

Figure 9-6. Checking IBM MQ reason codes

The queue manager or exit program returns a **completion code** and a **reason code** for each MQI to indicate the success or failure of the call.

If the completion code is not `MQCC_OK`, it is possible that the message never made it to the target queue or topic.

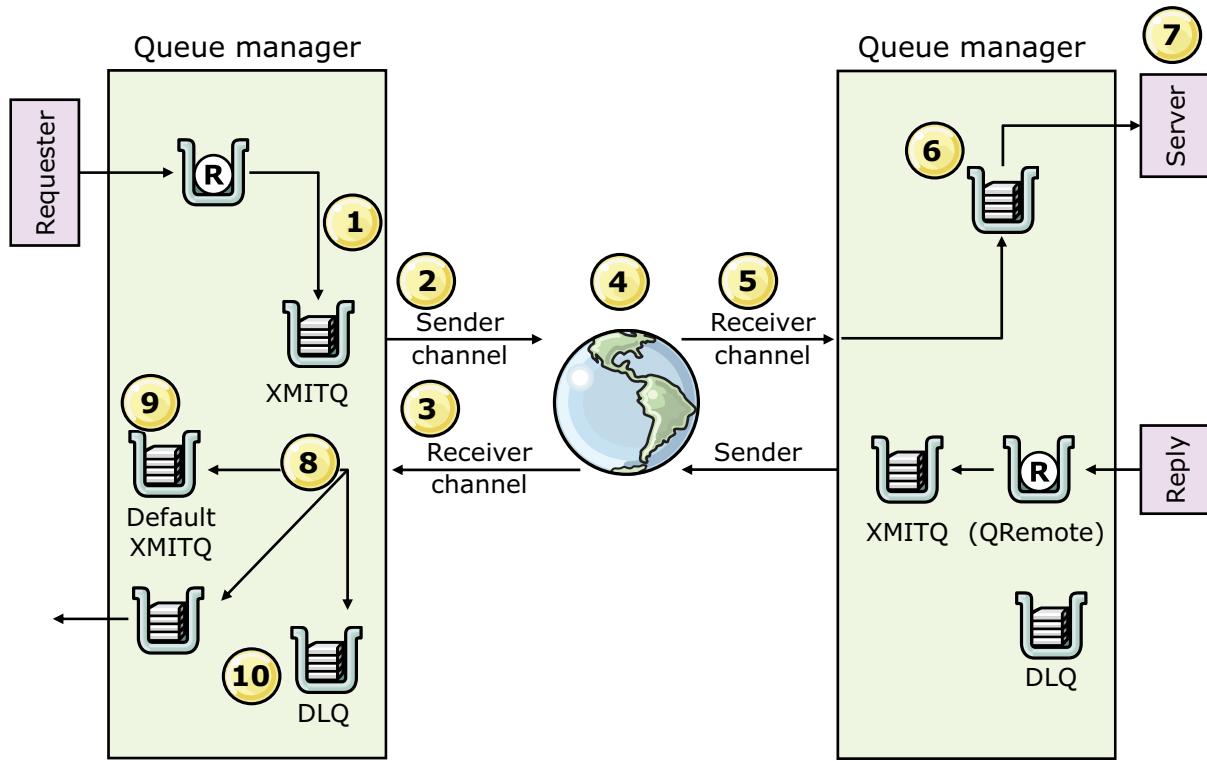
If the completion code is `MQCC_WARNING` or `MQCC_FAILED`, use the `mqrc` control command to check the reason code for more information about the problem.

An application that is retrieving a message should always check the completion code or the reason code. The figure shows sample logic that tests the reason code. If the reason code is not equal to `MQRC-NONE`, the sample logic displays an appropriate error message.

IBM Training



Checking for IBM MQ network problems



Diagnosing problems

© Copyright IBM Corporation 2017

Figure 9-7. Checking for IBM MQ network problems

The next step for determining the reason for a missing message is to check the MQ network for problems.

This figure shows a typical application with two queue managers and multiple queues. Potential failure points are highlighted in the figure. The following list provides possible reasons for the failure at those points.

1. Incoming messages were not put to the transmission queue because it is “put” or “get” disabled, or not defined as a transmission queue.
2. The sender channel is not running or was not triggered to start, or cannot start because the transmission queue is “get” disabled or at the maximum channel (MAXCHL) limit.
3. The channel initiator is not running.
4. The channel failed to start because of a network problem.
5. The receiver channel did not start because the listener is not running, or the receiver failed to put to a target queue because the target queue is “put” disabled or full. Another possible failure is that the channel initiator or event of the queue manager is not running.
6. The server program did not get the message because the target queue was “get” disabled.
7. The server program is not started or failed to be triggered.

8. The receiver channel received a message for another remote queue manager, but no transmission queue is defined for it, so the receiver channel put the message to the default transmit queue.
9. No channel is defined to serve the default transmission queue.
10. The receiver channel received an inbound message for an unknown local target queue, or the local target queue was full, so it was put on the dead-letter queue.

The same failure points are present in both directions; this list is not complete.

Checking message persistence

- If queue manager restarts while non-persistent message is on the queue, message is lost
 - Use IBM MQ Explorer or **DISPLAY QUEUE** command to check **Default persistence** property (**DEFPSIST**) on the queue
 - Use IBM MQ Explorer message browser to check message persistence that the PUT application defines

The screenshot shows the 'Message browser' window with the following details:

Position	Persistence	Message data	Put application name	Put date
1	Persistent	Test Mesesage 1 (default)	re MQ\java\jre\bin\javaw.exe	Jun 2
2	Not persistent	[header]□□* Input parameters for MQPut2 program *□□*□...	s\Downloads\jh03\rfhutil.exe	Jun 2

Diagnosing problems

© Copyright IBM Corporation 2017

Figure 9-8. Checking message persistence

Restarting a queue manager with nonpersistent messages on the queues is one of the most common reasons that messages are lost.

If the queue manager is restarted while a nonpersistent message is on the queue, the message is discarded. This behavior is standard for nonpersistent messages. They have much lower memory and storage requirements because they are not stored in a log.

Many administrators assume that because the default persistence (**DEFPSIST**) property of the target queue is set to **YES**, all messages on that queue are persistent. Persistence is a message attribute, not a queue attribute. If the application specifies a message with **Persistence = no** in the MQMD, the message setting overrides the default persistence setting of the target queue. If the queue manager is restarted while a nonpersistent message is on the queue, it is lost.

Message persistence can be checked by using the MQ Explorer Queue Browse facility or the **amqsbcg** sample program to browse a queue.

Checking message expiration

- Application that puts the message on the queue, sets the expiration
- Delays in processing can cause messages to expire before intended
- Check **Expiry** property on message by using IBM MQ Explorer message browser

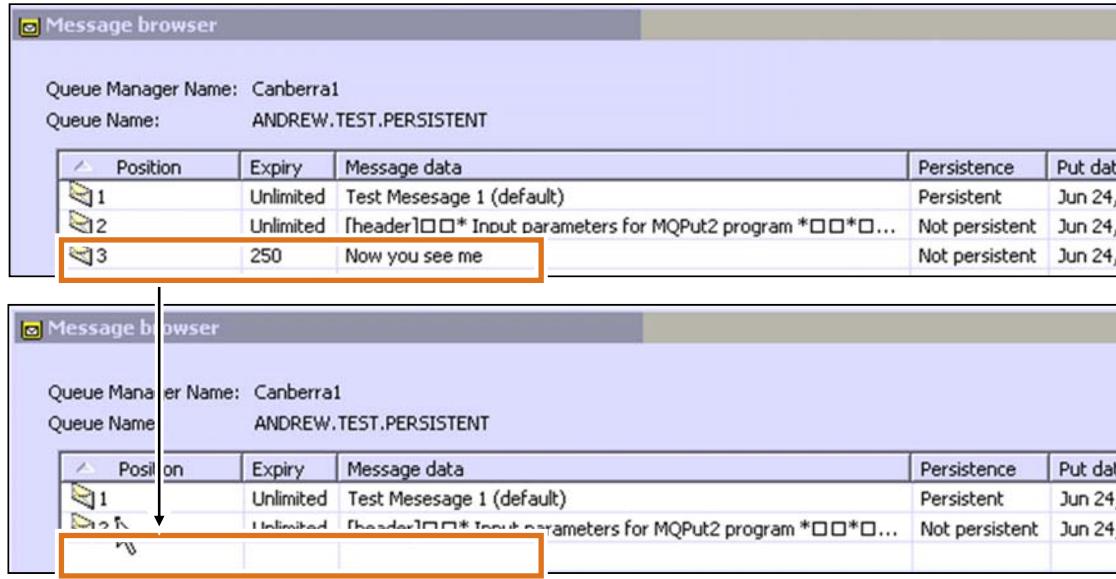


Figure 9-9. Checking message expiration

Message expiration is another reason that messages might seem to be lost.

Applications can explicitly set a life span of a message. Setting a life span is typically the case with time-sensitive data such as quotations.

Messages that expire on a loaded queue (a queue that was opened) are automatically removed from the queue within a reasonable time after their expiry.

The message expiry time is in tenths of second and represents the amount of time that remains (time to live). In the figure, the top image shows that the third message on this queue was put with an expiration time of 30 seconds (300 tenths of a second).

The bottom image shows the contents of the queue after the message expired; it is no longer on the queue.

Sometimes delays in message processing can cause messages to expire before they were intended. Handle this behavior in the application design.

Uncommitted messages

- Messages that are put to a queue under transactional control are not available to other applications until transaction is either committed or rolled back
- Use **DISPLAY QSTATUS** command to view information about queue and processes (handles) attached to the queue
 - Current queue depth property (**CURDEPTH**) includes uncommitted messages, even though they are not available
 - Uncommitted messages (**UNCOM**) property indicates whether some messages are not committed
 - Open processes (**OPPROCS**) property identifies number of processes that are attached to the queue

[Diagnosing problems](#)

© Copyright IBM Corporation 2017

Figure 9-10. Uncommitted messages

Messages that are put to a queue under transactional control are not available to another application until the transaction is either committed or rolled back.

In this scenario, it can seem that messages are lost if the number of messages that are processed by an application does not match the current depth count that is seen on a previous display. More often, the problem seems to be that messages are irretrievable.

To check for uncommitted messages:

1. Enter the MQSC **DIS QSTATUS** command with the **TYPE(HANDLE)** option to discover information about the processes that are attached to the queue.

Example: **DIS QSTATUS(TESTQ1) TYPE(HANDLE) ALL**

2. Check the uncommitted messages properties (**UNCOM**) to see whether some messages on the queue are uncommitted.

Displaying the queue status

- Use the MQSC command **DISPLAY QSTATUS** to display status of one or more queues and the processes (handles) that are accessing queues

```
DISPLAY QSTATUS(TESTQ1) TYPE(QUEUE)
AMQ8450: Display queue status details.
  QUEUE(TESTQ1)
  CURDEPTH(4)
  :
  OPPROCS(1)
  UNCOM(YES)
```

TYPE(QUEUE)
IPPROCS(0)
QTIME(,)

```
DISPLAY QSTATUS(TESTQ1) TYPE(HANDLE)
AMQ8450: Display queue status details
  QUEUE(TESTQ1)
  APPLTAG(c:\WMQ\bin\amqspput.exe)
  :
  OUTPUT(YES)
  QMURID(0.0)
  TID(1)
  URID(XA_FORMATID[ ] XA_GTRID[ ] XA_BQUAL[ ])
  URTYPE(QMGR)
```

TYPE(HANDLE)
APPLTYPE(USER)
PID(10880)
SET(NO)
USERID(slw@localhost)

Figure 9-11. Displaying the queue status

This figure provides examples of the **DISPLAY QSTATUS** command to determine whether the queue contains uncommitted messages and the processes that are connected to the queue.

In the first example in this figure, the MQSC **DISPLAY QSTATUS** command is run against the queue by using the **TYPE(QUEUE)** option. The results of the command show that the queue has a current depth (**CURDEPTH**) of 4. One or more of these messages are uncommitted as **UNCOM(YES)** indicates. The **DISPLAY QSTATUS** command does not indicate the number of these messages that are not available.

In the second example, the **DISPLAY QSTATUS** is run against the queue with the **TYPE(HANDLE)** option. The results of this command show that the application **amqspput** is attached to the queue.

First Failure Support Technology (FFST)

- Records “unexpected” errors
 - Detects and reports software events, such as internal queue manager failures
 - Collects information about software events
 - Generates data to help analyze software events, for example, probe IDs
- If an error occurs:
 - Note a description of the problem
 - Look for any related error log entries
 - Identify any FFST reports

Diagnosing problems

© Copyright IBM Corporation 2017

Figure 9-12. First Failure Support Technology (FFST)

First Failure Support Technology (FFST) for MQ provides information about events that, if an error occurs, can help IBM support personnel to diagnose the problem.

MQ uses FFST for the following tasks:

- Detect and report software events, for example, internal queue manager failures
- Collect information about software events, for example, dumps of storage
- Generate data to help analyze software events, for example, probe IDs

For example, if you do not set certain kernel parameters to the minimum values on a UNIX system before you start to use MQ, you might experience frequent FFST reports that are related to shared memory.

The information about an event is contained in an FFST file. FFST files do not always indicate an error. An FFST might be informational.

FFST files

- Contain information about a configuration problem with the system or an IBM MQ internal error
- Files are named **AMQnnnn.mm.FDC**
 - **nnnn** is the ID of the process that reports the error
 - **mm** starts at 0
 - If the full file name exists, this **mm** value increments by one until a unique FFST file name is found
- Location of FDC files:
 - Windows: **C:\ProgramData\IBM\MQ\errors**
 - UNIX and Linux: **/var/mqm/errors**

Figure 9-13. FFST files

In MQ, FFST files have a file type of FDC.

FFST files are named **AMQnnnnn.mm.FDC**, where **nnnnn** is the process ID reporting the error and **mm** is a sequence number, normally 0.

An instance of a process writes all FFST information to the same FFST file. If multiple errors occur while a process runs, an FFST file can contain many records. These records indicate either a configuration problem with the system or an MQ internal error.

On Windows, when a process writes an FFST record, it also sends a record to the Event Log. The record contains the name of the FFST file to help with automatic problem tracking. The Event Log entry is made at the application level.

FFST report example

```

Date/Time      :- Thu May 22 2016 07:19:02 Pa
UTC Time      :- 1400768342.384000
UTC Time Offset :- 60 (Pacific Daylight Time)
Host Name     :- MyHost
Operating System :- Windows Server 2008 R2 Serv
PIDS          :- 5724H7251
LVLS          :- 9.0.0.0
Product Long Name :- MQ for Windows (x64 platfor
Vendor         :- IBM
O/S Registered :- 1
Data Path      :- C:\ProgramData\IBM\MQ
Installation Path :- C:\Program Files\IBM\MQ
Installation Name :- Installation1 (1)
License Type   :- 
→ Probe Id      :- AD004020
Application Name :- MQM
→ Component    :- adhOpen
SCCS Info      :- F:\build\slot1\p000_P\src\lib\lqm\amqadho0.c,
. . .
UserID         :- MUSR_MQADMIN
→ Process Name :- C:\Program Files\IBM\MQ\bin\amqzlaa0.
. . .
QueueManager   :- QML01
UserApp        :- FALSE
. . .
→ Major Errorcode :- arcE_OBJECT_MISSING
Minor Errorcode :- OK
Probe Type     :- INCORROUT
Probe Severity :- 2
→ Probe Description :- AMQ6125: An internal MQ error has occurred.

```

Each FFST report contains various items of useful information:

- Probe ID that identifies where in code the error was detected
- Date and time the error occurred
- Any associated error message
- Variable number of memory dumps that include the function stack and trace history

Diagnosing problems

© Copyright IBM Corporation 2017

Figure 9-14. FFST report example

This figure shows an FFST report (an FDC file) that reports that an internal MQ error occurred on a Windows MQ system.

Each FFST report contains various items of useful information.

- **Probe Id** is a unique error code that identifies where in the code the error is detected.
- **Component** identifies the function that failed.
- **Process Name** is the name of the process that is running at the time of the failure.
- **Major Errorcode** is the named reason for the failure.
- **Probe Description** contains the externalized message ID and meaning, which is written to the error log.

IBM MQ error logs

- Captures messages that concern:
 - Operation of IBM MQ
 - Any queue managers that you start
 - Error data that comes from the channels that are in use
- Location depends on whether queue manager name is known and whether error is associated with a client
 - If queue manager name is known and queue manager is available:

Windows: C:\ProgramData\IBM\MQ\Qmgrs\QMgrName\errors\AMQERR01.LOG
 UNIX and Linux: /var/mqm/qmgrs/QMgrName/errors/AMQERR01.LOG
 - If an error occurs with a client application:

Windows: C:\ProgramData\IBM\MQ\errors\AMQERR01.LOG
 UNIX and Linux: /var/mqm/errors/AMQERR01.LOG
- Errors subdirectory can contain up to three error log files:
AMQERR01.LOG, **AMQERR02.LOG**, **AMQERR03.LOG**

[Diagnosing problems](#)

© Copyright IBM Corporation 2017

Figure 9-15. IBM MQ error logs

MQ error logs contain messages about the operation of MQ. Error messages identify normal errors, typically caused by users, such as the use of a nonvalid parameter on a control command.

Error messages are written to an error log file that is called **AMQERR01.LOG**. A separate error log file with this name exists in each of the error directories that are described in the figure.

The errors subdirectory can contain up to three error log files. When the error log file **AMQERR01.LOG** is full, its contents are copied to **AMQERR02.LOG** and **AMQERR01.LOG** is then reused. Before the copy, **AMQERR02.LOG** is copied to **AMQERR03.LOG**. The previous contents of **AMQERR03.LOG** if any, are discarded. In this way, **AMQERR02.LOG** and **AMQERR03.LOG** maintain a history of error messages.

On Windows, also examine the Windows application Event Log for relevant messages.

Example error log contents

```
7/20/2016 09:01:40 - Process(13492.1) User(userlibm) Program(endmqsvc.exe)
                     Host(IBM-12345) Installation(Installation1)
                     VRMF(9.0.0.0)
AMQ7291: The MQ service for installation 'Installation1' failed to end with
error 1051.

EXPLANATION:
The attempt to end the MQ service (amqsvc.exe) for installation
'Installation1' failed, the error from the operating system was 1051.

The formatted message text for error 1051 is 'A stop control has been sent
to a service that other running services are dependent on.' (if blank this
indicates that no message text was available).

ACTION:
Check that the service named 'IBM MQ (Installation1)' has been properly
configured and is enabled, then re-issue the command.
----- endmqsvc.c : 419 -----
```

Figure 9-16. Example error log contents

This figure shows an example of the contents of an error log. Each log entry is identified with a date and time stamp, the host, and MQ installation. It also includes the error message and the action to take.

IBM MQ AMQ messages

- IBM MQ diagnostic messages are grouped according to the part of IBM MQ from which they originate
 - AMQ4xxx: User interface messages (Windows and Linux)
 - AMQ5xxx: Installable services
 - AMQ6xxx: Common services
 - AMQ7xxx: IBM MQ product
 - AMQ8xxx: Administration
 - AMQ9xxx: Remote
- Use the `mqrc` command to display information about AMQ messages

[Diagnosing problems](#)

© Copyright IBM Corporation 2017

Figure 9-17. IBM MQ AMQ messages

MQ AMQ messages are diagnostic messages that are grouped according to the part of MQ from which they originate.

Each message contains:

- Four-digit decimal code.
- Summary of the message.
- Message severity:
 - 0: Information
 - 10: Warning
 - 20: Error
 - 30: Severe error
 - 40: Stop error
 - 50: System error
- An explanation of the message with further information.
- The response that is required from the user. In some cases, particularly for information messages, the response might be “none”.

AMQ message example

```
C:\>mqrc AMQ5005

 536901397 0x20005005 zrcX_PLUG_UNEXCEPTED
MESSSAGE:
Unexpected error
EXPLANATION:
An unexpected error occurred in an internal function of the product.
ACTION:
Save any generated output files and use either the MQ Support site:
//www.ibm.com/software/integration/wmq/support/ or IBM Support
Assistant (ISA): http://www.ibm.com/software/support/isa/, to see
whether a solution is already available. If you are unable to find a
match, contact your IBM support center
C:>
```

Figure 9-18. AMQ message example

This figure shows the example of an AMQ message that is displayed by running the `mqrc` control command with the message number.

The message includes the explanation of the message and an action to take.

Application activity trace

- Can be used to:
 - Track messages
 - Analyze or model application behavior and outage impact
 - Audit options that applications use
- Generation of detailed MQI activity
 - If configured, each MQI operation produces an activity record
 - Multiple records can be bundled into an activity trace message that is based on time or record count thresholds
 - Activity records are grouped in PCF format
 - Activity records are written to well-defined topics
- Controlled by:
 - **ACTVTRC (ON|OFF)** queue manager attribute
 - **AllActivityTrace** settings in **mqat.ini**
 - IBM MQ Explorer queue manager **Online monitoring** properties

Diagnosing problems

© Copyright IBM Corporation 2017

Figure 9-19. Application activity trace

An application activity trace reports on all the MQI operations from an application. The advantage of an activity trace is that applications and their relationships and resources can be analyzed without inspecting the application source code.

Activity trace runs “inside” the queue manager so it has access to more than just the MQI parameters that the application passes. Other information that is reported includes the queues that the application uses.

The application activity trace messages are sent to the SYSTEM.ADMIN TRACE.ACTIVITY.QUEUE queue. Like other events, it is possible to redefine the event queue to be a topic alias so that multiple consumers can work with these messages.

The output of the activity trace includes a set of PCF events, where each event holds details about multiple MQI calls.

You can specify whether activity trace information is collected by setting the queue manager property **ACTVTRC** to **ON**. The **ACTVCONO** queue manager property specifies whether applications can override the settings of the **ACTVTRC** queue manager property by using the MQI connection options.

You can also set the activity trace properties and enable an activity trace in MQ Explorer and in the **mqat.ini** file.

The mqat.ini file

- Created in queue manager data directory when queue manager is created with default contents
- Holds the default levels of tracing for any trace rules
- Can be used to define specific rules for tracing or not tracing any connections by using an application name
- Use the `setmqini` command to configure trace levels in the AllActivityTrace stanza

```
setmqini -m QMgrName -s AllActivityTrace -k KeyName -v Value
```

Figure 9-20. The mqat.ini file

The `mqat.ini` file defines the level and frequency of reporting activity trace data. The file also provides a way to define rules to enable and disable activity trace based on the name of an application.

The `mqat.ini` file is stored the queue manager data directory.

When the `mqat.ini` file is modified, newly created MQ connections are processed according to the modified version. Existing connections continue to use the previous version unless the queue manager parameters are altered.

The mqat.ini file

```

#*****#
#* Module Name: mqat.ini *                                #
#* Type : IBM MQ queue manager configuration file *#
# Function : Define the configuration of application activity *#
#* trace for a single queue manager. *#
#*****#


# Global settings stanza, default values
AllActivityTrace:
  ActivityInterval=1
  ActivityCount=100
  TraceLevel=MEDIUM
  TraceMessageData=0
  StopOnGetTraceMsg=ON
  SubscriptionDelivery=BATCHED

# Prevent the sample activity trace program from generating data
ApplicationTrace:
  ApplName=amqsact*
  Trace=OFF

```

Figure 9-21. The mqat.ini file

The `mqat.ini` file follows the same stanza key and parameter-value pair format as the `mqm.ini` and `qm.ini` files.

The `AllActivityTrace` stanza specifies the level and frequency of reporting activity trace data by default for all activity trace.

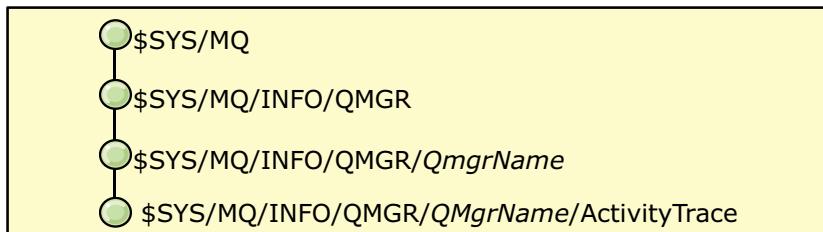
The `TraceLevel` identifies the amount of parameter detail to trace for each operation.

Each optional `ApplicationTrace` stanza defines a rule for the trace behavior for one or more connections, based on matching the application name of the connections to the rule.

The figure shows the first part of the `mqat.ini` file that contains the configuration information for an activity trace for a single queue manager.

Subscribing to activity trace data

- IBM MQ decouples publishers from subscribers
 - Queue manager holds a view of all the topic strings in a hierarchical construct that is known as the *topic tree*
- Administrator uses MQSC and IBM MQ Explorer to:
 - Enable publish/subscribe in the queue manager
 - Define queue manager publish/subscribe properties
 - Define and monitor topics and subscriptions
- IBM MQ publishes application activity trace from queue manager to \$SYS/MQ topic branch
 - Access to \$SYS/MQ is restricted to administrators by default
 - Others can subscribe to a subset of the data



Diagnosing problems

© Copyright IBM Corporation 2017

Figure 9-22. Subscribing to activity trace data

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 anything about each other for the information to be sent and received.

MQ publish/subscribe removes the need for your application to know anything about the target application, and MQ handles the distribution of that information. You can use MQSC commands and MQ Explorer to manage MQ topics and subscriptions.

Publish/subscribe support in MQ is taught in detail in courses WM213/ZM213, *IBM MQ V9 Advanced System Administration*.

You can subscribe to an IBM MQ system topic to collect application activity trace information.

In IBM MQ V9, application activity trace is published to the \$SYS/MQ topic branch. System administrators and analysts can then subscribe to the topics to access application trace data.

The figure shows the structure of the \$SYS/MQ topic. As shown in the figure, a \$SYS/MQ/INFO/QMGR topic branch exists for each queue manager. Each queue manager topic branch contains a subbranch for activity trace and for resource monitoring.

Activity trace topics and subscriptions

- Administrator can subscribe to IBM MQ activity trace topic to access trace data:

\$SYS/MQ/INFO/QMGR/<QmgrName>/ActivityTrace/<ResourceType>/<ResourceIdentity>

- Types of traced resource (`ResourceType`)
 - Application name (ApplName)**: Any connection from a matching application name
 - Channel name (ChannelName)**: Either any connection on a matching SVRCONN channel or any PUTs or GETs on a queue manager channel
 - Connection ID (ConnectionId)**: ID from the “application connections” in IBM MQ Explorer, or the CONN value concatenated with the EXTCONN value from `DISPLAY CONN`

Figure 9-23. Activity trace topics and subscriptions

You can subscribe to an MQ system topic string that represents the activity to trace.

Subscribing to an MQ activity trace topic automatically generates activity trace data messages and publishes them to the subscription destination queue. If you delete the subscription, the generation of activity trace data stops for that subscription.

A subscription can trace activity on one of the following resources:

- A specified application
- A specified IBM MQ channel
- An existing IBM MQ connection

You can create multiple subscriptions, with different or the same topic strings. Where you create multiple subscriptions with the same system activity trace topic strings, each subscription receives a copy of the activity trace data. This duplication of trace data might have adverse performance implications.



Important

Enabling any level of activity trace can adversely affect performance. The more subscriptions, or the more resources that are subscribed to, the greater the potential for adverse effects on performance.

To minimize the performance impact of collecting activity trace, the data is written to messages and delivered to the subscriptions asynchronously from the application activity itself. Often, multiple operations are written to a single activity trace data message. The asynchronous operation can introduce a delay between the application operation and the receipt of the trace data that records the operation.

Example activity trace topic strings

- Topic string for an application that is named **amqsput**:

```
$SYS/MQ/INFO/QMGR/QMGR1/ActivityTrace/AppName/amqsput.exe
```

- Topic string for a channel:

```
$SYS/MQ/INFO/QMGR/QMGR1/ActivityTrace/ChannelName/SYSTEM.DEF.SVRCONN
```

- Topic string for a specific existing connection:

```
$SYS/MQ/INFO/QMGR/QMGR1/ActivityTrace/ConnectionId/  
414D5143514D475231202020202020206B576B5420000701
```

- Topic string to trace all channels on queue manager QMGR1:

```
$SYS/MQ/INFO/QMGR/QMGR1/ActivityTrace/ChannelName/#
```

Figure 9-24. Example activity trace topic strings

This figure provides examples of activity trace topic strings for a queue manager that is named QMGR1.

Activity trace sample program

- IBM MQ sample program **amqsact** formats application activity trace messages
 - Default mode reads messages from SYSTEM.DATA.TRACE.ACTIVITY.QUEUE
 - Dynamic mode subscribes to the system topic

```
$ amqsact -m QMGR1 -a amqspput -w 60
Subscribing to the activity trace topic:
'$SYS/MQ/INFO/QMGR/QMGR1/ActivityTrace/AppName/amqspput'
MonitoringType: MQI Activity Trace
...
QueueManager: 'QMGR1'
ApplicationName: 'amqspput'
Application Type: MQAT_UNIX
...
=====
Tid Date Time Operation CompCode MQRC HObj (ObjName)
001 2016-04-14 09:56:53 MQXF_CONNX MQCC_OK 0000 -
001 2016-04-14 09:56:53 MQXF_OPEN MQCC_OK 0000 2 (QUEUE1)
001 2016-04-14 09:56:53 MQXF_PUT MQCC_OK 0000 2 (QUEUE1)
001 2016-04-14 09:56:53 MQXF_PUT MQCC_OK 0000 2 (QUEUE1)
001 2016-04-14 09:56:53 MQXF_CLOSE MQCC_OK 0000 2 (QUEUE1)
001 2016-04-14 09:56:53 MQXF_DISC MQCC_OK 0000 -
```

Diagnosing problems

© Copyright IBM Corporation 2017

Figure 9-25. Activity trace sample program

You can use the MQ **amqsact** sample program to format application activity trace messages.

The compiled program is located in the samples directory:

- On Linux and UNIX, the samples directory is `MQ_INSTALLATION_PATH/samp/bin`
- On Windows, the samples directory is `MQ_INSTALLATION_PATH\tools\c\Samples\Bin64`

By default, **amqsact** processes messages on SYSTEM.ADMIN.TRACE.ACTIVITY.QUEUE. You can override this behavior by specifying a queue name or topic string.

You can also control the trace period (**-s** and **-e**) that is displayed and specify whether the activity trace messages are removed or retained after display (**-b**).

You can also specify the topic string (**-t**), application name (**-a**), channel name (**-c**), or connection ID (**-i**).

The example displays the activity trace data for the **amqspput** application for queue manager QMGR1. The command also specifies a wait of 60 seconds (**-w 60**). If no trace messages appear in the specified period, **amqsact** exits.

Tracing IBM MQ components

- Extra information might be needed to find a problem
 - Files can be large
 - Time or component might cause limits
- Can also trace MQI, which is a useful aid for application debugging
- Trace commands:
 - Start trace: `strmqtrc`
 - Stop trace: `endmqtrc`
 - Format trace (Linux and UNIX): `dspmqtrc`



Always stop all tracing when not needed for troubleshooting

[Diagnosing problems](#)

© Copyright IBM Corporation 2017

Figure 9-26. Tracing IBM MQ components

In some cases, it might be necessary to generate a component trace. For example, IBM Support might request that you re-create a problem with an MQ component trace enabled. It is also possible to limit tracing to the MQI to help diagnose application problems.

The control command to start a trace is: `strmqtrc`

The control command to end a trace is: `endmqtrc`

The control command to format the trace results is: `dspmqtrc`



Important

The files that the component trace produces can be large so limit a trace by time or trace only specified components.

Trace commands

- Trace files are written to:
 - Created in the `MQ_DATA_PATH/trace` directory on Windows
 - Created in the `/var/mqm/trace` directory on UNIX and Linux
 - Delete or relocate old trace files before beginning a new trace

 - Start trace
 - For every IBM MQ process:
 - For one queue manager:
 - High detail trace for one queue manager:
- `strmqtrc -e`
`strmqtrc -m MY.QMGR`
- `strmqtrc -t all -t detail -m MY.QMGR`
- Stop all tracing
 - Format trace files on Linux and UNIX
 - Format wrapping trace files
- `endmqtrc -a`
`dspmqtrc *.TRC`
`dspmqtrc *.TRC *.TRS`



Stop all tracing when not needed for troubleshooting

Diagnosing problems

© Copyright IBM Corporation 2017

Figure 9-27. Trace commands

The trace files are written to specific locations on the file system.

The figure shows some examples of starting an MQ component trace for every process, and for one queue manager. It also shows examples of how you can modify the commands to control the trace detail and limit the size of the trace file.

Trace files are named `AMQppppp.qq.TRC` where `ppppp` is the process ID of the process that reports the error and `qq` is a sequence number.

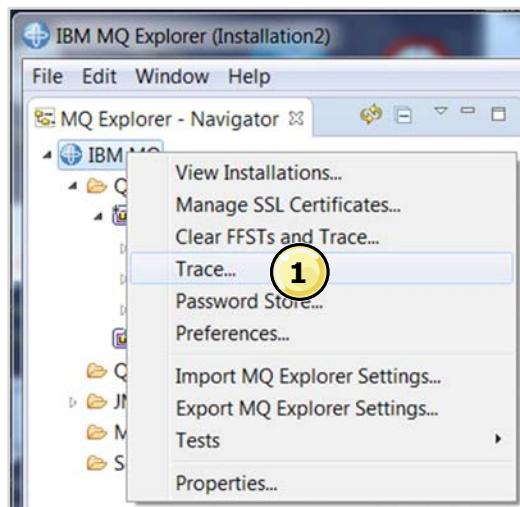
SSL trace files are named `AMQ.SSL.TRC` and `AMQ.SSL.TRC.1`. You cannot format SSL trace files; send them unchanged to IBM Support.

The trace formatter program, `dspmqtrc`, converts binary trace files into readable files that are named `AMQppppp.FMT` where `ppppp` is the process identifier that created the file.

If you specify a wrapping trace, then each time a trace (`.TRC`) file reaches the specified size limit, MQ renames it to use a `.TRS` extension and starts a new trace file. The trace formatter can convert both files to a single formatted file, but only if you format the `.TRC` and `.TRS` files at the same time, as shown in the last example in the figure.



Enabling trace in IBM MQ Explorer



- In IBM MQ Explorer, trace is limited:
 - Trace all components
 - High-level detail



Diagnosing problems

© Copyright IBM Corporation 2017

Figure 9-28. Enabling trace in IBM MQ Explorer

You can enable a component trace in MQ Explorer, but the options for the trace are limited to **Trace all components** and **High level detail**.

Example trace on Windows

```

Process : C:\Program Files\IBM\MQ\Tools\c\Samples\Bin64\amqsget.exe (64-bit)
Arguments :
Host : IBM
Operating System : Windows 7 Professional x64 Edition, Build 7601: SP1
Product Long Name : IBM MQ for Windows (x64 platform)
Version : 9.0.0.0      Level : p900-L160512.4
O/S Registered : 1
Data Path : C:\ProgramData\IBM\MQ
Installation Path : C:\Program Files\IBM\MQ
Installation Name : Installation2 (2)
License Type : Production
UTC Date : 2016/10/25: Time : 14:45:1.580
LCL Date : 2016/10/25: Time : 10:45:1.580  Eastern Standard Time
QueueManager : QM01

Counter TimeStamp          PID.TID   Ident       Data
=====
000006EF 10:45:01.565102  7200.1     :    !! - Thread stack (from
mqe.dll)
000006F0 10:45:01.565113  7200.1     :    !! - -> MQCONN
000006F1 10:45:01.565119  7200.1     :    !! - -> trmzstMQCONN
000006F2 10:45:01.565124  7200.1     :    !! - -> zstMQCONN
000006F3 10:45:01.565128  7200.1     :    !! - -> zstMQConnect
000006F4 10:45:01.565131  7200.1     :    !! - -> zstInitCS
000006F5 10:45:01.565136  7200.1     :    !! - -> xcsInitialize
000006F6 10:45:01.565142  7200.1     :    !! - ->
xcsInitGlobalSecurityData
  
```

Diagnosing problems

© Copyright IBM Corporation 2017

Figure 9-29. Example trace on Windows

The figure shows an example of a trace on Windows

Use the **dsprmqtrc** control command to format the trace report before viewing.

For example, to format all trace files in the current directory, type: **dsprmqtrc *.TRC**

Configuration files and problem determination

- Errors can stop a queue manager from being found
Example: **QUEUE MANAGER UNAVAILABLE**
- What to check
 - Configuration files exist
 - Configuration files have appropriate permissions
 - IBM MQ configuration file (or Windows registry) references queue manager with the correct information for locating its files

Figure 9-30. Configuration files and problem determination

If you receive an error message that indicates that the queue manager is not available, the cause might be something simple. For example, the queue manager is not started or an application specified an incorrect queue manager name.

If the cause is not obvious, check the following configuration options:

- Ensure that the MQ configuration file **mqm.ini** exists.
- Ensure that the MQ configuration file has the appropriate permissions. For example, on UNIX the MQ configuration file should have the following permissions:

```
-rwxrwxr-x 1 mqm mqm 1371 Sep 17 14:32 /var/mqm/mqm.ini
```

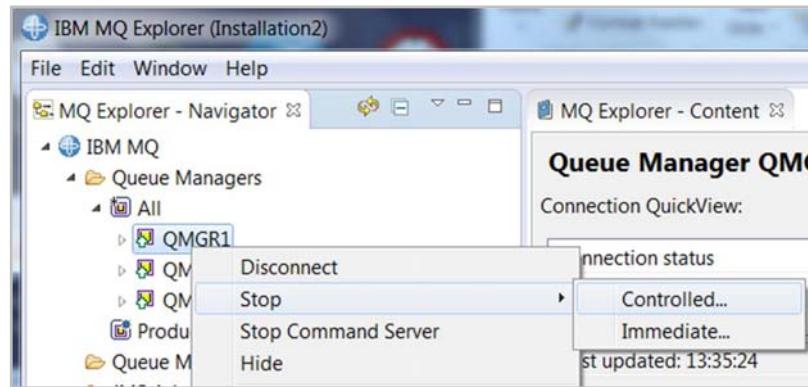
- Ensure that the MQ configuration file references the queue manager and has the correct information for locating the files that are associated with it.

Errors in a configuration file can typically prevent a queue manager from being found.



Stopping a queue manager manually

- Use `endmqm` control command or IBM MQ Explorer
 - Might take time
 - Give it a chance



- Only if no other option, find and stop the processes that are still running

Figure 9-31. Stopping a queue manager manually

During normal system administration, it might be necessary to stop the queue manager. You can stop the queue manager by using the `endmqm` control command or in the MQ Explorer.

If, for some reason, the queue manager does not stop, it might be necessary to force the queue manager to shut down by stopping the queue manager process. Use this option only when all other options to stop the queue manager failed.

Stopping a queue manager manually on Windows

1. List IDs of running IBM MQ processes by using Windows Task Manager
2. Use Windows Task Manager or `taskkill` command to stop running processes in this order:

AMQZMUC0	Critical process manager
AMQZXMA0	Execution controller
AMQZFUMA	OAM process
AMQZLAA0	LQM agents
AMQZLSA0	LQM agents
AMQZMUFO	Utility Manager
AMQZMGRO	Process controller
AMQZMUR0	Restartable process manager
AMQFQPUB	Publish/subscribe process
AMQFCXBA	Broker worker process
AMQRMPPA	Process pooling process
AMQCRSTA	Non-threaded responder job process
AMQCRS6B	LU62 receiver channel and client connection
AMQRRMFA	The repository process (for clusters)
AMQPCSEA	Command server
RUNMQTRM	Invoke a trigger monitor for a server
RUNMQDLQ	Invoke dead-letter queue handler
RUNMQCHI	Channel initiator process
RUNMQLSR	Channel listener process
AMQXSSVN	Shared memory servers

3. Stop IBM MQ service from **Administration tools > Services**

Figure 9-32. Stopping a queue manager manually on Windows

On Windows, you can use the Task Manager to find the running processes and then shut down the processes in the order listed on the figure.

Stopping a queue manager manually on UNIX and Linux

- Find process IDs of IBM MQ process by using the `ps` command and `grep` with the queue manager name

Example: `ps -ef | grep QMGR1`

- Use the `kill` command and process ID to stop any running processes in this order:

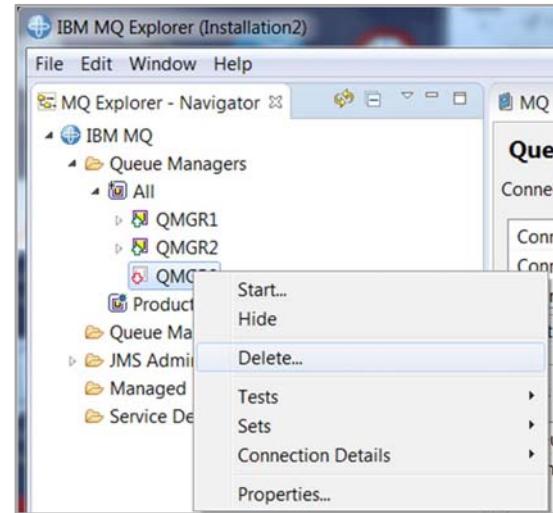
amqzmuc0	Critical process manager
amqzxma0	Execution controller
amqzfuma	OAM process
amqzlaa0	LQM agents
amqzlsa0	LQM agents
amqzmuf0	Utility Manager
amqzmur0	Restartable process manager
amqzmgr0	Process controller
amqfqpub	Publish Subscribe process
amqfcxba	Broker worker process
amqrmpa	Process pooling process
amqcrsta	Non-threaded responder job process
amqcrs6b	LU62 receiver channel and client connection
amqrrmfa	Repository process (for clusters)
amqpcsea	Command server
runmqtrm	Invoke a trigger monitor for a server
runmqdlq	Invoke dead-letter queue handler
runmqchi	Channel initiator process
runmqlsr	Channel listener process

Figure 9-33. Stopping a queue manager manually on UNIX and Linux

On UNIX or Linux, use the `ps -ef` command with `grep` to get a list all the MQ processes. Then, shut down the processes in the order listed on the figure with the UNIX `kill` command.

Removing a queue manager manually

- Use `dltmqm QMgrName` control command or IBM MQ Explorer
- If problems occur when deleting a queue manager:
 - Delete queue manager directory and its contents
 - Delete associated log directory and its contents
 - Remove queue manager's stanza from IBM MQ configuration file
 - Remove or change `DefaultQueueManager` stanza if required
- IBM MQ for Windows requires manual changes in Windows Registry



Diagnosing problems

© Copyright IBM Corporation 2017

Figure 9-34. Removing a queue manager manually

After you stop the queue manager, you can delete it by using the `dltmqm` control command or by using the MQ Explorer.

Deleting a queue manager is a drastic step because you also delete all resources that are associated with the queue manager, including all queues and their messages and all object definitions.

If you use the `dltmqm` command, a confirmation prompt is not displayed; when you press the Enter key, all the associated resources are lost.

If you cannot delete the queue manager by using the control command or MQ Explorer, you can delete the queue manager files manually, as described in the figure.

Channel problem determination

- Verify that queue manager is running
- Use TCP/IP **ping** command to verify connection from sender to receiver
- Verify that listener is running on receiving end of channel
- Verify that channels are paired and that they must have the same name
- Check error logs on the sending and receiving side of channel
- Use MQSC commands on sending side of problem channel to correct several channel-related problems:
 - **RESET CHANNEL** resets the message sequence number if necessary
 - **RESOLVE CHANNEL** requests a channel to commit or back out in-doubt messages

```
STOP CHANNEL (ChannelName)
RESET CHANNEL (ChannelName) SEQNUM(1)
RESOLVE CHANNEL (ChannelName) ACTION(BACKOUT | COMMIT)
START CHANNEL (ChannelName)
```

Figure 9-35. Channel problem determination

In some cases, you might suspect that a problem exists with the MQ channels. The figure lists some tips for locating problems with channels.

Remote administration failures

- AMQ4043 *Queue manager not available for connection* is generated for one of the following reasons:
 - Listener is not running on remote queue manager
 - Code page conversion failure
 - Security check failure
- Using IBM MQ Explorer or MQSC commands, check the following components on the remote queue manager:
 - IBM MQ command server is running
 - SYSTEM.ADMIN.SVRCONN channel is defined

[Diagnosing problems](#)

© Copyright IBM Corporation 2017

Figure 9-36. Remote administration failures

This figure lists the common causes of remote administration failures.

Ensure that the following requirements are satisfied before you try to use the MQ Explorer for remote administration. Verify the configuration.

- Verify that the MQ server and client are installed on the local and the remote computer.
- Verify that a command server is running for every queue manager.
- Verify that a TCP/IP listener exists for every queue manager. The listener can be the MQ listener (`runmq1sr`) or the InetD daemon (on UNIX), depending on your operating system environment.
- Verify that the server-connection channel that is called SYSTEM.ADMIN.SVRCONN exists on every remote queue manager. This channel is mandatory for every administered remote queue manager.
- Verify that the user ID of the initiator is a member of the “mqm” group on the local and remote computer. The model queue SYSTEM.MQEXPLORER.REPLY.MODEL must exist on every queue manager.

Channel triggering problems

- Channel initiator process `runmqchi` automates channel start and is started by default as part of queue manager
 - Verify that channel initiator is running
 - If channel initiator is not running, use the `runmqchi` control command to run a channel initiator process

Example: `runmqchi -q INIT.Q -m QMC01`
- Verify that channel initiator is monitoring initiation queue, not the transmission queue
- Check error log for channel error messages
- Try to start channel manually
 - If channel fails to start, or does not successfully move message from transmission queue to remote queue manager, then problem is with channel

[Diagnosing problems](#)

© Copyright IBM Corporation 2017

Figure 9-37. Channel triggering problems

You can make various checks if you encounter problems with triggering:

- Verify that the channel initiator is running.
- Use the `runmqchi` command to run the channel initiator process.
- Make sure that the channel initiator is monitoring the initiation queue, not the transmission queue.
- Check the error log for channel error messages.
- Try to start the channel manually.

When you debug a program with triggering a channel, set a short disconnect interval on the associated channel. The disconnect interval setting stops the channel quickly, with triggering enabled, and makes debugging easier.

Unit summary

- Determine the possible causes and locations of a missing message
- Analyze the error logs that IBM MQ generates
- Locate First Failure Support Technology (FFST) files on a system
- Use an IBM MQ trace to collect detailed information about IBM MQ operation
- Describe some of the more common problem types and how to approach initial problem determination
- Stop and remove a queue manager manually

Review questions

1. Why might a message be delayed on a transmission queue?
 - A. Queue manager is at MAXCHL
 - B. Transmission queue is full
 - C. Transmission queue is GET(DISABLED)
 - D. Channel initiator is not running
2. True or False: Persistent messages do not expire.
3. True or False: An uncommitted message can be browsed but not destructively removed.



Diagnosing problems

© Copyright IBM Corporation 2017

Figure 9-39. Review questions

Write your answers down here:

- 1.
- 2.
- 3.

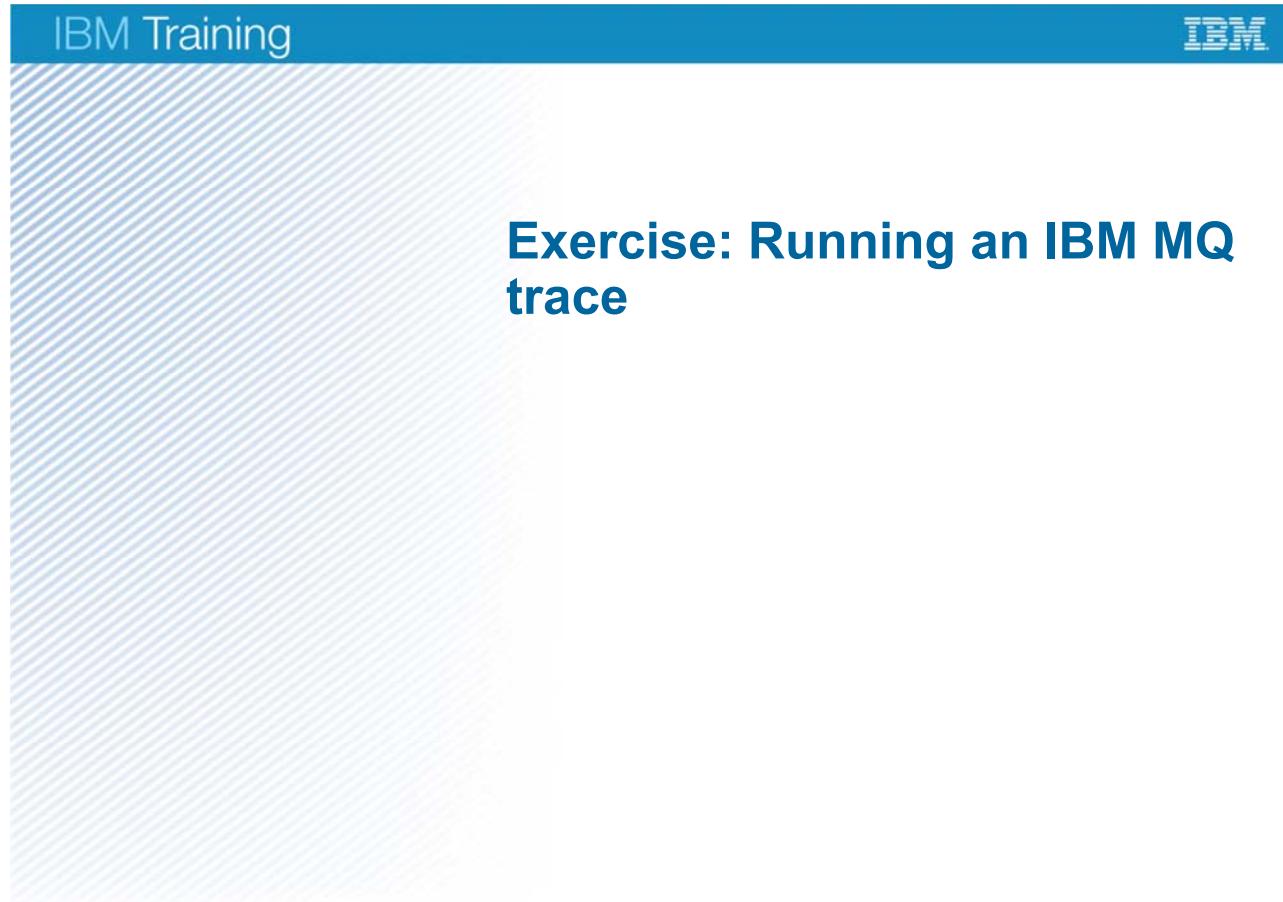
Review answers



1. Why might a message be delayed on a transmission queue?
 - A. [Queue manager is at MAXCHL](#)
 - B. Transmission queue is full
 - C. [Transmission queue is GET\(DISABLED\)](#)
 - D. [Channel initiator is not running](#)

The answer is A, C, and D. A full transmission queue returns an MQRC=2053 to the application that does the MQPUT or MQPUB.

2. True or [False](#): Persistent messages do not expire.
The answer is [False](#). An application can assign an expiration time for persistent messages.
3. True or [False](#): An uncommitted message can be browsed but not destructively removed.
The answer is [False](#). Uncommitted messages are not available to any application outside the current unit of work.



Diagnosing problems

© Copyright IBM Corporation 2017

Figure 9-41. Exercise: Running an IBM MQ trace

In this exercise, you start a trace on the IBM MQ `amqspput` and `amqsgget` sample programs. You then use the sample programs to send and receive messages and examine the trace output

Exercise objectives

- Start and stop an IBM MQ trace
- Analyze the output from the IBM MQ trace



Diagnosing problems

© Copyright IBM Corporation 2017

Figure 9-42. Exercise objectives

See the *Course Exercises Guide* for detailed instructions.

Unit 10. Implementing basic security in IBM MQ

Estimated time

01:00

Overview

In this unit, you learn how IBM MQ protects its objects by using access control lists (ACLs). You learn how the IBM MQ Object Authority Manager (OAM) uses these ACLs when a user attempts to access these objects. The unit also describes how to manage IBM MQ object authorizations and introduces Secure Sockets Layer (SSL) and Transport Layer Security (TLS) support.

How you will check your progress

- Review questions
- Hands-on exercise

References

IBM Knowledge Center for IBM MQ V9

Unit objectives

- Describe the role of the object authority manager (OAM) to provide security to IBM MQ resources
- Protect IBM MQ resources by using the OAM
- Use some of the OAM control commands
- Describe the Secure Sockets Layer (SSL) and Transport Layer Security (TLS) support that IBM MQ provides
- Implement basic channel authentication

Implementing basic security in IBM MQ

© Copyright IBM Corporation 2017

Figure 10-1. Unit objectives

The first topic provides a review on security. MQ security concepts were introduced in course WM103/ZM103, *Technical Introduction to IBM MQ V9*, which is a prerequisite for this course.

Security mechanisms

- **Identification**

Uniquely identify users of a system or application that is running in the system

- **Authentication**

Prove that a user or application is genuinely that person or what that application claims to be

- **Access control**

Protect resources in a system by limiting access to only authorized users and their applications

- **Confidentiality**

Encrypt messages to protect data

- **Auditing**

Track users and applications that access the system

Figure 10-2. Security mechanisms

A comprehensive security implementation includes the following requirements:

- The ability to uniquely identify users of a system or application
- The ability to prove that a user or application is authentic
- The ability to protect resources by limiting access to authorized users and applications
- The ability to protect confidential data
- The ability to track users and applications that access the system and the data

This unit shows how MQ supports the security implementations.

IBM MQ security implementations

- OAM installable service
- SSL for channel security
- Channel authentication rules for channel access control
- Connection authorization for queue manager access control
- Connection refusal events

Implementing basic security in IBM MQ

© Copyright IBM Corporation 2017

Figure 10-3. IBM MQ security implementations

This figure lists the MQ security implementations for identification, authentication, access control, confidentiality, and auditing.

The MQ Object Authority Manager (OAM) service provides **authorization** for MQI calls, commands, and access to objects to protect the local MQ resources.

The SSL protocol provides industry-standard channel security, with protection against eavesdropping, tampering, and impersonation to control access to the network. An MQ channel can be configured to receive and authenticate an SSL certificate from an SSL client.

By using MQ channel authentication, you can provide more precise control over the access that is granted to connecting systems at a channel level.

MQ connection authorization uses the supplied user ID and password to check whether a user has authority to access resources.

MQ security provides descriptive connection refusal events, which are written to a channel event queue.

Planning for security

1. Identify users that need authority to administer IBM MQ
2. Identify applications that need authority to work with IBM MQ objects
3. User ID that is associated with MCA's authority to access various IBM MQ resources

Implementing basic security in IBM MQ

© Copyright IBM Corporation 2017

Figure 10-4. Planning for security

You can use MQ for a wide variety of applications on a range of operating systems and hardware. The security requirements are likely to be different for each application. The first step to any security implementation is to identify the security requirements.

You must consider certain aspects of security when you implement MQ. If you ignore security aspects and do nothing on some operating systems, such as IBM i, UNIX, Linux, and Windows, you cannot use MQ.

First, identify users that need the authority to administer MQ. These users are the users that need to be able to enter commands and use MQ Explorer to access queue managers, queues, channels, and processes.

Second, identify applications that need to access to MQ queue managers, queues, processes, namelists, and topics by using MQI calls.

Third, identify the user IDs that are associated with the applications that need to administer channels and channel initiators, and open transmission queues.

For more information about security planning, see the IBM MQ product documentation.

IBM MQ access control overview

- Granular access control facilities
 - Provided by IBM MQ installable services
 - Which user? Which resource? What types of access?
 - Channel access control that is based on IP address, queue manager, SSL distinguished name, and asserted identity
- IBM MQ access control at user and group level
- Alternative user IDs can be specified when suitably authorized
- User needs access to the first named resource and not the alias queue or remote queue resolved resource

Implementing basic security in IBM MQ

© Copyright IBM Corporation 2017

Figure 10-5. IBM MQ access control overview

MQ access control is the primary security component. Access control allows MQ to control which users and applications are granted specific levels access to specific MQ resources. Resources that might be controlled in this way are the queue manager, queues, and processes.

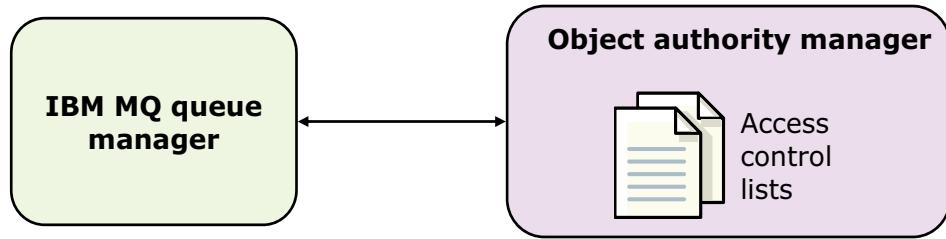
All queue managers on distributed operating systems provide access control facilities to control which users have access to which MQ resources. These queue managers use the OAM component of MQ to provide access control for MQ resources. By using the OAM, the MQ administrator can control access at the user level and at the group level.

When an application attempts to connect to MQ, MQ captures the user ID that is associated with the application from the MQCONN call. This user ID is used for the access control checks.

It is possible for authorized users to use an alternative user ID instead of the logged on user ID. The name that is specified in the MQ API command is used when MQ checks to see whether a user is authorized to access a particular resource.

For an alias or remote queue definition, the application user ID needs access to the queue that is specified in the MQ API command and not the resolved name. For model queues, MQ might generate the name of the dynamic queue in some instances. In this case, the user ID creating a dynamic queue is automatically given full access rights to the queue.

OAM installable service



- IBM MQ authorization service component
- Common access control list (ACL) manager for distributed queue managers
- Authorizations can be granted or revoked at the principal or group level
- IBM MQ Explorer, control commands, and MQSC commands can be used to configure and manage ACLs

[Implementing basic security in IBM MQ](#)

© Copyright IBM Corporation 2017

Figure 10-6. OAM installable service

The MQ OAM component is an MQ installable service.

The MQ OAM provides a full set of access control facilities for MQ including access control checking and commands to set, change, and inquire on MQ access control information.

Access to MQ entities is controlled through MQ user groups and the OAM.

The authorization service maintains an ACL for each MQ object to which it is controlling access. An ACL contains a list of all the group IDs and user IDs that can access the object.

Administrators can use MQ Explorer and commands to configure and manage ACL objects.

Principals and groups

- On UNIX and Linux
 - *Principal* is a user ID or an ID that is associated with an application program that is running on behalf of a user
 - *Group* is a system-defined collection of principals
 - ACLs are based on groups by default
 - Queue manager can be configured to support principals at creation with **-oa user** option or by editing **qm.ini**
- On Windows
 - *Principal* is a Windows user ID, or an ID that is associated with an application program that is running on behalf of a user
 - *Group* is a Windows group
 - ACLs are based on principals (user IDs) and groups
- Changes to a principal's group membership are not recognized until the queue manager is restarted or administrator runs **REFRESH SECURITY MQSC** command

[Implementing basic security in IBM MQ](#)

© Copyright IBM Corporation 2017

Figure 10-7. Principals and groups

Users can belong to groups. You can grant access to a particular resource to groups rather than to individuals to reduce the amount of administration required. For example, you might define a group that consists of users who want to run a particular application. Other users can be given access to all the resources they require by adding their user ID to the appropriate group.

On UNIX and Linux, all ACLs are based on groups by default. You can change the queue manager to support principals, similar to Windows.

On Windows, ACLs are based on user IDs and groups.

Any changes that you make to a principal's group membership are not recognized until the queue manager is restarted, or you enter the **REFRESH SECURITY MQSC** command.

Access control with the OAM

- Access control lists are specific to IBM MQ
 - Not integrated with system-level security
 - Changes to user's operating system authority are not recognized until queue manager restart or security refresh
 - One ACL for each queue manager, not shared between queue managers
- Use **`setmqaut`** control command, the **SET AUTHREC** MQSC command, and IBM MQ Explorer
 - Give users, and groups of users, access to IBM MQ objects
- Access control for IBM MQ objects
 - Queue manager
 - Queues
 - Processes
 - Namelists
 - Channels
 - Authentication information objects
 - Listeners

Implementing basic security in IBM MQ

© Copyright IBM Corporation 2017

Figure 10-8. Access control with the OAM

Each queue manager contains a set of ACLs for each queue manager. ACLs cannot be (automatically) shared with multiple queue managers. The ACLs are not integrated with system-level security.

The OAM provides access control facilities only for MQ objects: the queue manager, all queues, processes, namelists, channels, authentication information objects, listeners, and services.

You can use the **`setmqaut`** control command, the **SET AUTHREC** MQSC command, or IBM MQ Explorer to give users, and groups of users, access to MQ objects.



Note

On an IBM MQ Appliance, you can use only the **SET AUTHREC** command.

OAM access control lists

- One authorization file for each object plus global permissions files
- Can reference LDAP repository by specifying principal and group names in LDAP form
- Windows OAM bypasses authorization files for certain classes of principal:
 - SYSTEM
 - Local Administrators group
 - Local mqm group
- Uses IBM MQ object authorizations
 - Context, such as passall, passid
 - MQI, such as browse, put, get, set
 - Administration, such as dsp, chg, dlt
 - Generic, such as all, alladm, allmqi, none

Figure 10-9. OAM access control lists

Each OAM authorization file contains a set of access control stanzas. One stanza exists per principal for which access is controlled, where a principal is either a user ID or a group. MQ can be configured to connect to an LDAP repository and reference principal and group names in LDAP form.

The principals (user IDs, groups, or both) that can access an object are listed in this file. This file includes a bit string (in hex) that represents the access rights that are associated with that entity.

Certain principals or groups are granted automatic access to MQ resources:

- Members of the “mqm” group or the “mqm” user
- On Windows:
 - Administrator user and local group
 - SYSTEM user ID
 - The user or principal group that creates a resource

Authorizations can be assigned to the following categories:

- Authorizations for MQI context such as passing all the context fields (`passall`) or the identity context (`passid`) from the request message to a message that the application is putting on the queue
- Authorizations for sending MQI calls to browse, put, or get messages, or to set queue attributes
- Authorizations for entering commands for administration tasks such as displaying (`dsp`), changing (`chg`), and deleting (`del`) objects
- Generic authorizations such as all (`all`), all administration (`alladm`), all MQI (`allmqi`), and no authorizations (`none`)

Set or reset authorization

- Use **setmqaut** command to set or reset authorization by object type
 - Prefix authorization with a plus sign (+) to add
 - Prefix authorization with a minus sign (-) to revoke
- Command is cumulative
 - Option 1: Set authorization explicitly on each **setmqaut** command to avoid retaining unwanted pre-existing authorities
 - Option 2: Specify **-all** to remove all authorization and then grant required authorizations

Format: **setmqaut -m QMgr -t Objtype -n Profile [-p Principal | -g Group] permissions**

Example: **setmqaut -m JUPITER -t queue -n MOON.* -g VOYAGER +browse -put**
setmqaut -m QM -t queue -n Q1
-p cn=useradm,ou=users,o=ibm,c=UK +put

Figure 10-10. Set or reset authorization

Three control commands provide control and verification of the security environment for MQ: **setmqaut**, **dspmqaut**, and **dmpmqaut**. These programs require a connection to the authorization service; they can be used when the target queue manager is active and the OAM is enabled. All of the responses to these commands are displayed on the screen.

The **setmqaut** command sets the access to a particular resource by a principal or group. This command can be used to add or revoke privileges.

The **setmqaut** command can use generic profiles. In the first example, the **setmqaut** command allows members of the group VOYAGER (**-g VOYAGER**) to browse (**+browse**) but not put (**-put**) messages on the queues (**-queues**) that start with the characters “MOON” (**-n MOON.***) that the JUPITER queue manager (**-m JUPITER**) owns.

The second example shows how to use the LDAP form to reference a principal in an LDAP repository (**-p**). This command sets put permissions (**+put**) on the queue (**-queue**) that is named Q1 (**-n Q1**) on the queue manager that is named QM (**-m Q1**).

If you use the **setmqaut** command to set authorizations for an individual user ID, the authorizations are held at the level of the individual user ID. However, for MQ on UNIX, authorizations are held at the level of the primary group of the user ID, and so all members of that group acquire the same authorizations.

The control command `setmqaut` is used to grant and revoke authorizations to an MQ object. Using the command, you can grant or revoke authorizations for an individual user ID or for a group.

Display authorization

- Use **dspmqaut** command to verify that object authorization is correct
 - By object type
 - For a principal or group

Format: **dspmqaut -m QMgr -t ObjType -n ObjName [-p Principal | -g Group] [-s Service]**

Example: **dspmqaut -m SATURN -t q -n APPL.Q1 -p mquser**
 Entity mquser has the following authorizations for object APPL.Q1:
 get
 browse
 ...

Figure 10-11. Display authorization

The **dspmqaut** command displays current authorizations for MQ objects that include queues, queue managers, and processes.

This command does support generic profiles. If a user ID is a member of one or more groups, the display authorization command displays the combined authorizations of all the groups.

Only one group or principal can be specified.

The example command displays the authorizations for the queue (-q) that is named APPL.Q1 (-n APPL.Q1) on queue manager SATURN (-m SATURN) for the user that is named “mquser” (-p mquser).

Create authorization report

- Use **dmpmqaut** command to generate a report of current authorizations
 - By object type
 - For a principal or group

Format: **dmpmqaut -m Qmgr -t Objtype [-n Profile | -l]
[-p Principal | -g Group]**

Example: **dmpmqaut -m qm1 -t q -n a.b.c -p user1**
 profile: a.b.*
 object type: queue
 entity: user1
 type: principal
 authority: get, browse, put, inq

Figure 10-12. Create authorization report

Use the MQ **dmpmqaut** control command to create a report of the current authorizations that are associated with a specified profile (-n).

The -l parameter creates a report with a terse list of all profiles names and types.

Group names must exist, and you can specify one group name on the **dmpmqaut** command. MQ for Windows allows the use of local groups only.

The example in the figure creates a report that shows the object authority for the queues on queue manager **qm1** for the user **user1**.

Using MQSC to manage authorization

- Use MQSC command **SET AUTHREC** to set authority records that are associated with a profile name

Example:

```
SET AUTHREC OBJTYPE(QMGR)
PRINCIPAL('JohnDoe1@yourcompany.com') AUTHADD(connect)
```

- Use MQSC command **DISPLAY AUTHREC** to display authority records that are associated with a profile name
- Use MQSC command **DELETE AUTHREC** to delete authority records

You must use MQSC to manage authorization on the IBM MQ Appliance

Figure 10-13. Using MQSC to manage authorization

You can use the MQSC **SET AUTHREC**, **DISPLAY AUTHREC**, and **DELETE AUTHREC** commands to manage authorizations on a specific queue manager.

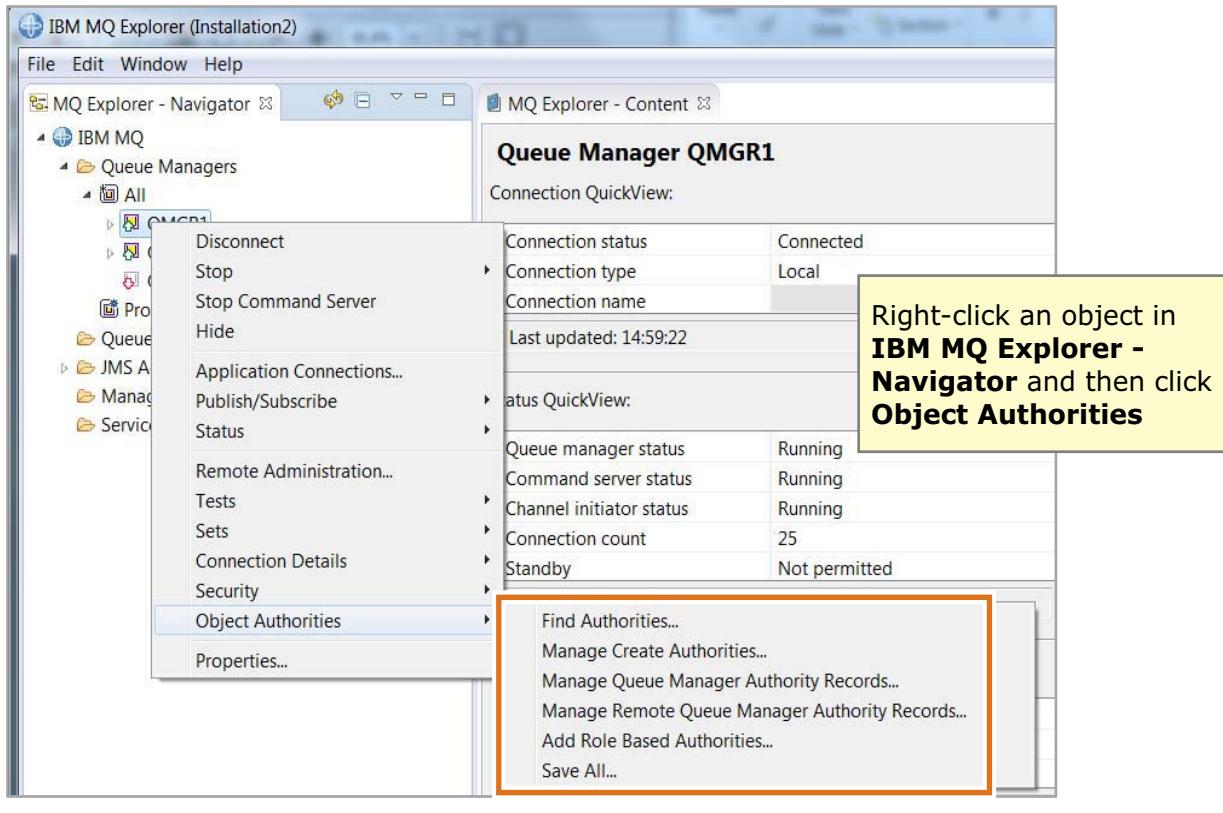
The list of authorizations to add and the list of authorizations to remove must not overlap. For example, you cannot add “display” authority and “remove display” authority with the same command. This rule applies even if the authorities are expressed by using different options.

For example, the following command fails because DSP authority overlaps with ALLADM authority:

```
SET AUTHREC PROFILE(*) OBJTYPE(QUEUE) PRINCIPAL(PRINC01) AUTHADD(DSP)
AUTHRMV(ALLADM)
```



Using IBM MQ Explorer to manage authorization



Implementing basic security in IBM MQ

© Copyright IBM Corporation 2017

Figure 10-14. Using IBM MQ Explorer to manage authorization

As an option, you can manage object authorities with MQ Explorer.

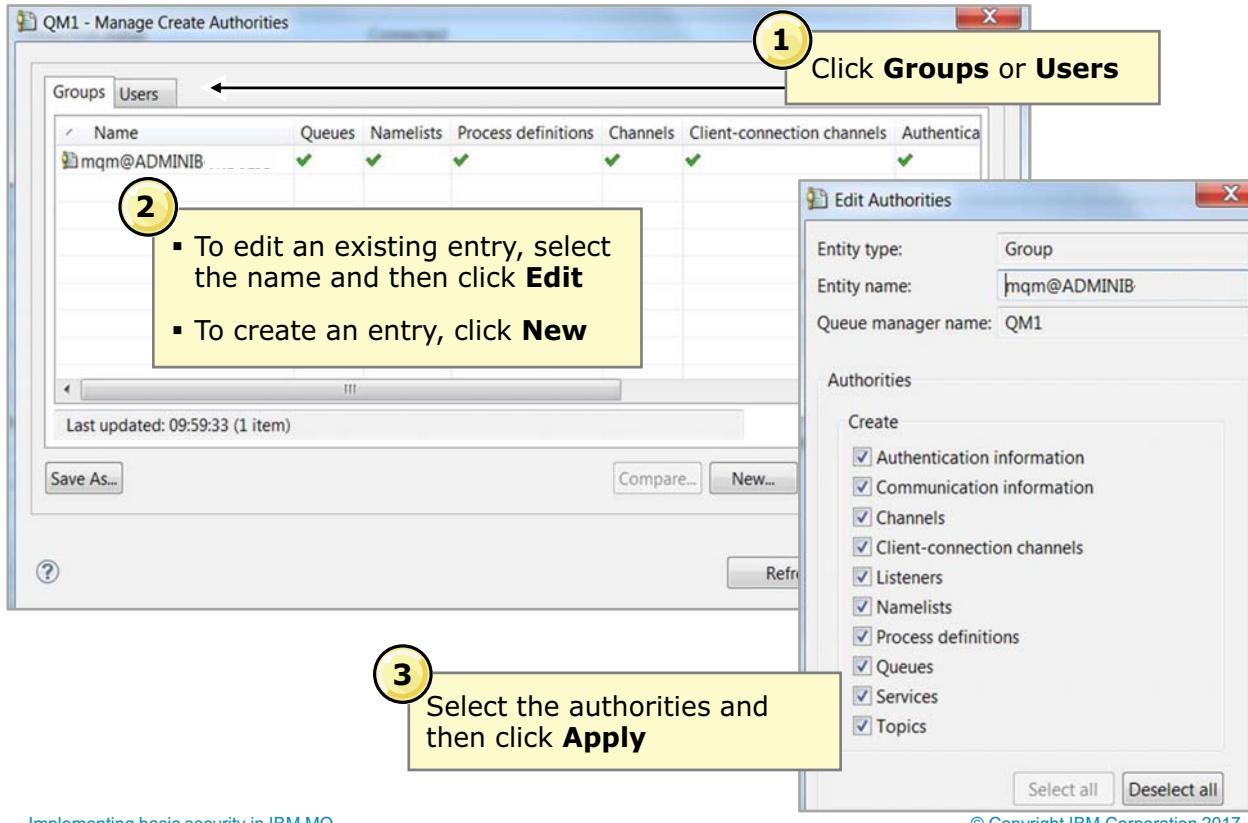
In MQ Explorer, you can:

- Find authorities
- Manage “create” authorities
- Manage queue manager authority records
- Manage remote queue manager authority records
- Add role-based authorities

For example, to manage the authority records for a queue manager, right-click the queue manager in the **MQ Explorer - Navigator** view and then click **Object Authorities**.

IBM Training

Queue manager authorization in IBM MQ Explorer



Implementing basic security in IBM MQ

© Copyright IBM Corporation 2017

Figure 10-15. Queue manager authorization in IBM MQ Explorer

To modify an existing authority record, complete the following steps:

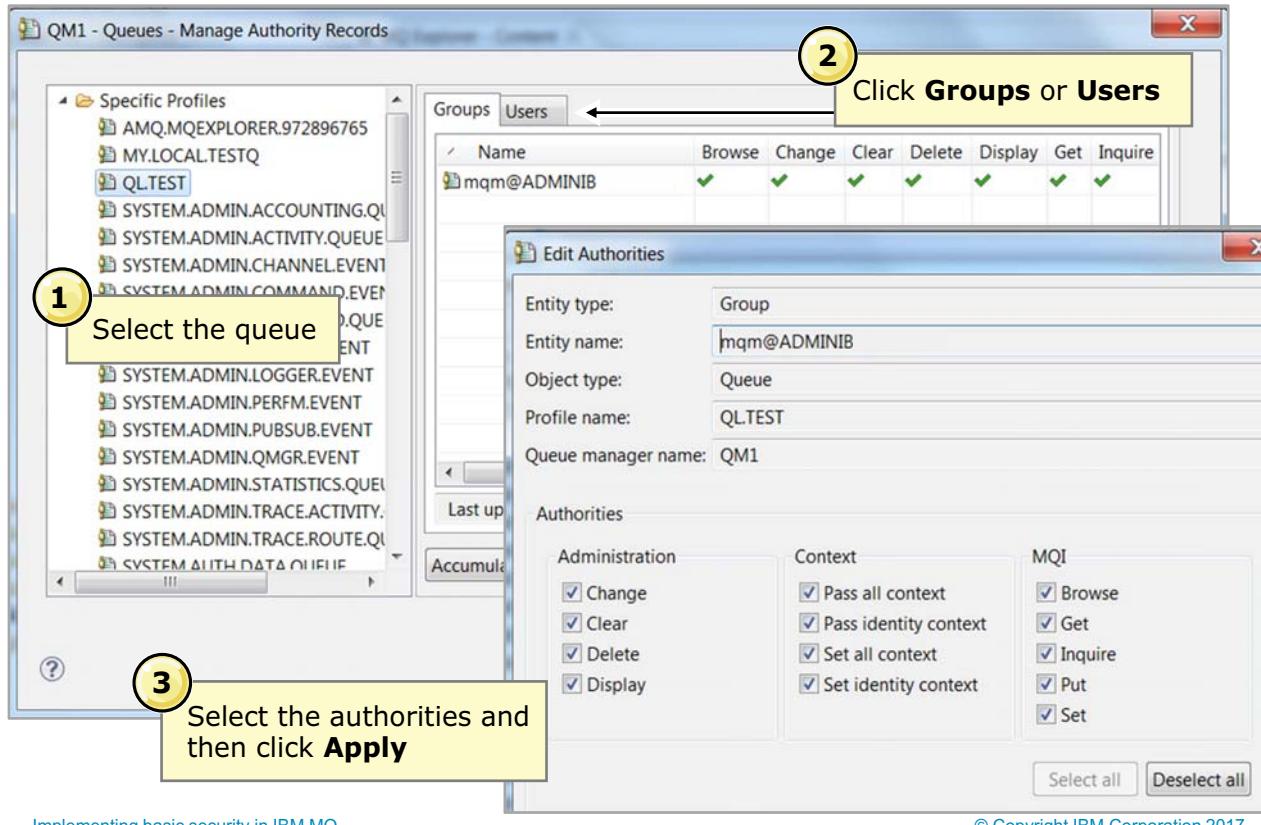
1. Click the **Groups** tab to modify a group record, or click the **Users** tab to modify the record for a specific user.
2. Select the group or user and then click **Edit**.
3. Modify the authorities and then click **OK**.

To create a new authority record:

1. Click the **Groups** tab to add a record for a group, or click the **Users** tab to add a record for a specific user.
2. Click **New**.
3. Enter an entity name, select the authorities, and then click **OK**.

IBM Training

Queue authorization in IBM MQ Explorer



Implementing basic security in IBM MQ

© Copyright IBM Corporation 2017

Figure 10-16. Queue authorization in IBM MQ Explorer

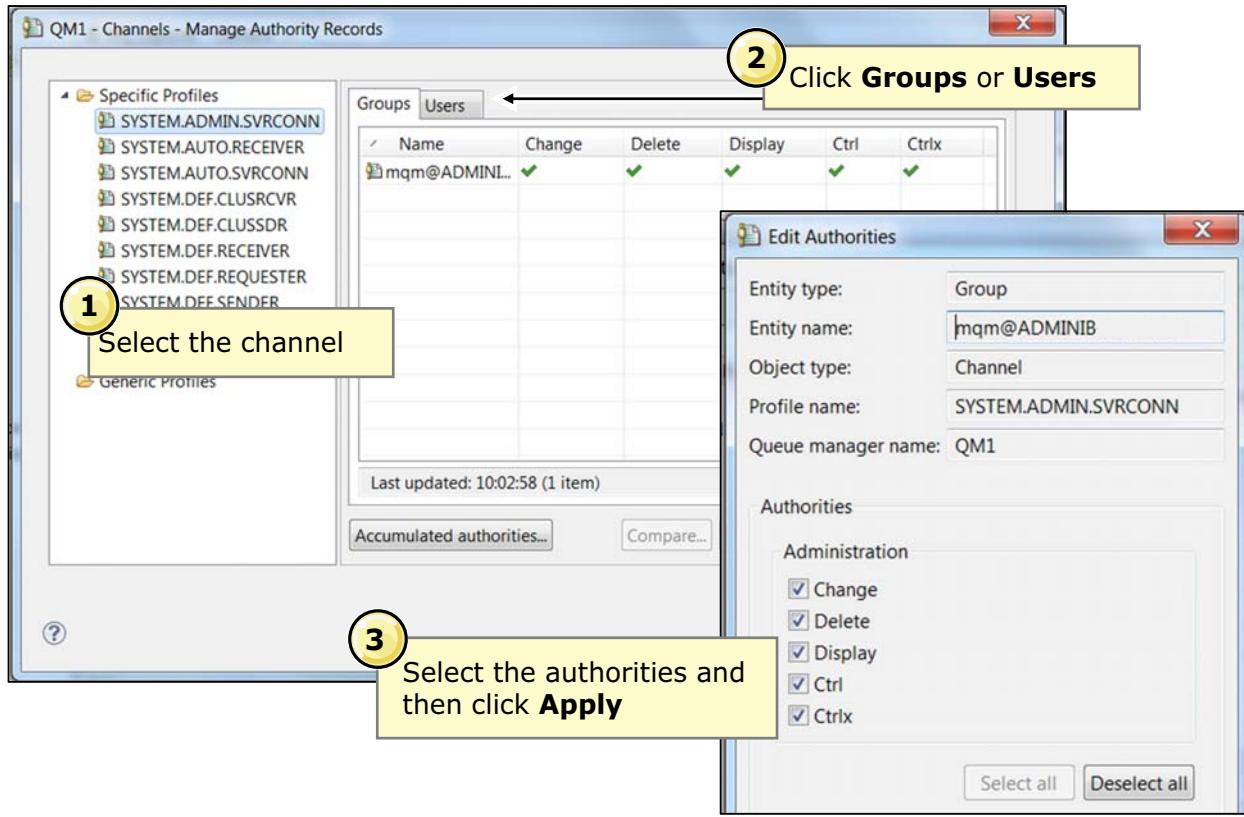
You can modify or add queue authority records in MQ Explorer, by right-clicking the queue in the **Queue** content view and then clicking **Object Authorities > Manage Authority Records**.

On the **Manage Authority Records** view, complete the following steps:

1. Select the queue.
2. Click the **Groups** or **Users** tab. Select the record and then click **Edit** to modify an existing record, or click **Add** to add a record.
3. Select the authorities and then click **Apply**.



Channel authorization in IBM MQ Explorer



Implementing basic security in IBM MQ

© Copyright IBM Corporation 2017

Figure 10-17. Channel authorization in IBM MQ Explorer

You can also use MQ Explorer to add or modify channel authority records by using the same technique that is used to add or modify queue authority records.

Right-click the channel in the **Channels** content view and then click **Object Authorities > Manage Authority Records**.

On the **Manage Authority Records** view, complete the following steps:

1. Select the channel.
2. Click the **Groups or Users** tab. Select the record and then click **Edit** to modify an existing record, or click **Add** to add a record.
3. Select the authorities and then click **Apply**.

Access control for IBM MQ control commands

- Use of most IBM MQ control commands is restricted
 - Example: `crtmqm`, `strmqm`, `runmqsc`, `setmqaut`
- UNIX and Linux restricts users to `mqm` group
 - Configuration as a part of IBM MQ installation
 - Control that the operating system imposes, not OAM
- Windows allows:
 - `mqm` group
 - Administrators group
 - System user ID

Implementing basic security in IBM MQ

© Copyright IBM Corporation 2017

Figure 10-18. Access control for IBM MQ control commands

Access control can be configured for most MQ control programs. Control programs include commands for creating queue managers, starting queue managers, running MQSC, and setting authorization.

By default, UNIX and Linux restrict access to control programs to members of the “`mqm`” group.

By default, Windows restricts access to control programs to members of the “`mqm`” and “Administrators” groups, and the Windows System ID.

Security and distributed queuing

- Put authority option for receiving end of message channel
 - Default user identifier is used
 - Context user identifier is used
- Transmission queue
 - Messages that are destined for remote queue manager are put on transmission queue by local queue manager
 - Application does not normally need to put messages directly on transmission queue, or need authorization to do so
 - Only special system programs should put messages directly on a transmission queue and have the authorization to do so

Figure 10-19. Security and distributed queuing

You can also implement security between queue managers. On the receiving end of a message channel, you can specify the user ID to use for checking the authority of the receiving MCA to open a destination queue.

- Choose **Default user identifier** to use the receiving MCA's default user identifier. A security exit or setting the MCAUSER attribute in the channel definition at the receiving end of the message channel can change this user identifier.
- Choose **Context user identifier** to use the user identifier in the context of the message.

In a typical implementation, it is the queue manager that puts messages that are destined for a remote queue manager onto the transmission queue. For special cases and custom applications, you can allow special system programs only to put messages directly on a transmission queue.

Security authorization for remote queues

- Distributed operating systems have authorizations for remote and clustered queues
- For applications that explicitly open `queue@qmgr`, which is a common pattern when using reply to information

Example: `setmqaut -m QM1 -t rqname -n QM2 -p mquser +put`

Figure 10-20. Security authorization for remote queues

MQ supports security authorization for remote and clustered queues by using a remote queue manager object that the OAM recognizes. Authorities are applied to the remote queue manager object instead of the transmission queue that is used to send the message to the remote queue.

An example of a command to set security authorization for a non-local queue is provided in the figure. The example gives the user “mquser” put authority on the remote queue (`-t rqname`) that is named QM2 (`-n QM2`).

When an application is attempting to access an MQ object, the general rule is that its authority is checked on the first object in the resolution path. For example, if the object descriptor supplies the name of a remote queue and remote queue manager, authority checking occurs on the transmission queue with the same name as the remote queue manager. If the application attempts to open a local definition of a remote queue, authority checking occurs against that object. For an alias queue, authority checking occurs at the level of the alias queue, not at the level of the queue to which it resolves.

Limit the ability to define queues to privileged users. Otherwise, normal access control can be bypassed by creating an alias queue. The use of the `+crt` authorization on the `setmqaut` control command allows you to specify which users are allowed to create queues.

Example: `setmqaut -m QMC01 -t queue -g GROUPB +crt`

Authorization checking in the MQI

- MQI calls with security checking
 - MQCONN and MQCONNX
 - MQOPEN
 - MQPUT1 (implicit MQOPEN)
 - MQSUB
 - MQCLOSE (for dynamic queues)
- If not authorized, then reason code **MQRC_NOT_AUTHORIZED** is returned

[Implementing basic security in IBM MQ](#)

© Copyright IBM Corporation 2017

Figure 10-21. Authorization checking in the MQI

This figure lists the MQI calls with security checking.

Applications that send the MQCONN or MQCONNX call must be authorized to connect to the queue manager.

For MQOPEN and MQPUT1, the authority check is made on the name of the object that is opened, and not on the name or names that result after a name is resolved. For example, an application might be granted authority to open an alias queue without having authority to open the base queue to which the alias resolves. The rule is to check the first definition that is encountered during the process of resolving a name that is not a queue manager alias, unless the queue manager alias definition is opened directly.

When an MQSUB call is sent, the queue manager verifies that the user identifier under which the application is running has the appropriate level of authority to subscribe to the topic object.

The MQCLOSE call has options for deleting and purging permanent dynamic queues. If one of these options is set, a check determines whether the user is authorized to delete the queue. This check is not done if the application that created the queue is attempting to delete the queue.

If authorization fails for any of the MQI calls, an MQRC_NOT_AUTHORIZED event is written to the SYSTEM.ADMIN.QMGR.EVENT event queue. This event indicates that a command is entered by a user ID that is not authorized to access the object that is specified in the command.

Authority events (1 of 2)

- Queue manager event
- Stored on SYSTEM.ADMIN.QMGR.EVENT queue
- MQRC_NOT_AUTHORIZED event types
 1. On MQCONN or system connection call, the user is not authorized to connect to the queue manager
 2. On an MQOPEN or MQPUT1, the user is not authorized to open the object for the options specified
 3. When closing a queue with MQCLOSE, the user is not authorized to delete the object, which is a permanent dynamic queue
 4. Command was issued from a user ID that is not authorized to access the object specified in the command
 5. On an MQSUB, the user is not authorized to subscribe to the specified topic
 6. On an MQSUB, the user is not authorized to use the destination queue with the required level of access

Figure 10-22. Authority events (1 of 2)

Authority events report an authorization, such as an application that tries to open a queue for which it does not have the required authority. It might also be a command that is issued from a user ID that does not have the required authority.

Authority events (2 of 2)

- Enable by setting **AUTHOREV (ENABLED)** on queue manager or **Events > Authority events** queue manager property to **Enabled** in IBM MQ Explorer
- Use the **amqsevt** sample program to format and display queue manager events

Example:

```
amqsevt -m QM01 -q SYSTEM.ADMIN.QMGR.EVENT
Event Type          : Queue Mgr Event [44]
Reason              : Not Authorized [2035]
Event created       : 2016/10/26 09:52_04.54 GMT
Queue Mgr Name     : QM01
Reason Qualifier   : Conn Not Authorized
User Identifier     : oamlabuser
Appl Type          : Unix
Appl Name          : amqspput
```

Figure 10-23. Authority events (2 of 2)

You can enable authority events by modifying the queue manager properties with the **ALTER QMGR** command or by using IBM MQ Explorer.

Channel authentication control

- Enabled by default
- Rules are based on:
 - Connecting IP address
 - Connecting queue manager name
 - SSL distinguished name
 - Asserted identity (including *MQADMIN option)
 - Derived identity from distinguished name mapping
- Rules can be applied in WARNING mode to allow connection but generate errors

Implementing basic security in IBM MQ

© Copyright IBM Corporation 2017

Figure 10-24. Channel authentication control

In MQ, channel authorization (CHLAUTH) records define the rules that are applied when a queue manager or client attempts to connect through a channel. Channel authorization is enabled by default.

Channel authentication uses rules to control access to a channel. A wildcard can be used with the MQ administrator ID (*MQADMIN) in these rules to cover the use of any ID that would otherwise gain automatic administrative rights over a queue manager. Having a generic user ID, such as *MQADMIN, makes it easier to have the same rules on all operating systems, where the actual definition of who is an administrator might vary.

Rules can also be defined as “WARN” and generate authorization events without blocking the connection. This behavior might help when you change to a secure environment by not turning off connections immediately.

Channel access control is taught in detail in course WM113/ZM113, *IBM MQ V9 Advanced System Administration*.

Channel authentication commands

- Use MQSC command **SET CHLAUTH** to create or modify a channel authentication record
- Use MQSC command **DISPLAY CHLAUTH** to test rules that you define
- Use MQSC command **ALTER QMGR CHLAUTH(DISABLED)** to disable channel authentication

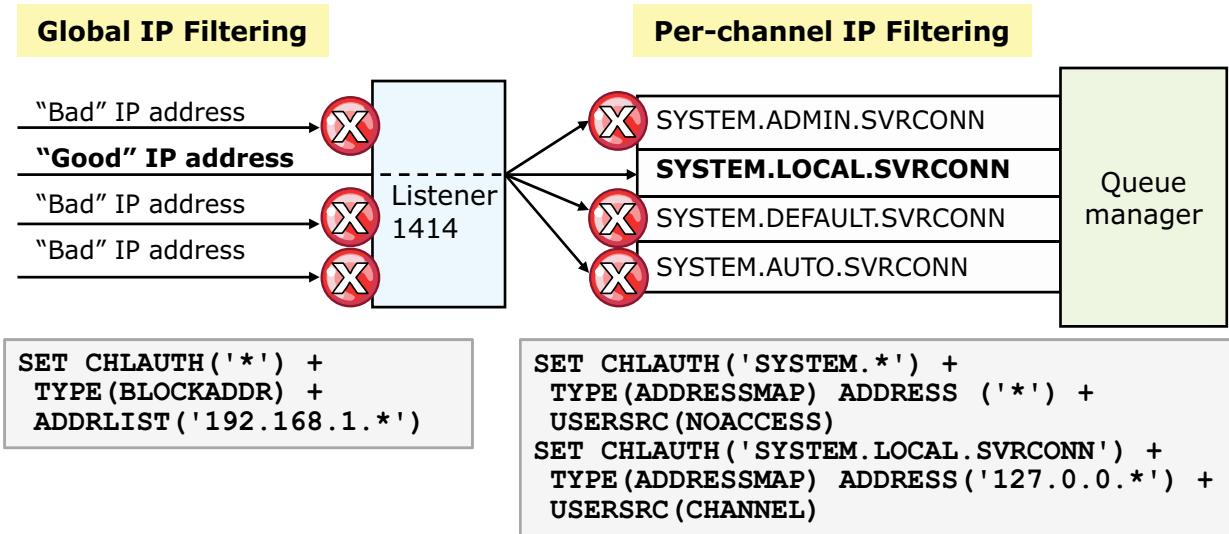
Figure 10-25. Channel authentication commands

Use the MQSC command **SET CHLAUTH** to configure channel authentication control.

Use the MQSC command **DISPLAY CHLAUTH** with the **MATCH(RUNCHECK)** option to verify a simulated connection. Rules can be tested from the console without making a real connection.

If necessary, such as in a development environment, you can disable channel authentication by using the MQSC command **ALTER QMGR CHLAUTH(DISABLED)**.

Channel authentication example



- Filter connection requests based on IP address of requester
- Per-channel rules match least-specific to most-specific
- Global blocking rules occur at listener before channel name is known and take precedence over per-channel rules

Implementing basic security in IBM MQ

© Copyright IBM Corporation 2017

Figure 10-26. Channel authentication example

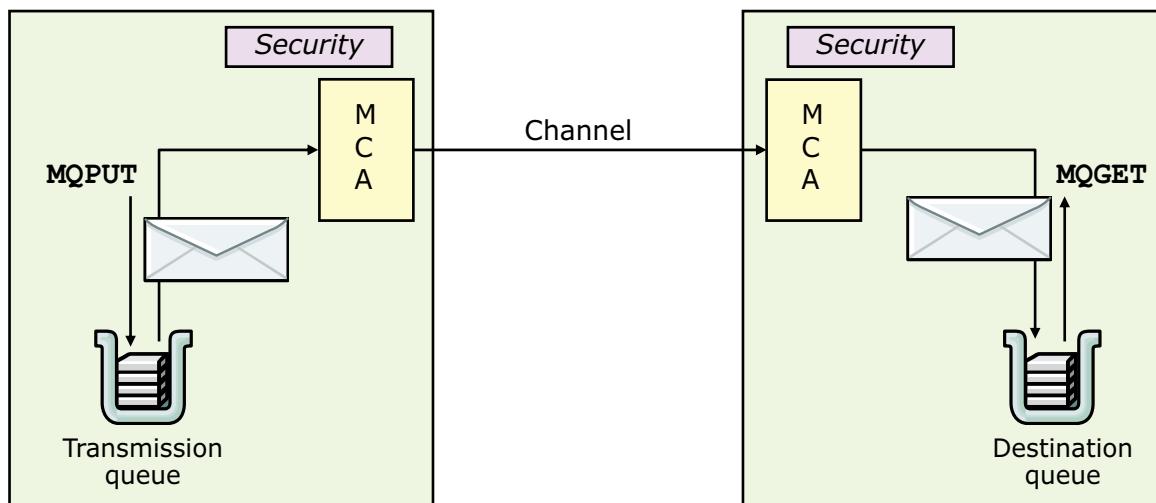
This figure provides an example of channel authentication.

The **SET CHLAUTH** command with the **TYPE(BLOCKADDR)** option provides global IP filtering. In the command, the address list (**ADDRLIST**) is the address from which you are refusing connections. The address can be a specific IP address or a pattern that includes the asterisk (*) as a wildcard or the hyphen (-) to indicate a range that matches the address. In this example in the figure, the **SET CHLAUTH** command blocks all IP addresses beginning with 192.168.1.

In the example on the right, the first **SET CHLAUTH** command restricts access to all SYSTEM channels from any IP address because the **USERSRC** parameter is set to **NOACCESS**. The **NOACCESS** option means that inbound connections that match this mapping do not have access to the queue manager and the channel ends immediately.

The second command gives the local host (127.0.0) access to the channel that MQ Explorer uses because the **USERSRC** parameter is set to **CHANNEL**. The **CHANNEL** option means that inbound connections that match this mapping use the flowed user ID or any user that is defined on the channel object in the **MCAUSER** field.

IBM MQ security exits



- Allows an MCA to authenticate its partner
- Usually work in pairs at each end of channel
- Called immediately after initial data negotiation completes on channel startup, but before any messages start to flow
- Formats of security message and security exit program are user-defined

[Implementing basic security in IBM MQ](#)

© Copyright IBM Corporation 2017

Figure 10-27. IBM MQ security exits

If the MQ provided security options do not provide the level of support that your application requires, you can use a *channel security exit* to use a custom application for security.

A channel security exit forms a secure connection between two security exit programs, where one program is for the sending MCA, and one is for the receiving MCA.

Security exits normally work in pairs, one at each end of a channel. They are called immediately after the initial data exchange negotiation completes on channel startup, but before any messages start to flow. One possible outcome of the conversation between the security exits is that the channel is closed and message flow is not allowed to proceed.

The name of the security exit is specified as a parameter on the channel definition at each end of a channel.

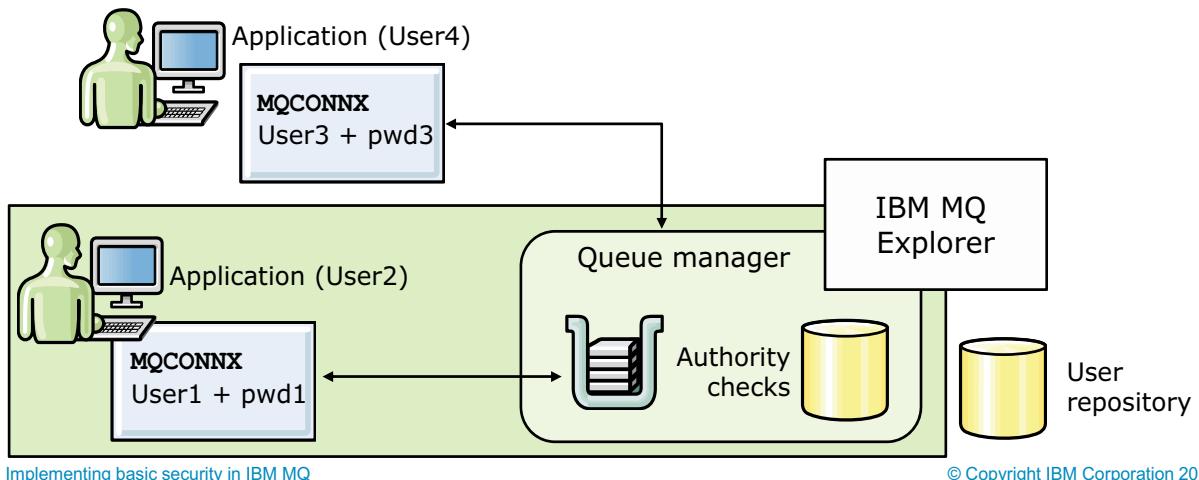


Note

Try to use MQ options such as SSL or connection authentication before you write a custom exit.

Connection authentication

- Client application can provide a user ID and password
- Queue manager can be configured to act on a supplied user ID and password
- LDAP repository can be used to determine whether a user ID and password combination is valid
- Application user ID, which is the usual operating system user ID presented to IBM MQ, might be different from the user ID that the application provides



Implementing basic security in IBM MQ

© Copyright IBM Corporation 2017

Figure 10-28. Connection authentication

Connection authentication in MQ can be achieved in various ways:

- An application can provide a user ID and password. The application can either be a client, or it can use local bindings.
- A queue manager can be configured to act on a supplied user ID and password.
- An external repository such as an LDAP repository can be used to determine whether a user ID and password combination is valid.

In the diagram, two applications are making connections with a queue manager, one application as a client and one using local bindings. Applications might use various APIs to connect to the queue manager, but all can provide a user ID and a password. The user ID that the application is running under, User1 and User3 in the diagram, might be different from the user ID that is provided by the application (User2 and User4).

Enabling connection authentication on a queue manager

- Define authentication information object with **DEFINE AUTHINFO** command
 - **IDPWOS** indicates that queue manager uses local operating system to authenticate user ID and password
 - **IDPWLDAP** indicates that queue manager uses an LDAP server to authenticate user ID and password
- Set **CONNAUTH** attribute on queue manager to name of an authentication information object
- Use **REFRESH SECURITY** command to refresh cached view of configuration for connection authentication

Figure 10-29. Enabling connection authentication on a queue manager

To configure a queue manager to use a supplied user ID and password to check whether a user has authority to access resources, enable connection authentication on the queue manager.

First, define an authentication information object. You can define the authentication object to use the local operating system to authenticate the user ID and password (**IDPWOS**). Optionally, you can use an LDAP server to authenticate the user ID and password (**IDPWLDAP**).

After you define the authentication object, set the **CONNAUTH** attribute on the queue manager to the name of an authentication information object.

You must refresh the configuration before the queue manager recognizes the changes.

Enabling connection authentication examples

Example 1: Define an authentication object that uses local file system to authenticate the user ID and password

```
DEFINE AUTHINFO(USE.OS) AUTHTYPE(IDPWOS)
ALTER QMGR CONNAUTH(USE.OS)
REFRESH SECURITY TYPE(CONNAUTH)
```

Example 2: Define an authentication object that uses an LDAP server to authenticate the user ID and password

```
DEFINE AUTHINFO(USE.LDAP) AUTHTYPE(IDPWLDAP) +
CONNNAME('ldap1(389),ldap2(389)') +
LDAPUSER('CN=QMGR1') LDAPPWD('passw0rd')
ALTER QMGR CONNAUTH(USE.LDAP)
REFRESH SECURITY TYPE(CONNAUTH)
```

Figure 10-30. Enabling connection authentication examples

This figure provides two connection authentication examples.

Example 1

- The **DEFINE AUTHINFO** command defines an authentication object that is named USE.OS. The **AUTHTYPE(IDPWOS)** attribute directs this authentication object to use the local operating system to authenticate the user ID and password.
- The **ALTER QMGR** command enables connection authorization on the queue manager by using the USE.OS authentication object.
- The **REFRESH SECURITY** command refreshes connection authorization security on the queue manager so that the queue manager recognizes the changes.

Example 2

- The **DEFINE AUTHINFO** command defines an authentication object that is named USE.LDAP that uses an LDAP server to authenticate the user ID and password.
- The **ALTER QMGR** command enables connection authorization on the queue manager by using the USE.LDAP authentication object.
- The **REFRESH SECURITY** command refreshes connection authorization security on the queue manager so that the queue manager recognizes the changes.

Secure Sockets Layer (SSL)

- Protocol that allows transmission of secure data over an insecure network
- Combines these techniques:
 - Symmetric and secret key encryption
 - Asymmetric and public key encryption
 - Digital signature
 - Digital certificates
- Protection
 - Client/server
 - Queue manager and queue manager channels
- Combats security problems
 - Eavesdropping: Encryption techniques
 - Tampering: Digital signature
 - Impersonation: Digital certificates

Implementing basic security in IBM MQ

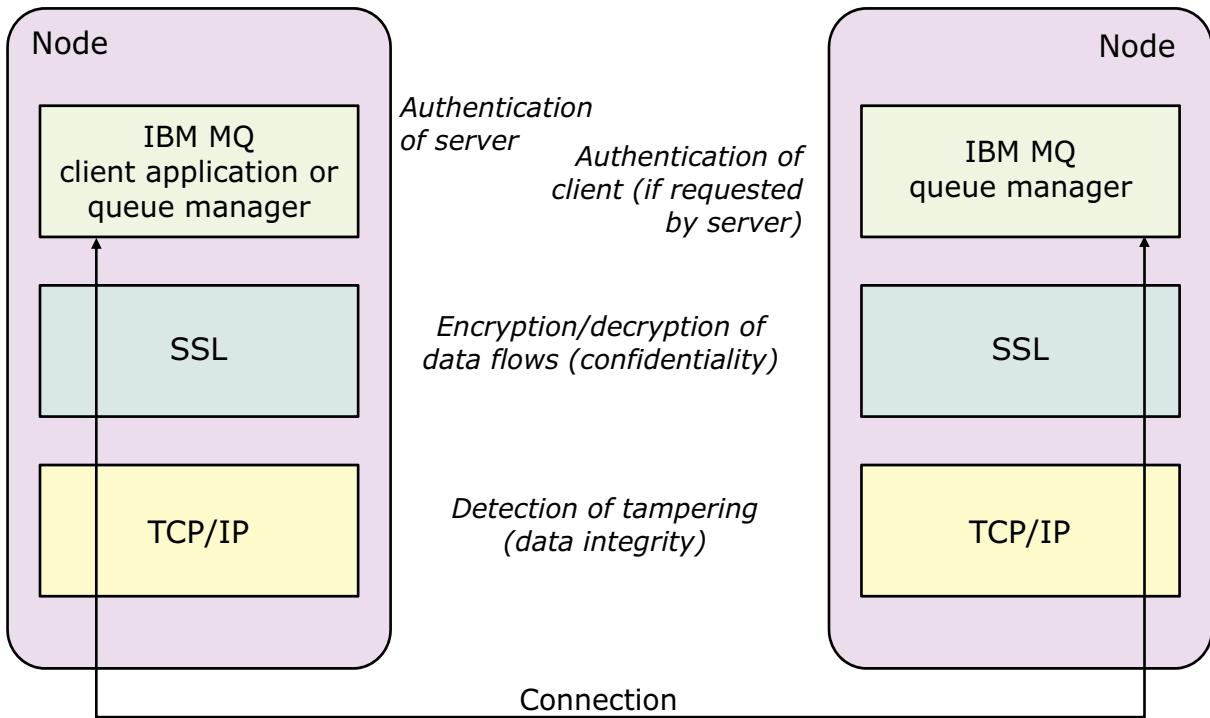
© Copyright IBM Corporation 2017

Figure 10-31. Secure Sockets Layer (SSL)

SSL is an industry-standard protocol for secure communications, involving encryption, authentication, and integrity of data. SSL is supported in both client/server and queue manager/queue manager channels (including clusters). MQ provides many flexible built-in capabilities such as the ability to identify the users based on their fully authenticated identity. This feature removes, for many people, the requirement to set up channel exits, where they were used for security purposes.

SSL uses a combination of public key and symmetric key encryption to ensure message privacy. Before messages are exchanged, an SSL server and SSL client do an electronic handshake during which they agree to use a session key and encryption algorithm. All messages between the client and the server are then encrypted. Encryption ensures that the messages remain private even if eavesdroppers intercept it.

IBM MQ SSL support



Implementing basic security in IBM MQ

© Copyright IBM Corporation 2017

Figure 10-32. IBM MQ SSL support

MQ provides the following support for SSL:

- Authentication of the server on the client
- Authentication of the client, if the server requests it
- Encryption/decryption of data flows
- Detection of tampering

Enabling SSL on the queue manager

- Use IBM MQ Explorer or the MQSC command **ALTER QMGR**

Attribute	Description
SSLCRLNL	Name of a namelist of authentication information objects that provide certificate revocation locations (CRLs) for enhanced TLS/SSL certificate verification
SSLCRYP	Name of parameter string that is required to configure cryptographic hardware present on system
SSLKEYR	Name of SSL key repository
SSLTASKS	Number of server subtasks to use for processing SSL calls
SSLEV	Enable or disable SSL event messages
SSLFIPS	Specify whether only FIPS-certified algorithm is used if cryptography is provided by IBM MQ, rather than cryptographic hardware

Implementing basic security in IBM MQ

© Copyright IBM Corporation 2017

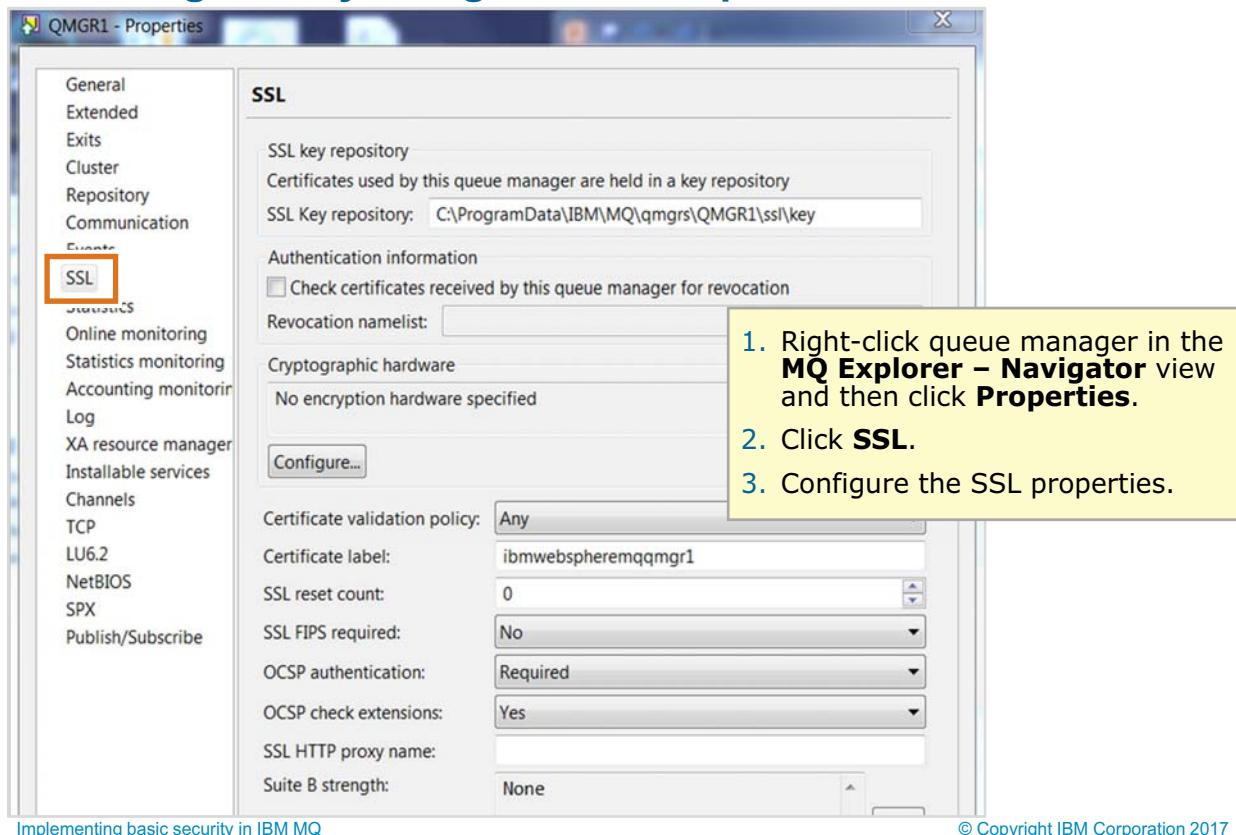
Figure 10-33. Enabling SSL on the queue manager

The figure lists the queue manager attributes that can be modified for SSL by using the **ALTER QMGR** command.

- **SSLCRLNL** (SSL CRL namelist) holds the name for the namelist of authentication information objects.
- **SSLCRYP** (SSL CryptoHardware) holds the name of the parameter string that is required to configure the cryptographic hardware that is present on the system.
- **SSLKEYR** (SSL key repository) holds the name of the SSL key repository.
- **SSLTASKS** (SSL tasks) holds the number of server subtasks to use for processing SSL calls. If you use SSL channels, you must provide two for these tasks.
- **SSLEV** enables or disables the sending of SSL event messages to the channel event queue, when a channel fails to establish an SSL connection.
- **SSLFIPS** specifies whether only FIPS-certified algorithms are used if cryptography is carried out in MQ.



Enabling SSL by using IBM MQ Explorer



Implementing basic security in IBM MQ

© Copyright IBM Corporation 2017

Figure 10-34. Enabling SSL by using IBM MQ Explorer

You can also use MQ Explorer to configure SSL on a queue manager.

Channel attributes for SSL

- Specify cipher specification to use when communicating with the queue manager and ensure that it matches the cipher specification on the target channel
- Use IBM MQ Explorer or the MQSC command **ALTER CHANNEL**
 - SSL_CIPHER** Defines a single cipher specification for an SSL connection
 - SSLPEER** Specifies distinguished name for SSL channel negotiation
 - SSLCAUTH** Specifies whether channel uses SSL for client authentication



Implementing basic security in IBM MQ

© Copyright IBM Corporation 2017

Figure 10-35. Channel attributes for SSL

You can configure the cipher specification to use when you communicate with the queue manager by using the MQSC command **DEFINE CHANNEL** or **ALTER CHANNEL**, or by using MQ Explorer.

The figure lists the attributes available with the **DEFINE CHANNEL** and **ALTER CHANNEL** commands for use with SSL and includes an example of the SSL properties in MQ Explorer.

- SSL_CIPHER** specifies the encryption strength and cipher specifications, for example NULL_MD5 or RC4_MD5_US. The cipher specifications must match at both ends of the channel.
- SSLPEER** specifies the distinguished name (unique identifier) allowed.
- SSLCAUTH** defines whether MQ requires and validates a certificate from the SSL client.

The server authentication of the client uses the public key of the server to encrypt the data. The server can generate the secret key only if it can decrypt that data with the correct private key.

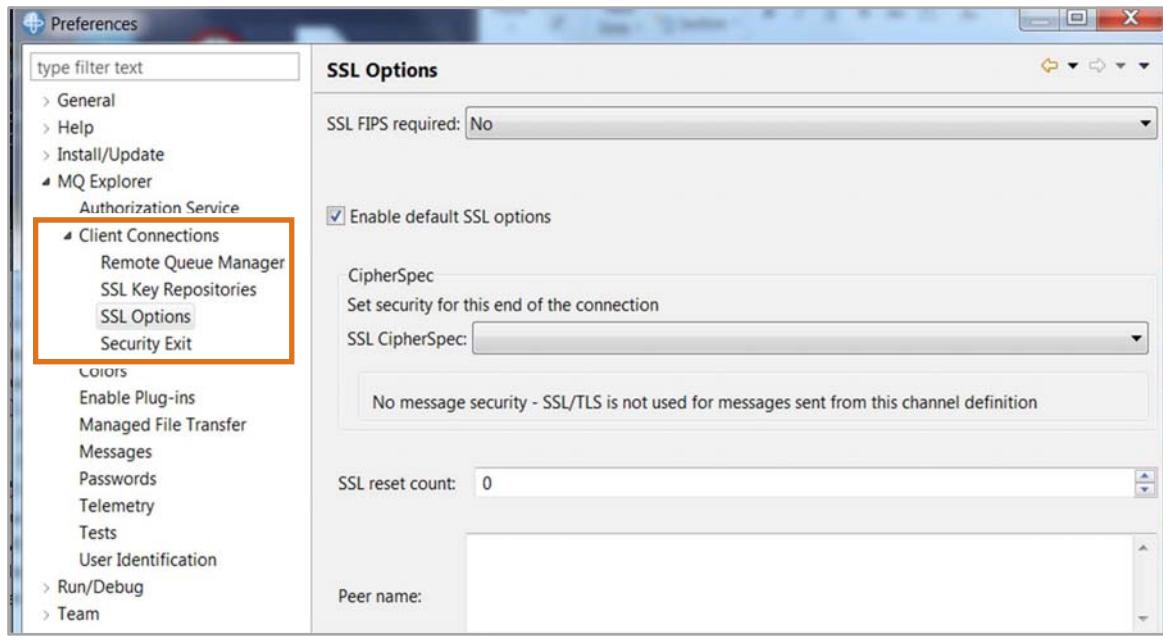
For client authentication, the server uses the public key in the client certificate to decrypt the data that the client sends during the handshake.

If any step of the authentication fails, the handshake fails and the session ends.

SSL configuration and implementation are described in detail in course WM113/ZM113, *IBM MQ V9 Advanced System Administration*.

Default security configuration in IBM MQ Explorer

- Use **Client Connections** tabs in **Preferences** to set default security options on all new client connections
- Default values can be overridden when a new queue manager is added



Implementing basic security in IBM MQ

© Copyright IBM Corporation 2017

Figure 10-36. Default security configuration in IBM MQ Explorer

Client security configuration support is provided in MQ Explorer. The default security preferences for client connections are part of the **Preferences** dialog box.

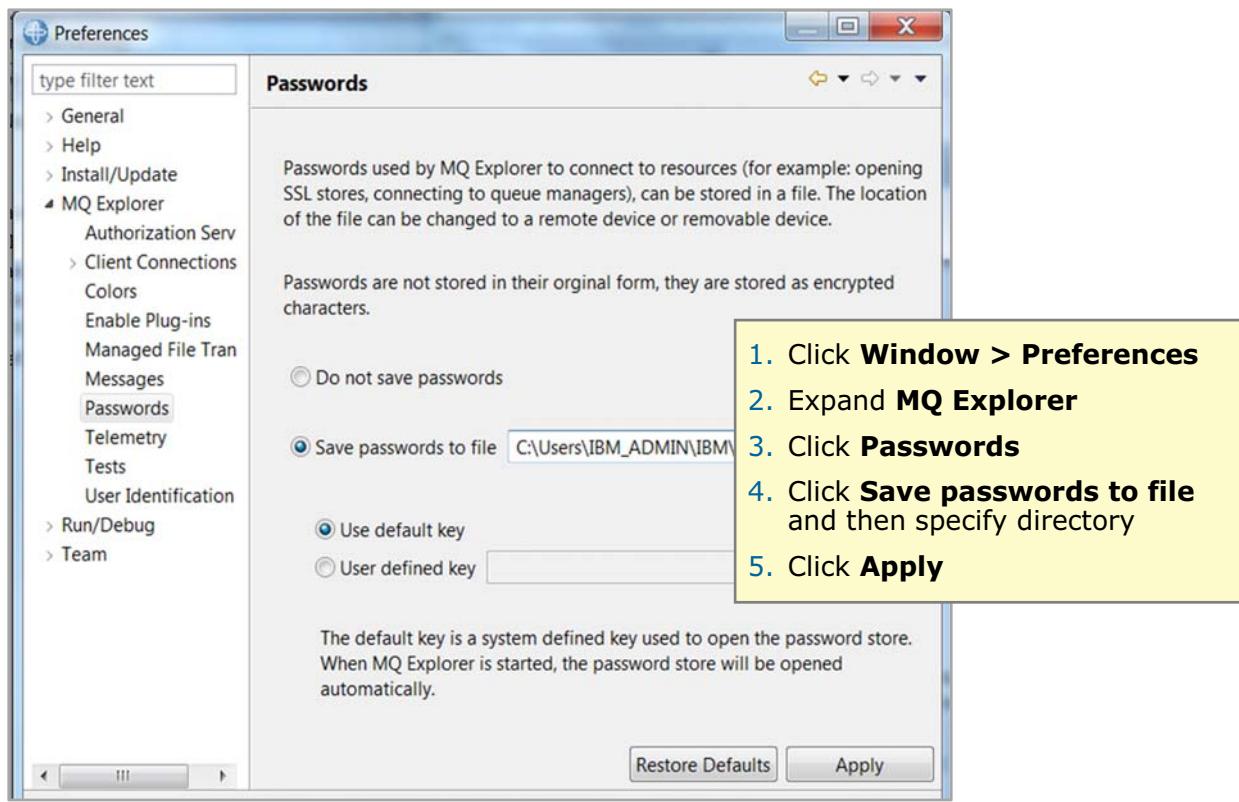
To modify the default security preferences for client connections in MQ Explorer, complete the following steps:

1. Click **Windows > Preferences** to open the **Preferences** dialog box.
2. Expand **MQ Explorer**.
3. Expand **Client Connections**. The default security settings for client connections are now accessible.

The figure shows the **SSL Options** client connections preferences.

IBM Training

Storing IBM MQ Explorer passwords



Implementing basic security in IBM MQ

© Copyright IBM Corporation 2017

Figure 10-37. Storing IBM MQ Explorer passwords

Using MQ Explorer, you can store passwords to a file so that you do not have to enter them every time you want to connect to resources.

The password file can be stored locally, to a remote device, or to a removable device.

To open the **Passwords** preference window, complete the following steps:

1. Click **Windows > Preferences**.
2. Expand **MQ Explorer**.
3. Select **Passwords** to show the **Passwords** view.

Unit summary

- Describe the role of the object authority manager (OAM) to provide security to IBM MQ resources
- Protect IBM MQ resources by using the OAM
- Use some of the OAM control commands
- Describe the Secure Sockets Layer (SSL) and Transport Layer Security (TLS) support that IBM MQ provides
- Implement basic channel authentication

Implementing basic security in IBM MQ

© Copyright IBM Corporation 2017

Figure 10-38. Unit summary

Review questions

1. True or False: You must either restart the queue manager or use the **REFRESH SECURITY** command to refresh the cached view of the configuration for connection authentication.
2. For an application to open a queue by using an alternative user ID, the initial user ID requires:
 - A. OAM delegate authority
 - B. OAM alternate user authority
 - C. Password of alternate user ID
3. Using the OAM, which command would you enter to allow PUT access only to the local queue REBATE.IN on MY.QMGR for the group that is named ASSESSORS?
 - A. `setmqaut -m MY.QMGR -t q -n REBATE.IN -g ASSESSORS +put`
 - B. `setmqaut -m MY.QMGR -t q -n REBATE.IN -p ASSESSORS +put`
 - C. `setmqaut -m MY.QMGR -t q -n REBATE.IN -g ASSESSORS -get`



Implementing basic security in IBM MQ

© Copyright IBM Corporation 2017

Figure 10-39. Review questions

Write your answers down here:

- 1.
- 2.
- 3.

Review answers

1. True or False: You must either restart the queue manager or use the **REFRESH SECURITY** command to refresh the cached view of the configuration for connection authentication.
The answer is True.
2. For an application to open a queue by using an alternative user ID, the initial user ID requires:
 - A. OAM delegate authority
 - B. OAM alternate user authority
 - C. Password of alternate user ID
 The answer is B.
3. Using the OAM, which command would you enter to allow PUT access only to the local queue REBATE.IN on MY.QMGR for the group that is named ASSESSORS?
 - A. setmqaut -m MY.QMGR -t q -n REBATE.IN -g ASSESSORS +put
 - B. setmqaut -m MY.QMGR -t q -n REBATE.IN -p ASSESSORS +put
 - C. setmqaut -m MY.QMGR -t q -n REBATE.IN -g ASSESSORS -get
 The answer is A.

Implementing basic security in IBM MQ

© Copyright IBM Corporation 2017

Figure 10-40. Checkpoint answers



Exercise: Controlling access to IBM MQ

© Copyright IBM Corporation 2017
Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Figure 10-41. Exercise: Controlling access to IBM MQ

In this exercise, you use the OAM utilities to set access control on a queue, and then use sample programs to see the effect of attempting to breach security.

Exercise objectives

- Use the **setmqaut** command to define access control on a queue
- Use the **dspmqaut** command to display the access control on a queue
- Use IBM MQ Explorer to manage authority records
- Enable and monitor authority events
- Test security by using IBM MQ sample programs



[Implementing basic security in IBM MQ](#)

© Copyright IBM Corporation 2017

Figure 10-42. Exercise objectives

See the *Course Exercises Guide* for detailed instructions.

Unit 11. Backing up and restoring IBM MQ messages and object definitions

Estimated time

01:00

Overview

In this unit, you learn about the various ways that IBM MQ maintains messages. You learn about differences between circular and linear logging, the implications of using persistence, and transaction management. You also learn about the methods for capturing and restoring an object image and backing up and restoring IBM MQ object definitions.

How you will check your progress

- Review questions
- Hands-on exercises

References

IBM Knowledge Center for IBM MQ V9

Unit objectives

- Describe how IBM MQ uses logging to record significant changes to the data controlled by the queue manager
- Describe the difference between circular and linear logging
- Develop a method for backing up the IBM MQ environment
- Use a media image to recover objects that become damaged
- Save the queue manager object definitions

Backing up and restoring IBM MQ messages and object definitions

© Copyright IBM Corporation 2017

Figure 11-1. Unit objectives

The need to back up

- Allow recovery of queue managers against possible corruption or loss of data that hardware failures cause
- IBM MQ file system backup can be done as part of a full system backup



Backing up and restoring IBM MQ messages and object definitions

© Copyright IBM Corporation 2017

Figure 11-2. The need to back up

Periodically backing up MQ files and its objects allows the recovery of queue managers against possible corruption that hardware failures cause.

If the hardware fails, a queue manager is forced to stop. If any queue manager log data is lost because of the hardware failure, the queue manager might be unable to restart. Through backing up queue manager data, you might be able to recover some, or all, of the lost queue manager data.

Backing up the IBM MQ file system

- Ensure that queue managers are not running
- Locate directories where the queue manager places its data and log files
 - Default paths on Windows
 - Programs: `C:\Program Files\IBM\MQ\`
 - Data: `C:\ProgramData\IBM\MQ`
 - Default paths on UNIX and Linux
 - Programs: `/usr/mqm`
 - Data: `/var/mqm`
- Take copies of all the data from the queue manager and log file directories, including subdirectories
- Restart queue managers

[Backing up and restoring IBM MQ messages and object definitions](#)

© Copyright IBM Corporation 2017

Figure 11-3. Backing up the IBM MQ file system

The queue manager must be stopped when you back up. If you try to back up a running queue manager, the backup might not be consistent because of updates in progress when the files were copied. If possible, stop your queue manager in an orderly way.

Before you back up, you must know the locations of the MQ components on the file system. The file system layout for MQ varies by operating system. The figure lists the default installation location for MQ software components and data. The paths might be different for your installation. Use the information in the configuration files to find the directories under which the queue manager places its data and its log files.

Some of the directories might be empty, but you need them all to restore from the backup, so save all directories and subdirectories.

Preserve the ownerships of the files. For MQ on UNIX and Linux, you can preserve ownership by using the `tar` command.

Restart the queue manager after the backup is complete.

Overview of IBM MQ objects

- IBM MQ object types include:
 - Queue managers
 - Queues
 - Channels
 - Process definitions
 - Listeners
 - Namelists
 - Client connection channels
 - Services
 - Authentication information objects
- Anything that you define to IBM MQ by using MQSC

Backing up and restoring IBM MQ messages and object definitions

© Copyright IBM Corporation 2017

Figure 11-4. Overview of IBM MQ objects

This figure is a review of the various MQ object types that are defined to MQ. Each object has an object definition that can be backed up.

Why back up IBM MQ object definitions

- To re-create objects that were deleted or missing
- To simplify administration tasks in IBM MQ including:
 - Migration
 - Cloning systems
 - Auditing

Backing up and restoring IBM MQ messages and object definitions

© Copyright IBM Corporation 2017

Figure 11-5. Why back up IBM MQ object definitions

When you update applications, you might want to keep the same object definitions from the queue managers of source environment. By backing up the MQ object definitions, you can restore them on the new queue manager quickly and easily. Otherwise, a manual process would be required to restore the queue managers.

Similarly, when cloning systems (typically for clustering) or for automation of application migrations, a backup provides a quick and simple way to copy the object definitions from one queue manager to another.

If you are auditing your MQ system, backing up the object definition provides a view of all the definitions in one place.

Fundamentally, backing up object definitions is about saving time and effort by using the backup file to re-create the object definition on a new queue manager.

Capturing IBM MQ object definitions

- Save IBM MQ configuration command: `dmpmqcfg`
 - Use IBM MQ commands to save queue manager object definitions
 - Can connect to local queue managers or use client connections to remote systems
- Output from MQSC `DISPLAY` command

Backing up and restoring IBM MQ messages and object definitions

© Copyright IBM Corporation 2017

Figure 11-6. Capturing IBM MQ object definitions

You can capture object definitions by using MQSC and the `dmpmqcfg` command.

- Save the MQ configuration by using the integrated `dmpmqcfg` command (the most comprehensive option).
- Save the output from the MQSC display queue manager, queue, and channel commands in an external file so that you have a reference of the properties and object definitions.

Capturing resource definitions with the DISPLAY command

```
C:\> runmqsc QmgrName
DISPLAY QL(*) ALL
DISPLAY QR(*) ALL
DISPLAY CHL(*) CHLTYPE(*) ALL
```

- Simple method to save properties of all the object definitions
- Output is formatted in single or multiple column formats
 - Needs reformatting to be used for restore purposes

[Backing up and restoring IBM MQ messages and object definitions](#)

© Copyright IBM Corporation 2017

Figure 11-7. Capturing resource definitions with the DISPLAY command

To use MQSC commands to see all the object definitions, enter **DISPLAY** and the **ALL** attribute. This command reports the resource definitions for the specified queue manager. This option does not provide a backup of all objects.

The output is useful for a quick overview of the object definitions; however, it is formatted in single or multi-columns, making it difficult to be used for restoration purposes. Reformatting is required to use the output for restoration purposes.

Custom code can be created to convert these reports into MQSC define commands, but custom code is burdensome and requires updating every time that a change occurs.

Save IBM MQ configuration: `dmpmqcfg`

- Back up object definitions and authorities, restore with MQSC
- Generate reports on objects and their access control
- Use it to rebuild queue manager on a new version of IBM MQ
- Must have appropriate authority to each object that is inquired
- Variety of output formats supported
 - Multi-line MQSC that can be used as direct input to `runmqsc`
 - MQSC on single line
 - MQSC with two lines that contain command and commented strings
 - `setmqaut` statements
 - Security policies statements for IBM MQ Advanced Message Security

[Backing up and restoring IBM MQ messages and object definitions](#)

© Copyright IBM Corporation 2017

Figure 11-8. Save IBM MQ configuration: `dmpmqcfg`

The `dmpmqcfg` command extracts a queue manager's configuration and displays it in MQSC syntax.

The `dmpmqcfg` command can connect to local queue managers or use client connections to remote systems.

You can specify the syntax of the backup report in the `dmpmqcfg` command. The syntax options are:

- Multi-line MQSC that can be used as direct input on a `runmqsc` control command
- MQSC with all attributes on a single line for line comparison
- MQ set authority access (`setmqaut`) statements for Linux, UNIX, and Windows queue managers
- On Linux, grant MQ authority access (`grtmqaut`) for granting access to the objects on iSeries

Backing up queue manager configuration

To take a backup copy of a queue manager's configuration:

1. Ensure that the queue manager is running.
2. Run `dmpmqcfg` command with the following options:
 - Formatting option of `-o mqsc` to create multi-line MQSC that can be used as direct input to MQSC
 - All attributes option `-a`
 - Standard output redirection to write the definitions into a file

Example:

```
dmpmqcfg -o mqsc -m MYQMGR -a > /mq/backups/MYQMGR.mqsc
```

Backing up and restoring IBM MQ messages and object definitions

© Copyright IBM Corporation 2017

Figure 11-9. Backing up queue manager configuration

This figure lists the steps for backing up a queue manager configuration by using the `dmpmqcfg` control command.



Important

When you back up the MQ file system, the queue manager must be stopped.

When you back up an MQ configuration by using the `dmpmqcfg` command, the queue manager must be running.

The example command in the figure creates a backup of all attributes (`-a`) of the queue manager that named MYQMGR (`-m MYQMGR`) in a multi-line MQSC format (`-o mqsc`). The results of the command `dmpmqcfg` are redirected to the `MYQMGR.mqsc` in the `/mq/backups/` directory.

Syntax examples of `dmpmqcfg`

```
dmpmqcfg [-m QMgrName] [-n ObjName] [-c Connection]
[-t ObjType] [-x ExportType] [-o Format] [-a] [-z]
```

- Example 1: Local queue manager, all objects, and authorities:

```
dmpmqcfg -m MYQMGR
```

- Example 2: Local queue manager, all object definitions of SYSTEM queues showing all attributes

```
dmpmqcfg -m MYQMGR -n SYSTEM.* -t queue -x object -a
```

- Example 3: Local queue manager, all channel authentication records, silence any warnings

```
dmpmqcfg -m MYQMGR -x chlauth -z
```

- Example 4: Local queue manager, all authorities in `setmqaut` format

```
dmpmqcfg -m MYQMGR -o setmqaut
```

- Example 5: Dynamic client connection to remote queue manager

```
dmpmqcfg -m RMTQMGR -c "DEFINE CHANNEL(SYSTEM.DEF.SVRCONN) +
CHLTYPE(CLNTCONN) CONNAME('dev.ibm.com(1414)')"
```

Backing up and restoring IBM MQ messages and object definitions

© Copyright IBM Corporation 2017

Figure 11-10. Syntax examples of `dmpmqcfg`

The figure shows examples of the `dmpmqcfg` control command.

Using `dmpmqcfg`: Output MQSC

```
*****
* Script generated on 2016-10-31 at 15.44.12
* Script generated by user 'JSMITH' on host 'dev.IBM.COM'
* Queue manager name: MYQMGR
* Queue manager platform: Windows
* Queue manager command level: (900/900)
* Command issued: dmpmqcfg -m MYQMGR
*****
ALTER QMGR +
* ALTDATE(2016-10-31) +
* ALTTIME(14.32.42) +
CCSID(37) +
CLWLUSEQ(LOCAL) +
* COMMANDQ(SYSTEM.ADMIN.COMMAND.QUEUE) +
* CRDATE(2016-10-31) +
* CRTIME(14.32.42) +
* PLATFROM(WINDOWS) +
* QMID(MYQMGR_2011-10-31_14.32.42) +
SSLKEYR('/QIBM/UserData/ICSS/Cert/Server/Default') +
* VERSION(09000000) +
FORCE
SET AUTHREC +
PROFILE('self') +
PRINCIPAL('QMQM') +
OBJTYPE(QMGR) +
...
...
```

Backing up and restoring IBM MQ messages and object definitions

© Copyright IBM Corporation 2017

Figure 11-11. Using `dmpmqcfg`: Output MQSC

The figure shows an example of a multi-line MQSC report that the `dmpmqcfg` command creates when the `-o mqsc` option is specified.

Some of the attributes in the `ALTER QMGR` command are commented because they are unique to the queue manager. These attributes are not used when you create a queue manager.

Using `dmpmqcfg`: Output `setmqaut`

```
#####
# Script generated on 2016-10-31 at 15.59.09
# Script generated by user 'JSMITH' on host 'dev.IBM.COM'
# Queue manager name: JSMITH
# Queue manager platform: Windows
# Queue manager command level: (900/9--)
# Command issued: dmpmqcfg -m MYQMGR -o setmqaut
#####
setmqaut -m MYQMGR -t qmgr -p QMQM +altusr +chg +connect +crt +dlt +dsp
+inq +set +setall +setid +ctrl +system
setmqaut -m MYQMGR -t qmgr -g QMQMADM +altusr +chg +connect +dlt +dsp
+inq +set +setall +setid +ctrl +system
setmqaut -m MYQMGR -n SYSTEM.ADMIN.ACOUNTING.QUEUE -t queue -p QMQM
+browse +chg +clr +dlt +dsp +get +inq +put +passall +passid +set
+setall +setid
setmqaut -m MYQMGR -n SYSTEM.ADMIN.ACOUNTING.QUEUE -t queue -g QMQMADM
+browse +chg +clr +dlt +dsp +get +inq +put +passall +passid +set
+setall +setid
setmqaut -m MYQMGR -n SYSTEM.ADMIN.ACTIVITY.QUEUE -t queue -p QMQM
+browse +chg +clr +dlt +dsp +get +inq +put +passall +passid +set
+setall +setid
setmqaut -m MYQMGR -n SYSTEM.ADMIN.ACTIVITY.QUEUE -t queue -g QMQMADM
+browse +chg +clr +dlt +dsp +get +inq +put +passall +passid +set
+setall +setid
...
...
```

Backing up and restoring IBM MQ messages and object definitions

© Copyright IBM Corporation 2017

Figure 11-12. Using `dmpmqcfg`: Output `setmqaut`

The figure shows an example of a `dmpmqcfg` report that is created in the `setmqaut` format. You can run this report against a queue manager to restore MQ object authority.

Overview of security definitions

- Security definitions in this context refer to IBM MQ OAM security definitions, not UNIX or Windows security profiles or SSL data
- OAM security definitions
 - Define the authorizations that are given to a specified user group
 - Use `setmqaut` to create

Backing up and restoring IBM MQ messages and object definitions

© Copyright IBM Corporation 2017

Figure 11-13. Overview of security definitions

The `setmqaut` commands that the `dmpmqcfg -o setmqaut` command generates are MQ OAM commands. They do not include Linux, UNIX, or Windows security profiles or SSL data.

It is important to understand that the security definitions refer to the MQ OAM authorities.

Need to back up security definitions

- Restoration of security definitions
- Migration between environments
- Audit purposes

Backing up and restoring IBM MQ messages and object definitions

© Copyright IBM Corporation 2017

Figure 11-14. Need to back up security definitions

Similar to backing up object definitions, backing up security definitions allows users to save time and effort by using the backup file to re-create the security definitions rather than doing it manually.

Regarding audits, backing up security definitions provides a way to view the security definition information in one place and in a format that might be more meaningful to the viewer.

Backing up security definitions

- Use the **dmpmqaut** command to create a report of the current authorizations

Sample output:

```
profile:      a.b.*  
object type: queue  
entity:       user1  
type:        principal  
authority:   get, browse, put, inq
```

- Use the **amqoamd -s** command to create a report of the current authorizations in a **setmqaut** format

Sample output:

```
setmqaut -m AJGMQ1 -n @CLASS -t channel -p andrew@IBM-L3M1562 +crt  
setmqaut -m AJGMQ1 -n @CLASS -t channel -g mqm +crt  
setmqaut -m AJGMQ1 -n @CLASS -t authinfo -p andrew@IBM-L3M1562 +crt  
setmqaut -m AJGMQ1 -n @CLASS -t authinfo -g mqm +crt  
setmqaut -m AJGMQ1 -n SYSTEM.BROKER.ADMIN.STREAM -t queue -p  
MUSR_MQADMIN@IBM-L3M1562 +browse +chg +clr +dlt +dsp +get +inq +put  
+passall +passid +set +setall +setid
```

[Backing up and restoring IBM MQ messages and object definitions](#)

© Copyright IBM Corporation 2017

Figure 11-15. Backing up security definitions

MQ provides three ways to back up the OAM security definitions: **dmpmqcfg**, **dmpmqaut**, and **amqoamd -s**.

The output from the **dmpmqaut** command is formatted in a report style. This output cannot be easily used for restoring security definitions unless it is reformatted.

The output from the **amqoamd -s** command, which is shown in the figure, is generated as a series of **setmqaut** commands. These commands can be used in a command window to re-create the OAM security definitions.

dmpmqaut command

```
dmpmqaut -m QMgrName [-n Profile | -l | -a] -t ObjectType
-s ServiceComponent [-p PrincipalName | -g GroupName]
[-e | -x]
```

Attribute	Description
-n <i>Profile</i>	Name of the profile for which to dump authorizations
-l	Dump only the profile name and type
-a	Generate set authority commands
-t <i>ObjectType</i>	The type of object for which to dump authorizations Possible values are: authinfo, channel, clntconn, listener, namelist, process, queue, qmgr, rqmname, service, or topic
-e	Display all profiles that are used to calculate the cumulative authority that the entity has for the object that is specified in -n <i>Profile</i>
-x	Display all profiles with the same name as specified in -n <i>Profile</i>

Backing up and restoring IBM MQ messages and object definitions

© Copyright IBM Corporation 2017

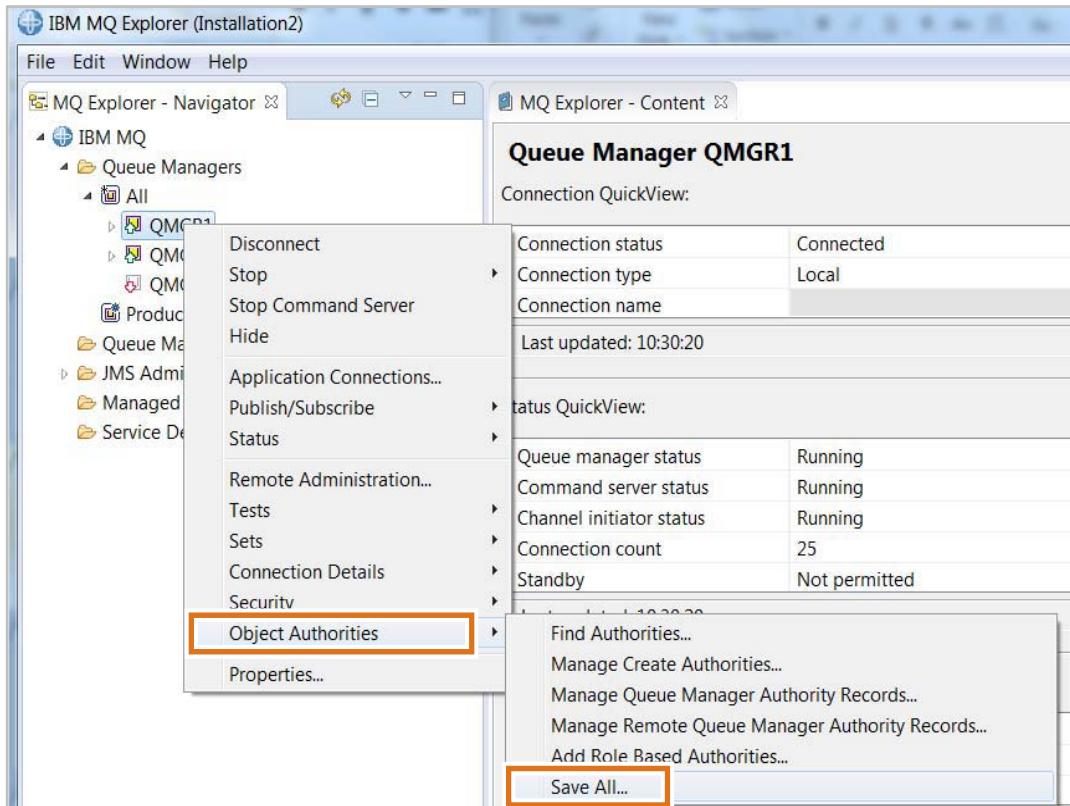
Figure 11-16. dmpmqaut command

You can use the MQ **dmpmqaut** control command to create a report of the current authorizations that are associated with a specified profile.

This figure describes the syntax for the **dmpmqaut** command.



Backing up security definitions in IBM MQ Explorer



Backing up and restoring IBM MQ messages and object definitions

© Copyright IBM Corporation 2017

Figure 11-17. Backing up security definitions in IBM MQ Explorer

As seen in the figure, the security definitions can be exported from the MQ Explorer.

The security definitions that are exported from MQ Explorer are in a format that is similar to the output from `amqoamd -s` command, which is a series of `setmqaut` commands.

Recovering persistent messages

- To restart, a queue manager requires:
 - Log records that are written since the last checkpoint
 - Log records that are written by transactions that were still active at the time queue manager stopped
- Persistent messages are recovered automatically when queue manager is restarted
- A damaged local queue can be detected only later
 - Reported as "object damaged"
 - Normally must be recovered manually

[Backing up and restoring IBM MQ messages and object definitions](#)

© Copyright IBM Corporation 2017

Figure 11-18. Recovering persistent messages

The queue manager recovers any damaged object that would prevent it from starting, but it would not normally include a local queue that is damaged. Such a queue can be detected later when an attempt is made to access it.

To restart, a queue manager requires:

- Log records that are written since the last checkpoint.
- Log records that are written by transactions that were still active when the queue manager stopped. Uncommitted persistent messages, put or got inside these transactions, are rolled back during restart.

Types of logs

Circular	Linear
Log files are viewed as a closed loop	Log files are viewed as a sequence
Amount of disk space that is required for the log does not increase with time	Log file is never deleted but becomes inactive when it contains no entries that are required to restart the queue manager
	Can be archived when it becomes inactive
	Required for media recovery

[Backing up and restoring IBM MQ messages and object definitions](#)

© Copyright IBM Corporation 2017

Figure 11-19. Types of logs

MQ records all significant changes to the data controlled by the queue manager in a recovery log. Changes include creating and deleting objects, persistent message updates, transaction states, changes to object attributes, and channel activities.

The log contains the information that is required for recovering all updates to message queues by:

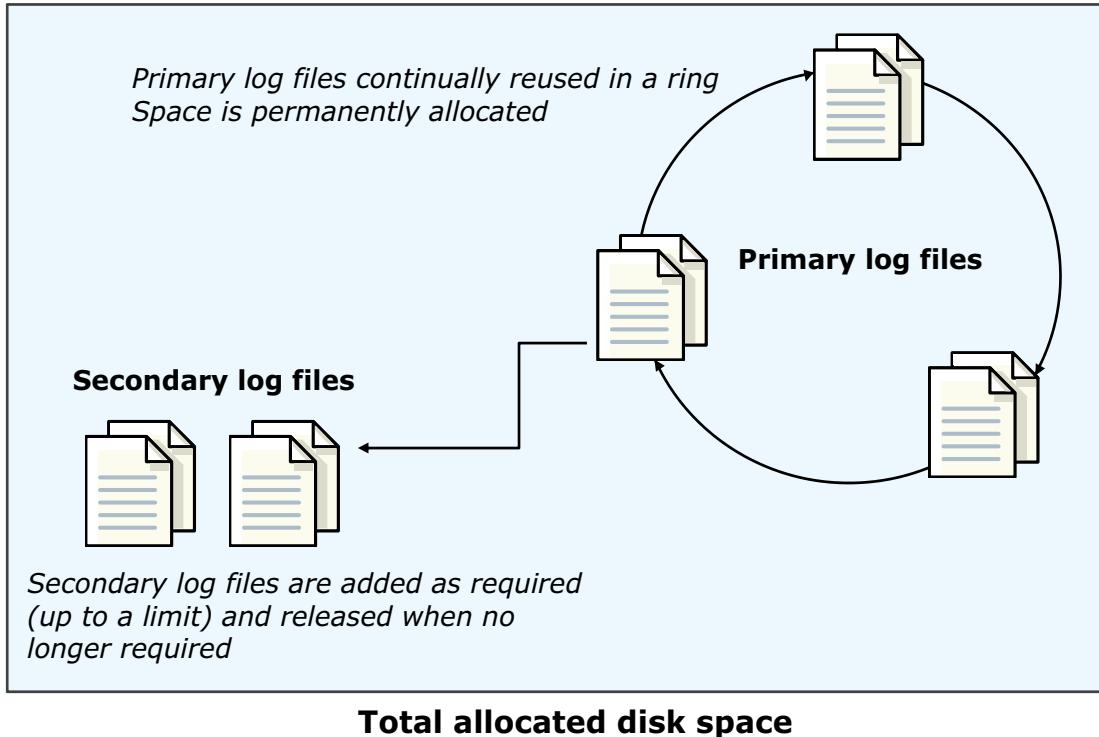
- Keeping records of queue manager changes
- Keeping records of queue updates for use by the restart process
- Allowing you to restore data after a hardware or software failure

The type of logging is selected when a queue manager is created. Unless you request linear logging when you create a queue manager, circular logging is provided by default.

A log checkpoint is taken periodically and provides a point of consistency for the queue manager data. A checkpoint is recorded in the log as a series of checkpoint records. Checkpoints reduce restart time by minimizing the log replay required.

MQ generates checkpoints automatically. They are taken when the queue manager starts, at shutdown, when logging space is running low, and after every 10,000 operations logged.

Circular logs



Backing up and restoring IBM MQ messages and object definitions

© Copyright IBM Corporation 2017

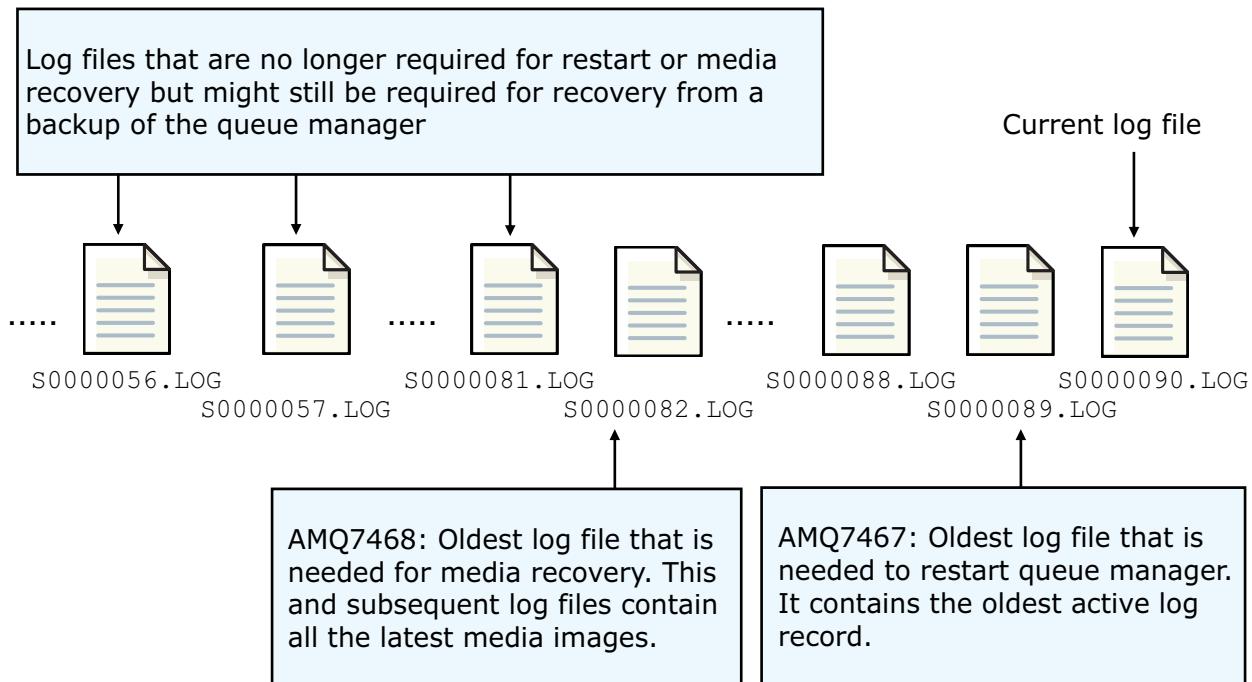
Figure 11-20. Circular logs

With circular logging, the log files are viewed as a closed ring. A log file becomes available for reuse when it contains no active log records. An active log record is one that is still required to restart the queue manager.

With circular logging, MQ can recover messages that follow a system failure but is unable to recover messages that follow a media failure. It has the advantage that the amount of disk space that is required for the log does not increase with time.

When you specify the type of logging, you can also specify the size and location of the log files if you do not want to accept the default values as specified in the MQ configuration file.

Linear logs



Safe implementation of linear logs requires regular oversight and management of the log files

[Backing up and restoring IBM MQ messages and object definitions](#)

© Copyright IBM Corporation 2017

Figure 11-21. Linear logs

With linear logs, the log files are viewed as a sequence. A log file is never deleted, but it becomes inactive when it contains no active log records. New log files are added to the sequence as required. Space is not reused. A linear log can recover from a media failure, but it requires the regular archival of inactive log files.



Important

If the logs are not actively managed, a normally operating linear logged queue manager can fail. Depending on message traffic load, message persistence, excessive MAXDEPTH and MAXMSGL settings, and disk space allocations, the logs use all available space and the queue manager stops unless you take steps to prevent it.

IBM provides several SupportPacs that manage linear log files. Typically, these tools use a scripting language to identify inactive log extents and dispose of them. The queue manager provides commands to inquire on which extents are active, as an option for administrators who want to provide their own instrumentation for this process.

Dumping the log

- Use **dmpmqlog** to dump a formatted version of the log
- Queue manager must be stopped
- By default, the dump starts from the head of the log
- Optionally, log dump can start from:
 - Base of the log
 - Log record that a specified *log sequence number* (LSN) identifies
 - Log file that a specified *extent number* (linear logs only) identifies
- Log records include:
 - Put and get of persistent messages
 - Transaction events
 - Creation, alteration, and deletion of MQ objects

[Backing up and restoring IBM MQ messages and object definitions](#)

© Copyright IBM Corporation 2017

Figure 11-22. Dumping the log

The **dmpmqlog** control command can be used to dump a formatted version of the log. It can be used only when the queue manager is not running.

The head of the log is the checkpoint that starts the active portion of the log. Normally, the head of the log would be the most recent checkpoint. However, if transactions were active when the queue manager stopped and uncommitted persistent messages were inside these transactions before the most recent checkpoint, the head of the log might be positioned at an earlier checkpoint.

- The base of the log is the first log record in the log file that contains the head of the log.
- A unique log sequence number (LSN) identifies each log record.
- Each log file has a file name of the form Snnnnnnn.LOG where nnnnnnnn is the extent number.

Options on the **dmpmqlog** command specify a different starting point for the log. The log includes the following information:

- Header information about the log, for example, whether the log is circular or linear, or the LSN of the log record at the head of the log
- Start queue manager and stop queue manager log records
- Start checkpoint and end checkpoint log records

- Put message and get message log records for persistent messages only and contains hex values for the application data and the message descriptor
- Various types of log records for events that are associated with transactions: for example, start transaction, prepare transaction, and commit transaction
- Log records that are associated with the creation, alteration, and deletion of MQ objects

Damaged objects and media recovery

- IBM MQ objects can be marked as damaged
 - Corrupted data in the queue file
 - Missing queue file
 - Disk failure
- Damaged objects can be deleted
- A damaged object can be re-created from a **linear log**
 - Known as *media recovery*
 - Queue manager records media images automatically at certain times
 - Use control command **rcdmqimg** to record media image of a local queue regularly
- Media recovery
 - Automatic if a damaged object is detected during restart
 - For a local queue, it is normally done by using control command **rcrmqobj**

Backing up and restoring IBM MQ messages and object definitions

© Copyright IBM Corporation 2017

Figure 11-23. Damaged objects and media recovery

A queue manager does not normally detect that a local queue is damaged during restart. If it was storing uncommitted persistent messages that were put or got inside a transaction that was still active when the queue manager stopped, it would detect a damaged local queue. In this case, the queue manager would automatically re-create the local queue as it must be able to roll back the transaction that did not complete. As a result, a damaged local queue is normally detected only when an attempt is made to access it.

Recording media image to a log

- Use `rcdmqimg` command to write an image of an object, or group of objects, to the log for use in media recovery
 - Requires linear logging
 - Moves the log sequence number forward and frees up old log files for archival or deletion
 - Can take a long time to run if queues contain many messages

Example: Record an image of queue manager QMGR1 in the log

```
rcdmqimg -t qmgr -m QMGR1
```

- t Type of object to record
- m Queue manager

Figure 11-24. Recording media image to a log

The control command to record a media image is `rcdmqimg`.

The object name can have a trailing asterisk to record any objects with names that match the portion of the name before the asterisk.

Re-creating an object from an image

- Use the **rcrmqobj** command to re-create an object, or group of objects, from media images that are contained in the log
 - Can be used with using linear logging only
 - Use on a running queue manager

Examples

- Re-create all local queues for the default queue manager:
`rcrmqobj -t ql *`
- Re-create all remote queues that are associated with queue manager QMGR1:
`rcrmqobj -m QMGR1 -t qr *`

[Backing up and restoring IBM MQ messages and object definitions](#)

© Copyright IBM Corporation 2017

Figure 11-25. Re-creating an object from an image

The **rcrmqobj** control command can be used to re-create a damaged object.



Information

Although you can use the control commands **rcdmqimg** and **rcrmqobj** with other types of MQ objects, the simplest way to re-create such an object is to rerun the MQ command that created it. This advice applies to an alias queue, a model queue, a local definition of a remote queue, a process object, and a channel.

Valid object types for recording and re-creating images

* or all	All object types
authinfo	Authentication information object for SSL channel security
channel or chl	Channels
clntconn or clcn	Client connection channels
clchltab	Client channel table
listener or lstr	Listener
namelist or nl	Namelists
process or prcs	Processes
queue or q	All types of queue
qalias or qa	Alias queues
qlocal or ql	Local queues
qmodel or qm	Model queues
qremote or qr	Remote queues
service or srvc	Service
syncfile	Channel synchronization file
topic or top	Topics

Backing up and restoring IBM MQ messages and object definitions

© Copyright IBM Corporation 2017

Figure 11-26. Valid object types for recording and re-creating images

When you use a control command to record a media image and re-create objects, you can specify the object type to record and re-create. This figure lists the object types and syntax.

Unit summary

- Describe how IBM MQ uses logging to record significant changes to the data controlled by the queue manager
- Describe the difference between circular and linear logging
- Develop a method for backing up the IBM MQ environment
- Use a media image to recover objects that become damaged
- Save the queue manager object definitions

Backing up and restoring IBM MQ messages and object definitions

© Copyright IBM Corporation 2017

Figure 11-27. Unit summary

Review questions

1. If you want to re-create your security definitions, which command would be the best method to back them up?
 - A. **dmpmqaut**
 - B. **amqoamd -s**
 - C. **bkupaut**
2. True or False: It is acceptable to back up IBM MQ files while the IBM MQ queue managers are running.



Backing up and restoring IBM MQ messages and object definitions

© Copyright IBM Corporation 2017

Figure 11-28. Review questions

Write your answers down here:

- 1.
- 2.

Review answers

1. If you want to re-create your security definitions, which command would be the best method to back them up?
A. dmpmqaut
B. amqoamd -s
C. bkupaut

The answer is B. The output is given as a series of setmqaut commands, which can be used on the command interface to re-create the security definitions.

2. True or False: It is acceptable to back up IBM MQ files while the IBM MQ queue managers are running.
The answer is False. The backup is not consistent because of updates in progress when the files were copied.



Exercise: Using a media image to restore a queue

Backing up and restoring IBM MQ messages and object definitions

© Copyright IBM Corporation 2017

Figure 11-30. Exercise: Using a media image to restore a queue

In this exercise, you capture a media image of a queue, deliberately damage the queue, and then restore it.

Exercise objectives

- Capture an object media image
- Re-create an IBM MQ object from an object media image



Backing up and restoring IBM MQ messages and object definitions

© Copyright IBM Corporation 2017

Figure 11-31. Exercise objectives

See the *Course Exercises Guide* for detailed instructions.

Exercise: Backing up and restoring IBM MQ object definitions

Backing up and restoring IBM MQ messages and object definitions

© Copyright IBM Corporation 2017

Figure 11-32. Exercise: Backing up and restoring IBM MQ object definitions

In this exercise, you use the `dumpmqcfg` command to unload a queue manager's object definitions. You then create a queue manager and load the same definitions, and use MQSC commands or IBM MQ Explorer to show that the definitions are the same.

Exercise objectives

- Use IBM MQ commands to back up object definitions of a queue manager
- Use IBM MQ commands to upload object definitions to another queue manager



[Backing up and restoring IBM MQ messages and object definitions](#)

© Copyright IBM Corporation 2017

Figure 11-33. Exercise objectives

See the *Course Exercises Guide* for detailed instructions.

Unit 12. Introduction to queue manager clusters

Estimated time

01:00

Overview

In this unit, you learn about the basic concepts of queue manager clustering. The unit provides an overview of queue manager cluster components and definitions that are required for setting up a simple clustered environment.

How you will check your progress

- Review questions
- Hands-on exercise

References

IBM Knowledge Center for IBM MQ V9

Unit objectives

- Describe a cluster and list the components that are involved
- Describe the difference between a full and a partial repository queue manager
- Configure a basic cluster
- Describe the administration tasks that must be considered in a clustered environment

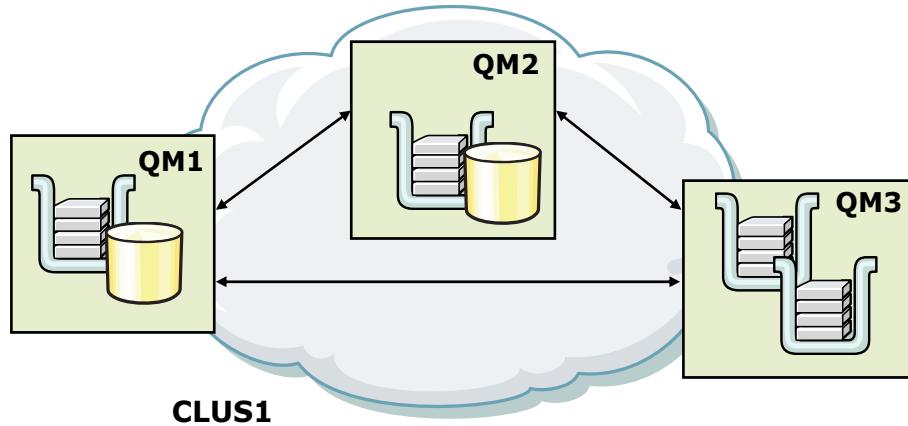
Introduction to queue manager clusters

© Copyright IBM Corporation 2017

Figure 12-1. Unit objectives

What is a cluster?

- Named group of queue managers that make the queues that they host available to other queue managers in the cluster



Introduction to queue manager clusters

© Copyright IBM Corporation 2017

Figure 12-2. What is a cluster?

A cluster is a collection of queue managers that can be on different servers and operating systems, and typically serve a common application.

Every queue manager can make the queues that it hosts available to every other queue manager in the cluster, without the need for (remote) queue definitions. Cluster-specific objects also remove the need for explicit channel definitions and transmission queues for each destination queue manager.

The queue managers in a cluster often assume the role of a client or a server. The servers host the queues that are available to the members of the cluster and applications that process these messages and generate responses. The clients PUT messages to the server queues and might receive response messages.

Queue managers in a cluster normally communicate directly with each other, although typically, many of the client systems never need to communicate with other client systems.

Clustering was introduced in the prerequisite course WM103/ZM103, *A Technical Introduction to IBM MQ*.

Benefits of clustering

- Improved performance and distributed workload of message traffic
 - If a cluster contains more than one instance of the same queue, IBM MQ selects a queue manager to route a message to
 - IBM MQ uses a cluster workload management algorithm and cluster workload-specific attributes to determine the optimal queue manager
- Greater resilience to system failures through redundancy
- Simplified administration and flexible connectivity
 - Not necessary to explicitly define channels, remote-queue definitions, or transmission queues for every destination in the cluster

Introduction to queue manager clusters

© Copyright IBM Corporation 2017

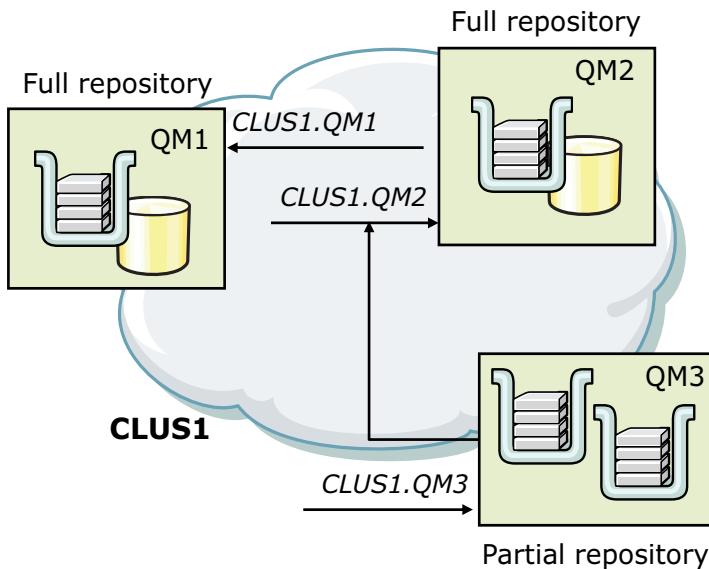
Figure 12-3. Benefits of clustering

Clusters simplify the administration of MQ networks, which usually require many object definitions for channels, transmission queues, and remote queues.

Clusters can be used to distribute the workload of message traffic across queues and queue managers in the cluster. Such distribution allows the message workload of a single queue to be distributed across equivalent instances of that queue that are on multiple queue managers.

The distribution of the workload can be used to achieve greater resilience with system failures, and to improve the scaling performance of active message flows in a system.

Overview of cluster components



- *Full repository queue manager* hosts a complete set of information about every queue manager in the cluster
- *Partial repository queue manager* contains information about those queue managers with which it needs to exchange messages
- *Cluster queues*
 - Application queues
 - Transmission queues
- *Cluster channels*
 - Cluster-receiver
 - Cluster-sender

Introduction to queue manager clusters

© Copyright IBM Corporation 2017

Figure 12-4. Overview of cluster components

A cluster can contain two or more queue managers. The example cluster in the figure contains three queue managers in a cluster that is named CLUS1: QM1, QM2, and QM3.

Two of queue managers, QM1 and QM2, are configured as *full repository queue managers*. Full repository queue managers contain information about all the queue managers in the cluster. The repositories are represented in the figure by the shaded cylinders. The other queue manager in this cluster is a *partial repository queue manager*. A partial repository queue manager holds records for local objects and remote objects that are used locally.

All the queue managers in the cluster can host local queues that are accessible to any other queue manager in the cluster. These queues are called *cluster queues*. As with distributed queuing, an application uses an MQPUT call to put a message on a cluster queue at any queue manager. An application uses the MQGET call to retrieve messages from a cluster queue on the local queue manager.

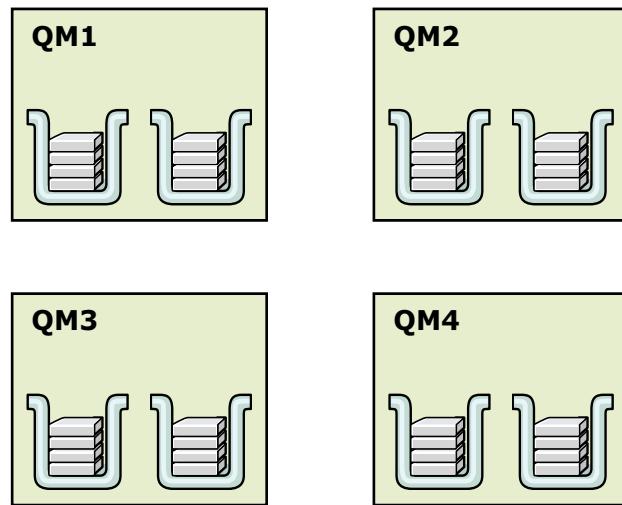
Each queue manager has a *cluster-receiver channel* for the receiving end of a channel. Include the name of the cluster and the name of the queue manager that is the destination of that channel in each channel name. A cluster-receiver channel is similar to a receiver channel that is used in distributed queuing, but in addition to carrying messages, this channel also carries information about the cluster.

Each queue manager also has a *cluster-sender channel* definition for the sending end of a channel. A cluster-sender channel is similar to a sender channel that is used in distributed queuing, but in addition to carrying messages, this channel also carries information about the cluster.

Distributed queuing versus clustering (1 of 3)

- Example shows the definitions that are required in a network with four queue managers, each with two local queues in the configuration:
 - Using distributed queuing
 - Using a cluster

***Four queue managers,
each with two local queues***



Introduction to queue manager clusters

© Copyright IBM Corporation 2017

Figure 12-5. Distributed queuing versus clustering (1 of 3)

IBM MQ clusters allow queue instances to be added. If you do not use clusters, your queue managers are independent and communicate by using distributed queuing. If one queue manager needs to send messages to another, it must define:

- A transmission queue
- A channel to the remote queue manager
- Remote queues

If queue managers are grouped in a cluster, the queue managers can make the queues that they host available to every other queue manager in the cluster. Any queue manager can send a message to any other queue manager in the same cluster without explicit channel definitions, remote-queue definitions, or transmission queues for each destination. Every queue manager in a cluster has a single transmission queue from which it can transmit messages to any other queue manager in the cluster.

Each queue manager in a cluster needs to define only:

- One cluster-receiver channel on which to receive messages
- One cluster-sender channel that points to one of the full repository queue managers

The MQ network in this comparison scenario uses four queue managers. Each queue manager has two application queues. Consider how many definitions are needed to connect these queue managers with distributed queuing when compared with a cluster.

Distributed queuing versus clustering (2 of 3)

- Definitions that are required for distributed queuing with four queue managers

Definitions	Number per queue manager	Total for four queue managers
Sender channel on which to send messages to every other queue manager	3	12
Receiver channel on which to receive messages from every other queue manager	3	12
Transmission queue for a transmission queue to every other queue manager	3	12
Local queue for each local queue	2	8
Remote queue for each remote queue to which this queue manager wants to put messages	6	24

Introduction to queue manager clusters

© Copyright IBM Corporation 2017

Figure 12-6. Distributed queuing versus clustering (2 of 3)

The figure shows the number of definitions that are required to use distributed queuing with four queue managers and two application queues on each queue manager.

The maximum number of definitions can be as many as 17 on each queue manager, which is a total of 68 for this network. You might be able to reduce the number of definitions by using generic receiver-channel definitions, for example.

Distributed queuing versus clustering (3 of 3)

- Definitions when using a cluster with four queue managers:
 - Requires one CLUSSDR and one CLUSRCVR channel definition at each queue manager
 - Separately defined transmission queues are not required
 - Remote-queue definitions are not required

Definitions	Number per queue manager	Total for four queue managers
Cluster-sender channel on which to send messages to a repository queue manager	1	4
Cluster-receiver channel on which to receive messages from other queue managers in the cluster	1	4
Transmission queue for a transmission queue to every other queue manager	0	0
Cluster queue for each local queue	2	8
Remote queue for each remote queue to which this queue manager wants to put messages	0	0

Introduction to queue manager clusters

© Copyright IBM Corporation 2017

Figure 12-7. Distributed queuing versus clustering (3 of 3)

This figure shows the number of definitions that are required when you use clustering to support the simplified administration example with four queue managers.

To set up this cluster of queue managers (with two full repositories), you would need four definitions on each queue manager; a total of 16 definitions. You would also need to alter the queue-manager definitions for two of the queue managers to make them full repository queue managers for the cluster.

The cluster-sender and cluster-receiver channel definitions are made one time. When the cluster is in place, you can add or remove queue managers (other than the repository queue managers) without any disruption to the other queue managers. This process significantly reduces the number of definitions that are required to set up a network that contains many queue managers.

You have less risk for errors because you define fewer objects.

- Object names always match, for example, the channel name in a sender-receiver pair.
- The transmission queue name that is specified in a channel definition always matches the correct transmission queue definition or the transmission queue name that is specified in a remote queue definition.
- After a cluster is set up, cluster queues can be moved from one queue manager to another within the cluster without any system management on any other queue manager. You have no

chance of forgetting to delete or modify channels, remote queues, or transmission queues. You can add new queue managers to a cluster without any disruption to the existing network.

Compare the tables on both figures so that students see the difference in the configuration and recognize that clustering simplifies administration.

Steps to set up a basic cluster

1. Determine which queue managers to add to the cluster (organization of the cluster and naming conventions)
2. Establish which queue managers are to hold full repositories
3. Alter the full repository queue manager definitions to add the repository information
4. Define the cluster-receiver (CLUSRCVR) channels on each queue manager
5. Define the cluster-sender (CLUSSDR) channels on each queue manager
6. Define the cluster queues
7. Verify and test the cluster

MQSC commands
or
IBM MQ Explorer **Create Cluster** wizard

Figure 12-8. Steps to set up a basic cluster

The figure lists the basic steps for setting up a cluster.

You can also use MQSC or one of the wizards that are supplied with IBM MQ Explorer to create a cluster.

To use IBM MQ Explorer, right-click the **Queue Manager Clusters** folder, click **New > Queue Manager Cluster**, and follow the instructions in the wizard.

Each of these steps is described in detail next. You also define a simple cluster in the lab exercise for this unit.

Considerations for a full repository

- Full repository queue managers host a complete set of information about every queue manager in the cluster
 - Cluster information is stored in the form of messages on SYSTEM.CLUSTER.REPOSITORY.QUEUE
- Ensure that full repository queue managers are highly available
 - Avoid single point of failure
 - Define two full repositories to improve availability
 - Locate on highly available servers
 - Avoid other application workload if possible
- Full repository queue manager must be fully interconnected
 - Define a cluster-receiver channel at each queue manager
 - Define one cluster-sender channel between the full repository queue managers

Figure 12-9. Considerations for a full repository

It is best to have two full repositories in a cluster so that the cluster has a backup repository. You interconnect the full repository queue managers by defining cluster-sender channels between them.

When a queue manager sends out information about itself or requests information about another queue manager, the information or request is sent to the full repositories. A full repository that is named on a cluster-sender channel handles the request whenever possible, but if the chosen full repository is not available, another full repository is used. When the first full repository becomes available again, it collects the most recent information from the other full repository so that they contain the same information.

Defining full repository queue managers

- On each queue manager that is to hold a full repository, use the **ALTER QMGR** command to change the queue manager definition
 - REPOS** attribute defines this queue manager as a repository for the named cluster

Example: **ALTER QMGR REPOS (CLUS1)**

- REPOSNL** attribute defines this queue manager as a repository for all clusters that are specified in the specified name list

Example:

```
DEFINE NAMELIST(CLUSTERLIST) +
DESCR('List of clusters that I host') +
NAMES(CLUS1, CLUS2, CLUS3)

ALTER QMGR REPOSNL(CLUSTERLIST)
```

Figure 12-10. Defining full repository queue managers

In each cluster, you must select at least one, preferably two, queue managers to hold full repositories. A cluster can work adequately with only one full repository, but two full repositories improve availability.

The full repository queue managers store information about all the queue managers in the cluster in the SYSTEM.CLUSTER.REPOSITORY.QUEUE. The full repository queue managers store queue manager information for 30 days. To prevent this data from expiring, the queue manager resends information about its configuration after 27 days. When data expires, it is not immediately removed; instead, it has a grace period of 60 days. If, during the grace period, no changes occur, then the data is removed. This grace period provides a queue manager that was temporarily out of service at the expiry date the right to rejoin if it does not stay disconnected from the cluster for 90 days. Then, that queue manager no longer is part of the cluster.

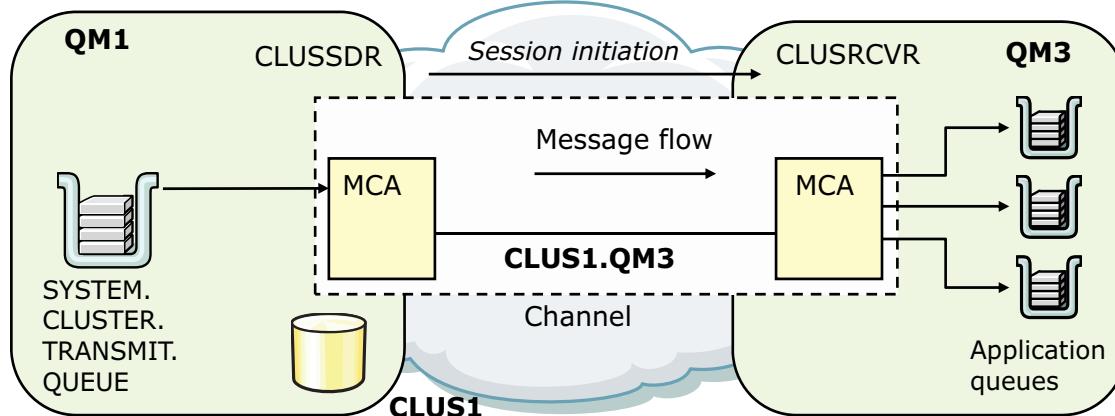
To designate a queue manager as a cluster repository queue manager, use the **ALTER QMGR** command.

To specify that a queue manager holds a full repository for one cluster, use the **ALTER QMGR** command and specify the **REPOS(clustername)** attribute. The value in the **REPOS** attribute is the name of the cluster. In the example in the figure, the cluster name is **CLUS1**.

To specify that a queue manager holds a full repository for several clusters, define a namelist that contains the names of the clusters. Then, use the **REPOSNL(namelist)** attribute on the **ALTER QMGR** command.

Cluster channels

- Cluster-receiver (CLUSRCVR)
 - Define one for each cluster queue manager
 - Enables queue managers to automatically define corresponding cluster-sender channels
- Cluster-sender (CLUSDR)
 - Explicitly define one between the full repositories and one from each partial repository to one full repository
 - Other cluster-sender channels are defined automatically



Introduction to queue manager clusters

© Copyright IBM Corporation 2017

Figure 12-11. Cluster channels

In a cluster, each queue manager has a cluster-receiver channel on which it can receive messages and information about the cluster. You must manually define a cluster-receiver channel on each queue manager in the cluster.

In a cluster, each queue manager has a cluster-sender channel on which it can send cluster information to one of the full repository queue managers. Queue managers can also send messages to other queue managers on cluster-sender channels. Manually define a cluster-sender channel between the full repository queue managers and from each partial repository queue manager to one of the full repository queue managers. The other cluster-sender channels are defined automatically.

Define cluster-receiver channels

- Define one cluster-receiver channel for each queue manager to receive messages: **CHLTYPE (CLUSRCVR)**
- **TRPTYPE** identifies the transport protocol
- **CONNNAME** defines queue manager physical location
- **CLUSTER** identifies the cluster

Example cluster-receiver channel on QM1:

```
DEFINE CHANNEL(CLUS1.QM1) CHLTYPE(CLUSRCVR) TRPTYPE(TCP) +
CONNNAME(localhost(5002)) CLUSTER(CLUS1) +
DESCR('Cluster-receiver channel for QM1 on cluster CLUS1')
```

[Introduction to queue manager clusters](#)

© Copyright IBM Corporation 2017

Figure 12-12. Define cluster-receiver channels

A cluster-receiver channel is a channel definition of the **TYPE(CLUSRCVR)** on which a cluster queue manager can receive messages from within the cluster. Through the definition of this object, a queue manager is advertised to the other queue managers in the cluster so that they can automatically define the appropriate cluster-sender (**CLUSSDR**) channels. At least one cluster-receiver channel is required for each cluster queue manager.

Each cluster channel is given a unique name. The **CLUSTER** attribute identifies the cluster, which in this example is CLUS1.

The naming convention that is used in this example includes the cluster name and the queue manager name in the channel definition. This convention makes administration easier in cases where the queue manager is shared with other clusters. In this example, the channel name is CLUS1.QM1.

The connection name (**CONNNAME**) is the network address and port of the queue manager server. The network address can be entered as a DNS host name, or an IP address. If the port number is not specified, the default TCP listener port for MQ (1414) is used.

Define cluster-sender channel

- Manually define:
 - One cluster-sender channel between the full repository queue managers
 - One cluster-sender channel from each partial repository to one full repository queue manager
- Channel names on the CLUSSDR definitions must match cluster names on the corresponding CLUSRCVR definitions
- **CONNNAME** refers to the target queue manager

Example cluster-sender channel from QM1 to QM2:

```
DEFINE CHANNEL(CLUS1.QM2) CHLTYPE(CLUSSDR) TRPTYPE(TCP) +
CONNNAME(QM2.CHSTORE.COM) CLUSTER(CLUS1) +
DESCR('Cluster-sender channel from QM1 +
to repository at QM2 on CLUS1')
```

Introduction to queue manager clusters

© Copyright IBM Corporation 2017

Figure 12-13. Define cluster-sender channel

A cluster-sender channel is a channel definition of the **TYPE(CLUSSDR)** on which a cluster queue manager can send messages to another queue manager in the cluster. This channel is used to notify the repository of any changes of the status of the queue manager, for example, the addition or removal of a queue. This channel is also used to send messages to the other queue managers in the cluster.

Manually define one cluster-sender channel from each queue manager in the cluster to one of the full repository queue managers. The other cluster-sender channels are created automatically when the queue manager needs the channel. Manually defining the other cluster-sender channels results in duplicate channels.

For a group of full repositories to be fully connected, manually define one cluster-sender channel from each full repository to the other full repositories.

The cluster-sender name must match the cluster-receiver name on the other end of the channel.

The figure includes an example for defining a cluster-sender channel with the **DEFINE CHANNEL** command and TCP as the transport protocol.

Display information about cluster queue manager

- **DISPLAY CLUSQMGR()** command returns information about all cluster queue managers:

CLUSTER	Name of cluster
QMTYPE	Identifies queue manager as full or partial repository
CHANNEL	Cluster-receiver channel name for the queue manager
- Channel attributes are returned for all or the (generically) named channels

Example:

```
DISPLAY CLUSQMGR(*) QMTYPE
AMQ8441: Display Cluster Queue Manager Details
  CLUSQMGR(QMC01)
  CLUSTER(CLUS1)
  CHANNEL(CLUS1.QMC01)
  QMTYPE(REPOS)
AMQ8441: Display Cluster Queue Manager Details
  CLUSQMGR(QMC02)
  CLUSTER(CLUS1)
  CHANNEL(CLUS1.QMC02)
  QMTYPE(REPOS)
```

Introduction to queue manager clusters

© Copyright IBM Corporation 2017

Figure 12-14. Display information about cluster queue manager

The **DISPLAY CLUSMGR** command returns cluster information about queue managers in a cluster, which is stored in the local SYSTEM.CLUSTER.REPOSITORY.QUEUE.

The **DEFTYPE** (definition type) parameter can be set to one of the following options:

- **CLUSSDR**: Cluster-sender channel from an explicit definition
- **CLUSSDRA**: Cluster-sender by automatic definition
- **CLUSSDRB**: Cluster-sender channel, both from an explicit definition and by automatic definition
- **CLUSRCVR**: Cluster-receiver channel

Display channel status

- Use **DISPLAY CHSTATUS(*)** command to check the channel definitions

Example:

```
DISPLAY CHSTATUS(*)  
AMQ8417: Display Channel Status details.  
  CHANNEL(CLUS1.QM2)           XMITQ( )  
  CONNAME(9.20.40.24)          CURRENT  
  CHLTYPE(CLUSRCVR)           STATUS(RUNNING)  
  RQMNAME(QM1)  
  
AMQ8417: Display Channel Status details.  
  CHANNEL(CLUS1.QM1)           XMITQ(SYSTEM.CLUSTER.TRANSMIT.QUEUE)  
  CONNAME(9.20.51.25)          CURRENT  
  CHLTYPE(CLUSSDR)            STATUS(RUNNING)  
  RQMNAME(QM1)
```

Introduction to queue manager clusters

© Copyright IBM Corporation 2017

Figure 12-15. Display channel status

Check the channel status for the cluster by entering: **DISPLAY CHSTATUS(*)**

As shown in the figure, the channel status includes the channel name, transmission queue name, connection name, channel type, status, and remote queue manager name.

Define local queues to the cluster

- Clustered queues are application queues that a cluster queue manager hosts and makes available to other queue managers in the cluster
- Specify the CLUSTER keyword on the local queue definition
 - Advertises the queue to the cluster
 - Available to other queue managers in the cluster
 - No need for remote queue definition

Example: Define the local queue QL.A to the cluster CLUS1

```
DEFINE QLOCAL(QL.A) CLUSTER(CLUS1)
```

Figure 12-16. Define local queues to the cluster

A *cluster queue* is a queue that a cluster queue manager hosts and is accessible by queue managers in the cluster. The local queue is either preexisting or created on the local queue manager. The other queue managers can see this queue and use it to put messages to it without the use of remote queue definition. The cluster queue can be advertised in more than one cluster.

It is not necessary to define remote-queue definitions for cluster queues. Applications can put messages to any cluster queue in the cluster. They can receive responses to their messages by providing a reply-to queue and specifying its name when they PUT the message on the queue.

The **CLUSTER** option on the **DEFINE QLOCAL** command identifies the cluster.

Controlling clusters with MQSC commands

- **DISPLAY CLUSQMGR** Display cluster information
- **SUSPEND QMGR** Remove a queue manager from a cluster
- **RESUME QMGR** Reinstate a queue manager to a cluster
- **REFRESH CLUSTER**
 - Discard locally held data about a cluster
 - Objects are rebuilt according to REPOS parameter
 - Enforces a cold start of the queue manager
 - Used only in exceptional circumstances
- **RESET CLUSTER**
 - Sent by a repository queue manager
 - Forcibly removes a queue manager from a cluster
 - Used only in exceptional circumstances

[Introduction to queue manager clusters](#)

© Copyright IBM Corporation 2017

Figure 12-17. Controlling clusters with MQSC commands

The figure lists the MQSC commands that you can use to control clusters:

- **DISPLAY CLUSQMGR**

If you enter this command from a queue manager with a full repository, the information that is returned is for every queue manager in the cluster. If you enter this command from a queue manager that does not have a full repository, the information that is returned is for only the queue managers in which it has an interest.

- **SUSPEND QMGR** and **RESUME QMGR**

Use the **SUSPEND QMGR** command to remove a queue manager from a cluster temporarily. Then, use the **RESUME QMGR** command to reinstate into the cluster. The **SUSPEND** command does not completely stop messages from being sent to a queue manager. If the suspended queue manager is the only queue manager with an available copy of a queue, the messages are sent to the queue on the suspended queue manager.

- **REFRESH CLUSTER**

You can enter **REFRESH CLUSTER(*)** to refresh the queue manager in all of the clusters that it is a member of. If used with **REPOS(YES)**, this command forces the queue manager to restart its search for full repositories from the information in the local cluster-sender definitions. The

search occurs even if the cluster-sender channel connects the queue manager to several clusters.

- **RESET CLUSTER**

In an emergency where a queue manager is temporarily damaged, you might want to inform the rest of the cluster before the other queue managers try to send it messages. The **RESET CLUSTER** command can be used to remove the damaged queue manager. The **RESET CLUSTER** command can also be used to remove a queue manager that does not belong to a cluster, perhaps because of a security problem. If necessary, you can use the **REFRESH CLUSTER** command to reverse the effect of **RESET CLUSTER** and put the queue manager back into the cluster.



Important

Use the commands to suspend, refresh, and reset the cluster with caution.

Cluster administration considerations

- Maintaining a cluster queue manager
 - SUSPEND and RESUME a queue manager
- Refreshing a cluster queue manager when necessary
 - REFRESH CLUSTER command
- Recovering a queue manager
 - Restore the queue manager from a linear log
- Maintaining the cluster transmission queue
 - Avoid queue full or disk full conditions
 - Ensure PUT(enabled) and GET(enabled) always
- When a queue manager fails, undelivered messages are backed out to the cluster transmission queue on the sending queue manager
- Cluster channels online monitoring by using **DISPLAY CHSTATUS**

Introduction to queue manager clusters

© Copyright IBM Corporation 2017

Figure 12-18. Cluster administration considerations

At some point, you need to back up the queue manager data or apply software updates. If the queue manager hosts any queues, its activities must be suspended. When the maintenance is complete, the queue manager activities can be resumed.

The **SUSPEND** command does not completely stop messages from being sent to a queue manager. If only the suspended queue manager has an available copy of a queue, the messages are sent to the queue on the suspended queue manager.

The **REFRESH CLUSTER** command allows a queue manager to be restarted regarding its full repository content. MQ ensures that no data is lost from your queues.

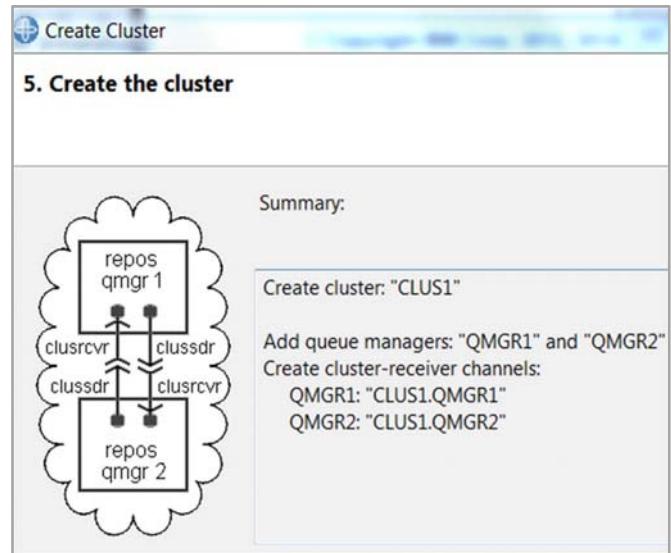
To restore from a point-in-time backup, enter the **REFRESH CLUSTER** command on the restored queue manager for all clusters in which the queue manager participates.

The availability and performance of the cluster transmission queue are essential to the performance of clusters. Make sure that it does not become full, and do not change this queue to get-disabled or put-disabled.

The **DISPLAY CHSTATUS** is a useful command when you troubleshoot connection problems in a cluster, such as identifying duplicate channels or mismatches in channel names.

Defining a cluster by using IBM MQ Explorer

- **Create Cluster** wizard
 1. Specify cluster name
 2. Identify two full repository queue managers
 3. Specify first full repository cluster-receiver channel name and connection details
 4. Specify second full repository cluster-receiver channel name and connection details
- Prerequisites:
 - Each full repository queue manager in the cluster must have a running listener
 - You must know the connection details of each full repository queue manager in the cluster
 - If the full repository queue managers belong to another cluster, you cannot use the **Create Cluster** wizard



Introduction to queue manager clusters

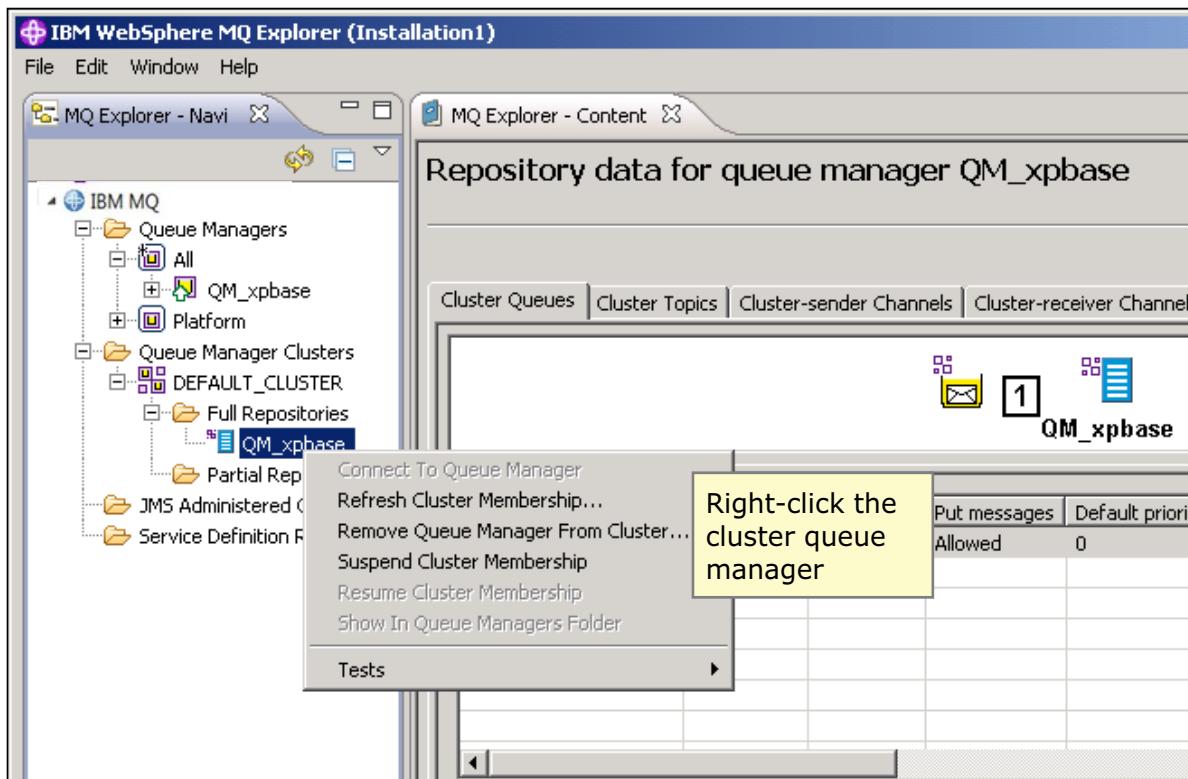
© Copyright IBM Corporation 2017

Figure 12-19. Defining a cluster by using IBM MQ Explorer

You can define a cluster by using the **Create Cluster** wizard in IBM MQ Explorer.

Before you use the Create Cluster wizard, you must create the queue managers and the queue manager must be running. You must also ensure that remote queue managers are accessible by IBM MQ Explorer.

Controlling clusters with IBM MQ Explorer



Introduction to queue manager clusters

© Copyright IBM Corporation 2017

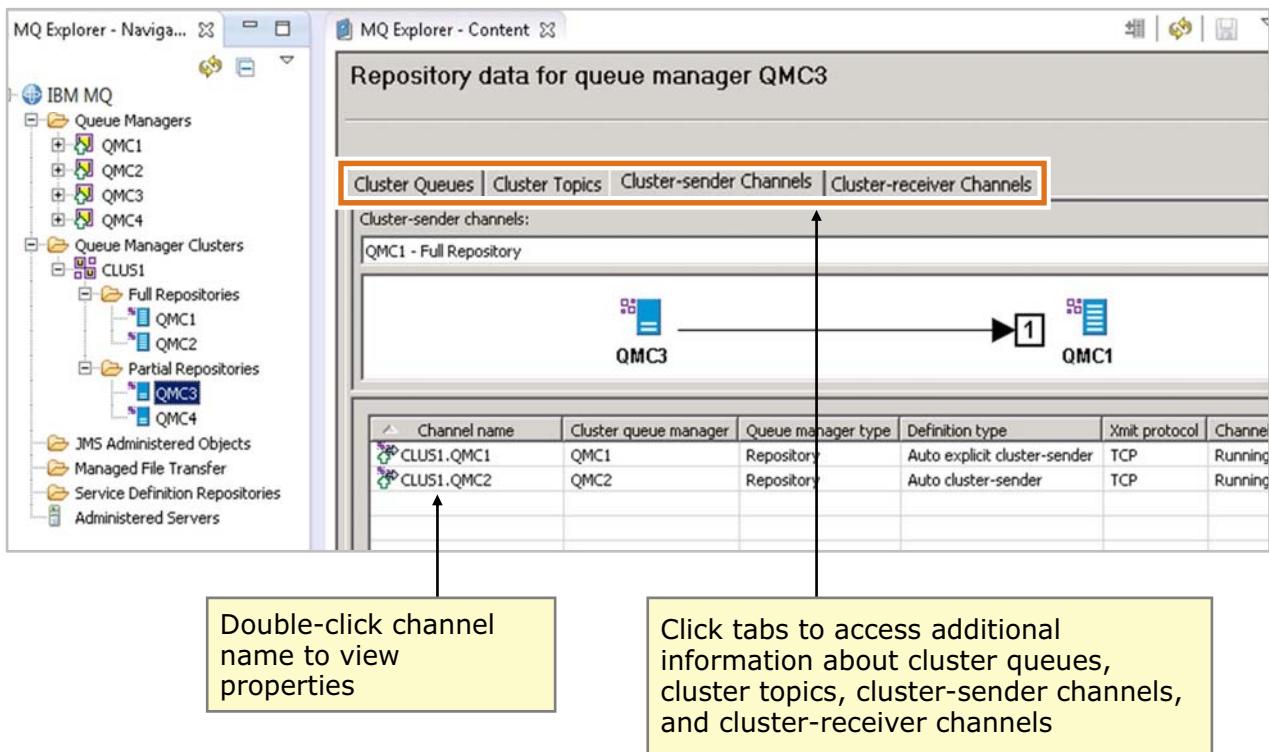
Figure 12-20. Controlling clusters with IBM MQ Explorer

You can use IBM MQ Explorer to refresh and suspend cluster membership, and remove a queue manager from a cluster. The queue managers that are in a cluster are listed in the **Queue Managers Clusters** folder in the **MQ Explorer - Navigator**.

Cluster management by IBM MQ Explorer assumes that any remote servers are accessible from IBM MQ Explorer.

IBM Training

Monitoring clusters with IBM MQ Explorer



Introduction to queue manager clusters

© Copyright IBM Corporation 2017

Figure 12-21. Monitoring clusters with IBM MQ Explorer

When a cluster queue manager is selected, tabs in the MQ Explorer **Content** view can be selected to access additional information about the cluster. Double-click an object, such as the channel name, to view its properties.

Cluster workload management options

- If cluster contains more than one instance of the same queue, IBM MQ uses the cluster workload management algorithm and cluster workload-specific attributes to determine target queue
 - Use-queue
 - Channel rank and queue rank
 - Channel and queue priority
 - Most recently used
 - Channel weighting
 - Application binding options

[Introduction to queue manager clusters](#)

© Copyright IBM Corporation 2017

Figure 12-22. Cluster workload management options

Cluster support allows more than one queue manager to host an occurrence of the same queue. So, two or more queue managers can be clones of each other, capable of running the same applications and having local definitions of the same queues. This technique can be used to implement queues for two reasons:

- To increase the capacity available to process messages
- To introduce an ability to fail over work from one server to another and improve availability of the service

This architecture can be used to spread the workload between queue managers, if applications allow it to do so.

With multiple choices, which are based on availability and channel priorities, a built-in workload management algorithm determines the remote queue manager. A local occurrence takes precedence by default. The figure lists the cluster workload options that can be specified on channel and queue definitions. Cluster workload management is taught in detail in course WM213, *IBM MQ V9 Advanced System Administration*.

Use-queue basics

- Allows remote queues to be chosen when a local queue exists
- Messages that a cluster channel receives must be put to a local queue if one exists

DEFINE QL(CLUS.Q1) CLUSTER(CLUS1) CLWLUSEQ()

- **LOCAL** = If a local queue exists, choose it
- **ANY** = Choose either local or remote queues
- **QMGR** = Use the use-queue value from the queue manager

ALTER QMGR CLWLUSEQ()

- **LOCAL** = If the queue specifies **CLWLUSEQ(QMGR)** and a local queue exists, choose it
- **ANY** = If the queue specifies **CLWLUSEQ(QMGR)**, choose either local or remote queues

Introduction to queue manager clusters

© Copyright IBM Corporation 2017

Figure 12-23. Use-queue basics

The use-queue attribute, **CLWLUSEQ**, specifies the behavior of an MQPUT operation when the target queue has a local instance and at least one remote cluster instance in most cases. The exception is in cases where the MQPUT originates from a cluster channel.

The valid options for the use-queue attribute on a queue are **LOCAL**, **ANY**, and **QMGR**.

- If you specify **QMGR** for the attribute value on the queue, the **CLWLUSEQ** parameter of the queue manager definition determines the behavior. This parameter is valid only for local queues.
- If you specify **ANY**, the queue manager treats the local queue as another instance of the cluster queue for the purposes of workload distribution.
- If you specify **LOCAL**, the local queue is the only target of the MQPUT operation.

The valid options for the use-queue attribute on a queue manager are **LOCAL** and **ANY**

Application binding options

- When multiple queues with the same name are advertised on a queue manager cluster, applications can choose one of the following options on MQOPEN call:
 - Send all messages from this application to a single instance (MQOO_BIND_ON_OPEN)
 - Allow the workload management algorithm to select the most suitable destination on a per message basis (MQOO_BIND_NOT_FIXED)
 - Allow an application to request that a 'group' of messages be all allocated to the same destination instance (MQOO_BIND_ON_GROUP)
- When application specifies MQOO_BIND_AS_Q_DEF, queue DEFBIND attribute determines binding

Introduction to queue manager clusters

© Copyright IBM Corporation 2017

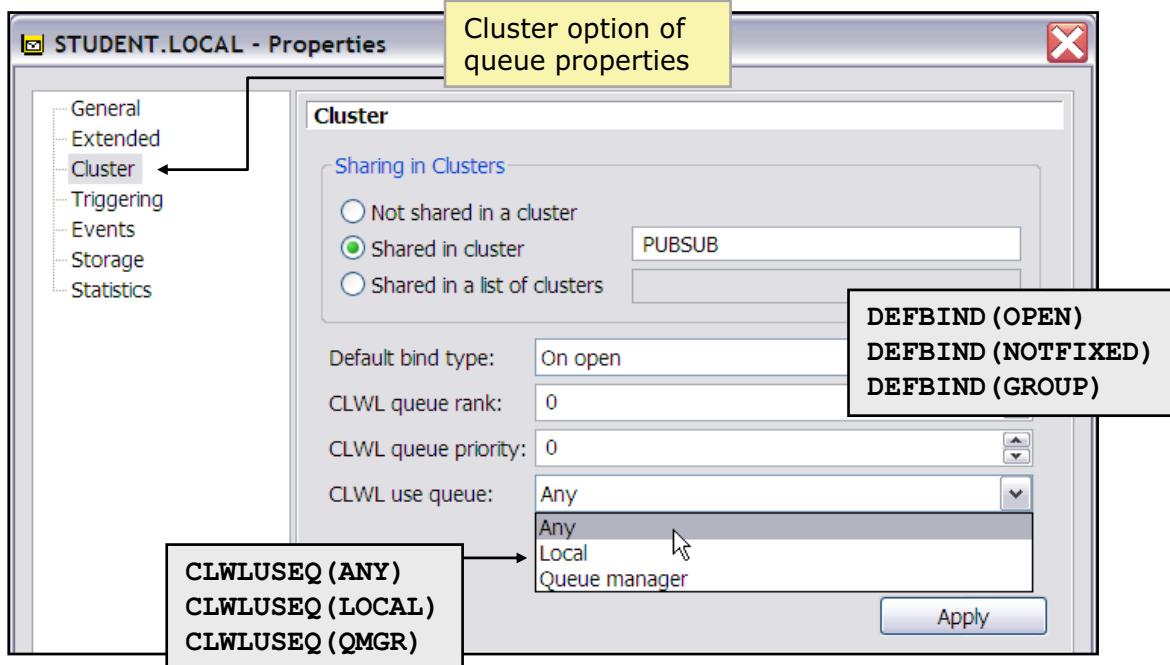
Figure 12-24. Application binding options

An option on the MQOPEN call allows the application to specify that the target queue manager needs to be fixed when there are multiple instances of the same queue within a cluster. That is, all messages that are put to the queue that specifies the object handle that is returned from the MQOPEN call must be directed to the same queue manager, by using the same route.

If it is not necessary for the application to force all the messages to be written to the same destination, specify BIND_NOT_FIXED on the MQOPEN call. This configuration selects a destination at MQPUT time, that is, on a message-by-message basis.

If the application does not specify BIND_ON_OPEN, BIND_NOT_FIXED, or BIND_ON_GROUP, the default option is BIND_AS_Q_DEF. Using BIND_AS_Q_DEF takes the binding that is used for the queue handle from the DEFBIND queue attribute.

Using IBM MQ Explorer to specify cluster queue properties



Introduction to queue manager clusters

© Copyright IBM Corporation 2017

Figure 12-25. Using IBM Explorer to specify cluster queue properties

The figure shows how to configure the cluster default bind and workload use-queue attribute for a queue in the queue **Cluster** properties.

Verifying and testing the cluster

- Use **DISPLAY CLUSTER** and **DISPLAY CLUSQMGR** command to verify the cluster

Example results when command entered on QM2:

```
DISPLAY CLUSQMGR(*)
AMQ8441: Display Cluster Queue Manager details
    CLUSQMGR(QM2)          CLUSTER(CLUS1)
    CHANNEL(CLUS1.QM2)

AMQ8441: Display Cluster Queue Manager details
    CLUSQMGR(QM1)          CLUSTER(CLUS1)
    CHANNEL(CLUS1.QM1)
```

- Use IBM MQ sample programs to put and get messages to cluster queues

Figure 12-26. Verifying and testing the cluster

You can verify the cluster configuration by entering the **DISPLAY CLUSQMGR(*)** command to list the cluster queue manager details.

Use the IBM MQ sample programs, such as **amqspput** and **amqsgget**, to test the cluster.

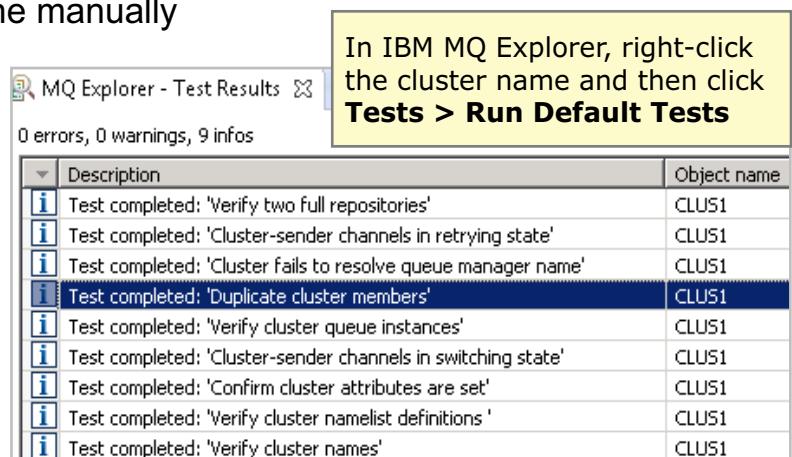
Cluster tests in IBM MQ Explorer

- Verifies two full repositories in the cluster
- Checks whether any of manually defined cluster-sender channels are still in a *Retrying* state
- Checks that cluster can successfully resolve all queue manager names
- Checks whether any cluster memberships list the same queue manager more than one time
- Verifies that all instances of a cluster queue have the same attributes
- Checks whether any of the manually defined cluster-sender channels are still in a *Switching* state
- Confirms that cluster channels have a cluster value
- Checks the cluster name attributes of queues, channels, and queue managers

Introduction to queue manager clusters

© Copyright IBM Corporation 2017

Figure 12-27. Cluster tests in IBM MQ Explorer



In IBM MQ Explorer, right-click the cluster name and then click **Tests > Run Default Tests**

Description	Object name
Test completed: 'Verify two full repositories'	CLUS1
Test completed: 'Cluster-sender channels in retrying state'	CLUS1
Test completed: 'Cluster fails to resolve queue manager name'	CLUS1
Test completed: 'Duplicate cluster members'	CLUS1
Test completed: 'Verify cluster queue instances'	CLUS1
Test completed: 'Cluster-sender channels in switching state'	CLUS1
Test completed: 'Confirm cluster attributes are set'	CLUS1
Test completed: 'Verify cluster namelist definitions '	CLUS1
Test completed: 'Verify cluster names'	CLUS1

IBM MQ Explorer includes built-in tests to verify some of the cluster properties and configuration.

- **Verify two full repositories** checks that the cluster contains at least two full repository queue managers.
- **Cluster-sender channels in retrying state** checks whether any of manually defined cluster-sender channels are still in a *Retrying* state.
- **Cluster fails to resolve queue manager name** checks that the cluster can successfully resolve all queue manager names.
- **Duplicate cluster members** checks whether any cluster memberships list the same queue manager more than one time.
- **Verify cluster queue instances** verifies that all instances of a cluster queue have the same attributes.
- **Cluster-sender channels in switching state** checks whether any of the manually defined cluster-sender channels are still in a *Switching* state.
- **Confirm that cluster attributes are set** checks that all cluster channels have a cluster value.
- **Verify cluster names** checks the cluster name attributes of queues, channels, and queue managers, except for capitalization.

Implementation

- Be clear about requirements
 - Reduced system administration?
 - Workload balancing?
- Familiarize yourself with the queue manager clusters commands
- Use consistent naming conventions
- Document processes for system changes
 - Add and remove queue manager to, from, or to and from cluster
 - Take queue manager offline for maintenance
- Have two full repositories and consider where they are hosted
- Monitor health of the system queues and channels

Introduction to queue manager clusters

© Copyright IBM Corporation 2017

Figure 12-28. Implementation

The figure lists some guidelines to use when you implement clusters.

Cluster troubleshooting

- For channels, verify:
 - All cluster channels are paired
 - Every queue manager in the cluster has a *RUNNING* cluster-receiver channel
 - Channel state by using: **DISPLAY CHSTATUS (*)**
 - Every full repository has a *RUNNING* cluster-sender channel to every other full repository
- For cluster queue managers, verify:
 - All queue managers in the cluster are aware of all the full repositories by using: **DISPLAY CLUSQMGR (*) QMTYPE**
 - Intended full repository queue managers are defined as full repositories and are in the cluster by using: **DISPLAY QMGR REPOS or REPOSNL**
- Queues:
 - Verify that messages are not building up on transmission queues
 - Verify that messages are not building up on system queues

Introduction to queue manager clusters

© Copyright IBM Corporation 2017

Figure 12-29. Cluster troubleshooting

Use this list and the troubleshooting advice in the IBM Knowledge Center to help you to detect and resolve problems when you use queue manager clusters.

Administering the SYSTEM.CLUSTER queues

- SYSTEM.CLUSTER.REPOSITORY.QUEUE
 - Holds persistent view of repository cache
 - CURDEPTH should be greater than zero
- SYSTEM.CLUSTER.COMMAND.QUEUE
 - Holds inbound administrative messages
 - IPPROCS should always be 1
 - CURDEPTH should be zero or decrementing
- SYSTEM.CLUSTER.TRANSMIT.QUEUE
 - Holds outbound administrative and user messages
 - Correlation ID in transmission queue header contains the name of the cluster-sender channel
- SYSTEM.CLUSTER.HISTORY.QUEUE
 - Store the history of cluster state information for service purposes
 - To suppress history collection, change this queue to PUT(DISABLED)

Introduction to queue manager clusters

© Copyright IBM Corporation 2017

Figure 12-30. Administering the SYSTEM.CLUSTER queues

MQ contains special SYSTEM queues for clustering. These queues are created automatically when a queue manager is created.

The SYSTEM.CLUSTER.REPOSITORY.QUEUE holds a persistent copy of the cluster repository.

The SYSTEM.CLUSTER.COMMAND.QUEUE is used to communicate repository changes between queue managers. Messages that are written to this queue contain updates to the repository data that applies to the local copy of the repository, or requests for repository data.

The SYSTEM.CLUSTER.TRANSMIT.QUEUE is the transmission queue for all destinations in the cluster.

The SYSTEM.CLUSTER.HISTORY.QUEUE stores the history of cluster state information for service purposes. In the default object settings, SYSTEM.CLUSTER.HISTORY.QUEUE is set to PUT(ENABLED). To suppress history collection, change the setting to PUT(DISABLED).

All of these queues typically contain many messages.

Unit summary

- Describe a cluster and list the components that are involved
- Describe the difference between a full and a partial repository queue manager
- Configure a basic cluster
- Describe the administration tasks that must be considered in a clustered environment

Introduction to queue manager clusters

© Copyright IBM Corporation 2017

Figure 12-31. Unit summary

Review questions

1. True or False: The SYSTEM.CLUSTER.COMMAND.QUEUE holds inbound and outbound administrative messages.
2. True or False: Remote queue definitions are not required when using clusters.
3. True or False: A cluster workload algorithm uses workload balancing attributes and rules to select the final destination for messages that are put onto cluster queues.



Introduction to queue manager clusters

© Copyright IBM Corporation 2017

Figure 12-32. Review questions

Write your answers down here:

- 1.
- 2.
- 3.

Review answers

1. True or False: The SYSTEM.CLUSTER.COMMAND.QUEUE holds inbound and outbound administrative messages.
The answer is False. The SYSTEM.CLUSTER.COMMAND.QUEUE holds inbound administrative messages only.

2. True or False: Remote queue definitions are not required when using clusters.
The answer is True.

3. True or False: A cluster workload algorithm uses workload balancing attributes and rules to select the final destination for messages that are put onto cluster queues.
The answer is True.



Exercise: Implementing a basic cluster

Introduction to queue manager clusters

© Copyright IBM Corporation 2017

Figure 12-34. Exercise: Implementing a basic cluster

In this exercise, you use IBM MQ Explorer to create a cluster of four queue managers. You then test the cluster by using the cluster mechanism to send messages between queues on all queue managers in the cluster.

Exercise objectives

- Use IBM MQ Explorer to define a simple queue manager cluster
- Use the IBM MQ sample programs to test a cluster environment



Introduction to queue manager clusters

© Copyright IBM Corporation 2017

Figure 12-35. Exercise objectives

See the *Course Exercises Guide* for detailed instructions.

Unit 13. Monitoring and configuring IBM MQ for performance

Estimated time

01:30

Overview

In this unit, you learn about the information that the accounting and statistics system management utilities provide for monitoring an IBM MQ network.

How you will check your progress

- Review questions
- Hands-on exercise

References

IBM Knowledge Center for IBM MQ V9

Unit objectives

- Describe the statistics and accounting data that IBM MQ provides
- View and generate accounting and statistical data
- Subscribe to IBM MQ statistic topics
- Interpret statistics and accounting data to identify possible system performance benefits
- Configure and tune IBM MQ for improved performance

Collecting statistics and accounting data

- IBM MQ collects sets of data to be written as messages to predefined queues
 - Data can be post-processed to give information about system activity
 - Data can be used for capacity planning, chargeback, and other information
- Statistical data collection is divided into three categories:
 - MQI statistics
 - Queue statistics
 - Channel statistics
- Collection of data for each class can be selected independently
- Queue manager and queue or channel attributes control collection
- Accounting data collects information about MQI applications
- Can be activated by modifying MQ object properties or by subscribing to IBM MQ statistics topics

Figure 13-2. Collecting statistics and accounting data

MQ can generate statistics messages that record the information about the activities that occur in an MQ system.

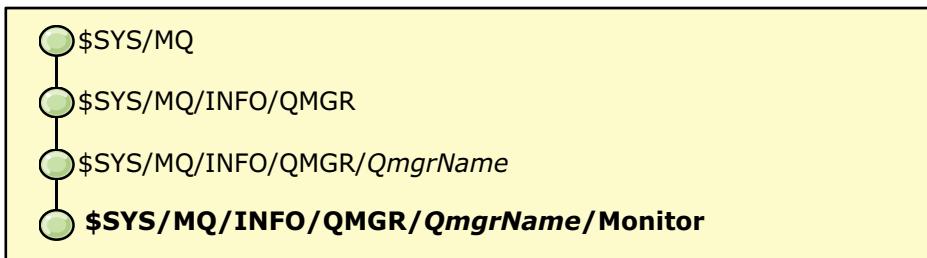
The information that is contained within statistics messages can be used for:

- Accounting for application resource use
- Recording application activity
- Planning for capacity
- Detecting problems in your queue manager network
- Helping to determine the causes of problems in your queue manager network
- Improving the efficiency of your queue manager network
- Familiarizing yourself with the running of your queue manager network
- Confirming that your queue manager network is running correctly

Statistical data is available for MQI, queues, and channels.

Subscribing to statistics data (1 of 2)

- IBM MQ publishes resource monitoring data from the queue manager to \$SYS/MQ topic branch
 - Access to \$SYS/MQ is restricted to administrators by default
 - Others can subscribe to a subset of the data
- Not controlled by queue manager configuration options
 - Statistics topic data is generated only when an application subscribes to the system topic
- Statistics are organized according to class and type: CPU, DISK, STATMQI, STATQ
- Includes all statistics in SYSTEM.ADMIN.STATISTICS.QUEUE plus statistics for monitoring CPU and disk



[Monitoring and configuring IBM MQ for performance](#)

© Copyright IBM Corporation 2017

Figure 13-3. Subscribing to statistics data (1 of 2)

IBM MQ Version 9.0 publishes statistics and activity trace information messages to MQ system level topic strings.

In MQ, statistics are published to a system topic under \$SYS/MQ/INFO/QMGR. An authorized user can subscribe to the topics to receive monitoring information for the queue manager.

You must be authorized at, or deeper than, \$SYS/MQ to be granted authority to use the \$SYS/MQ topic tree.

You can use MQSC and IBM MQ Explorer to define and monitor topics and subscriptions.

Subscribing to statistics data (2 of 2)

- Statistics are published approximately every 10 seconds
- STATINT queue manager attribute controls the interval over which the subscription high and low watermarks are taken
- IBM MQ sample program `amqsrua` reports metadata from queue manager topic
 - Source file is in the IBM MQ `samples` directory
- User applications discover which statistics are available by subscribing to a system topic
- Topic tree structure:
`$SYS/MQ/INFO/QMGR/<QmgrName>/Monitor/<Class>/<Type>`
Example: `$SYS/MQ/INFO/QMGR/QMGR1/Monitor/CPU/QMgrSummary`

Benefits of subscribing to statistics topics

- Dynamically enable and disable statistics without modifying queue manager configuration
- Supports multiple subscribers to the same set of information, allowing more than one monitoring tool to access statistics
- Ability to give non-administrative users permission to subscribe to a subset of information specific to their application resources

Monitoring and configuring IBM MQ for performance

© Copyright IBM Corporation 2017

Figure 13-5. Benefits of subscribing to statistics topics

IBM MQ statistics that are published to topics only

- CPU statistics
 - User CPU time
 - System CPU time
 - CPU load
 - RAM free percentage CPU
 - RAM total bytes
- STATQ PUT statistics
 - Lock contention
 - Queue avoided puts
 - Queue avoided bytes
- DISK statistics
 - Log bytes maximum
 - Log physical bytes written
 - Log logical bytes written
 - Log write latency

[Monitoring and configuring IBM MQ for performance](#)

© Copyright IBM Corporation 2017

Figure 13-6. IBM MQ statistics that are published to topics only

This figure summarizes the MQ statistics that are available only in published topics.

- **User CPU time** is the average (taken over the last 10-second interval) percentage of time that is used by the CPU when it was in non-privileged code.
- **System CPU time** is the average (taken over the last 10-second interval) percentage of time that is used by the CPU when it was in privileged code.
- **CPU load** is the load average.
- **RAM total bytes** is an approximation of the memory that is used by the queue manager.
- **Log bytes maximum** refers to the maximum number of bytes that can be written to the log when all the primary and secondary extents are full. This value is less than the size of the log file system.
- **Log physical bytes written** and **Log logical bytes written** are the physical number of bytes written to disk.
- **Log write latency** is a rolling average that represents the time that a single write to disk takes.
- **Lock contention** is the percentage of attempts to lock the queue that resulted in waiting for another process to release the lock first. Decreasing lock contention is likely to increase the

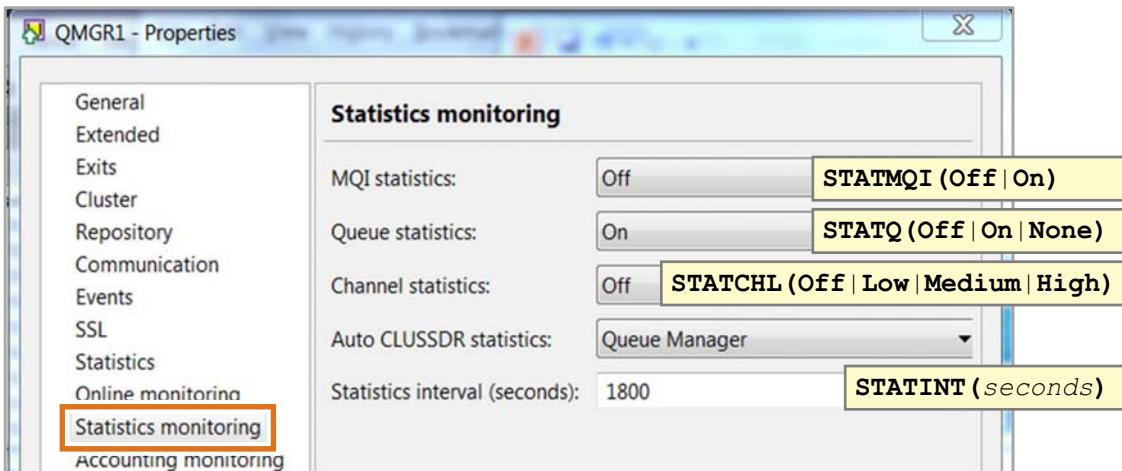
maximum throughput of your system because taking a lock that is not currently locked is a much cheaper operation than waiting for a lock to be released.

- If a message is put to a queue when a “getter” is waiting, the message might not need to be queued as it might be possible for it to be passed to the getter immediately. It is said that this message avoided the queue. **Queue avoided puts** and **Queue avoided bytes** are the count of such messages and bytes. Increasing queue avoidance is likely to increase the maximum throughput of your system because it avoids the cost of putting the message onto the queue and getting it off again.

For more information, see the “Statistics published to the system topic in MQ v9” MQdev Blog on IBM DeveloperWorks at:

https://www.ibm.com/developerworks/community/blogs/messaging/entry/Statistics_published_to_the_system_topic_in_MQ_v9?lang=en

Queue manager statistics monitoring properties



- Edit properties on this page to configure how statistics monitoring data is collected for queues and channels on this queue manager

[Monitoring and configuring IBM MQ for performance](#)

© Copyright IBM Corporation 2017

Figure 13-7. Queue manager statistics monitoring properties

You can enable statistics collection for MQI, queues, and channels on the queue manager by using the MQSC commands or the queue manager **Statistics monitoring** properties in IBM MQ Explorer. The figure shows the IBM MQ Explorer statistics property and associated MQSC commands.

The queue manager MQI statistics property (**STATMQI**) controls the collection of MQI statistics. When the MQI statistics property is set to **On**, the MQI statistics information is collected for every connection to the queue manager.

Queue statistics can be enabled for individual queues, or for multiple queues at the queue manager level, by using the queue manager **Queue statistics** property (**STATQ**). When **Queue statistics** is set to **On**, queue statistics are collected for queues that have the **Queue statistics** property set to **Queue Manager**.

The queue manager **Channel statistics** property (**STATCHL**) controls the collection of channel statistics for all channels, or individual channels.

The **Statistics interval** property (**STATINT**) specifies the interval, in seconds, between the generation of statistics messages. The default statistics interval is 1800 seconds (30 minutes).

MQI statistics

- Generated only for queues that are opened after statistics collection is enabled
- Contain information about the number of MQI calls made during a configured interval
 - Success and failure counts for each MQI verb
 - MQI and byte counts for PUT and GET on queues for persistent and nonpersistent messages
- Written in the form of PCF records to SYSTEM.ADMIN.STATISTICS.QUEUE and IBM MQ topics

- Methods for enabling MQI statistics:
 - Use the MQSC command **ALTER QMGR** with **STATMQI(ON)**
 - Subscribe to **STATMQI** statistics topic
 - Use IBM MQ Explorer to modify queue manager **Statistics Monitoring** and queue **Statistics** properties

[Monitoring and configuring IBM MQ for performance](#)

© Copyright IBM Corporation 2017

Figure 13-8. MQI statistics

MQI statistics must be enabled on a queue before it is opened.

MQI statistics messages contain information that relates to the number of MQI requests processed during a configured interval. For example, the statistics message information can include the number of MQI commands that a queue manager processes.

A statistics message is a PCF message that contains PCF structures. Statistics messages are delivered to the system queue (SYSTEM.ADMIN.STATISTICS.QUEUE) at configured intervals.

You can use MQSC and IBM MQ Explorer to control statistics generation. In IBM MQ V9, you can also subscribe to MQ topics to activate statistics collection.

Queue statistics

- Can be configured at both queue manager and queue level
- Minimum and maximum depth of queue
- Average time-on-queue for messages that are retrieved from the queue
- API and byte counts for GET, PUT, BROWSE (persistent and non-persistent)
- Methods for enabling queue statistics:
 - Alter the queue definition to use the **STATQ(ON)** attribute
 - Subscribe to **STATQ** statistics topic
 - Use IBM MQ Explorer to modify queue manager **Statistics Monitoring** and queue **Statistics** properties

Figure 13-9. Queue statistics

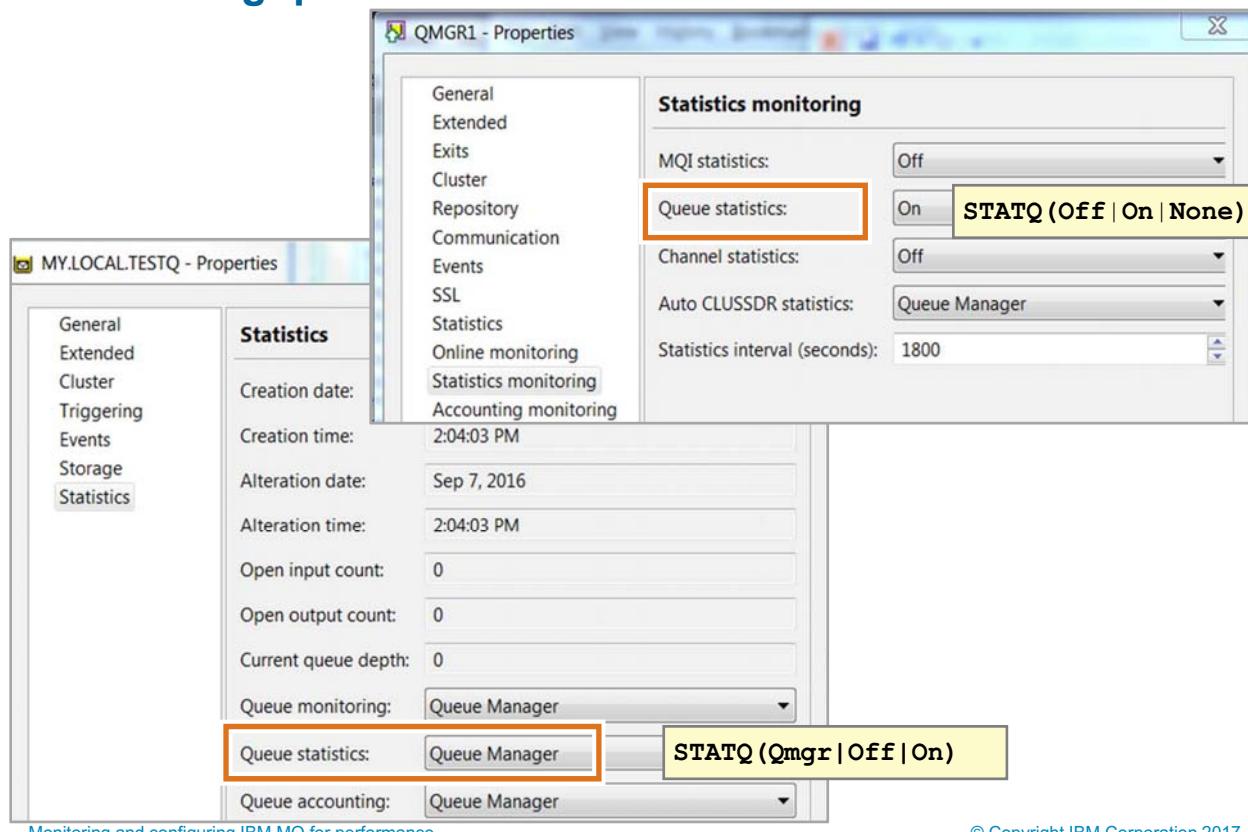
Queue statistics messages contain information about the activity of a queue during a configured interval. The information includes the number of messages that are put on and retrieved from the queue, and the total number of bytes that a queue processes.

Each queue statistics message can contain up to 100 records. Each record contains activity information for each queue for which statistics were collected.

You can enable queue statistics by using MQSC, subscribing to the MQ STATQ topic, or by using IBM MQ Explorer.

IBM Training

Controlling queue statistics



Monitoring and configuring IBM MQ for performance

© Copyright IBM Corporation 2017

Figure 13-10. Controlling queue statistics

By using MQ Explorer or MQSC commands, you can enable queue statistics at the queue manager level or at the queue.

If the **Queue statistics** property is set to **Queue Manager** on the queue, the queue manager controls statistics collection. If the **Queue statistics** properties are set to **On** on the queue, statistics information is collected for every connection to the queue manager that opens the queue.

When you use MQSC, the queue attribute **STATQ** on the **ALTER** command for the queue controls individual channels. The queue manager attribute **STATQ** on the **ALTER OMGR** command controls many queues together.

Channel statistics

- Messages are written at end of the interval that contains up to 100 channel records
- Only channels “active” in the time interval have statistics that are recorded
- Can be configured at both queue manager and channel levels by using MQSC or IBM MQ Explorer
 - Set **STATACLS** queue manager attribute to control automatically defined cluster-sender channels

[Monitoring and configuring IBM MQ for performance](#)

© Copyright IBM Corporation 2017

Figure 13-11. Channel statistics

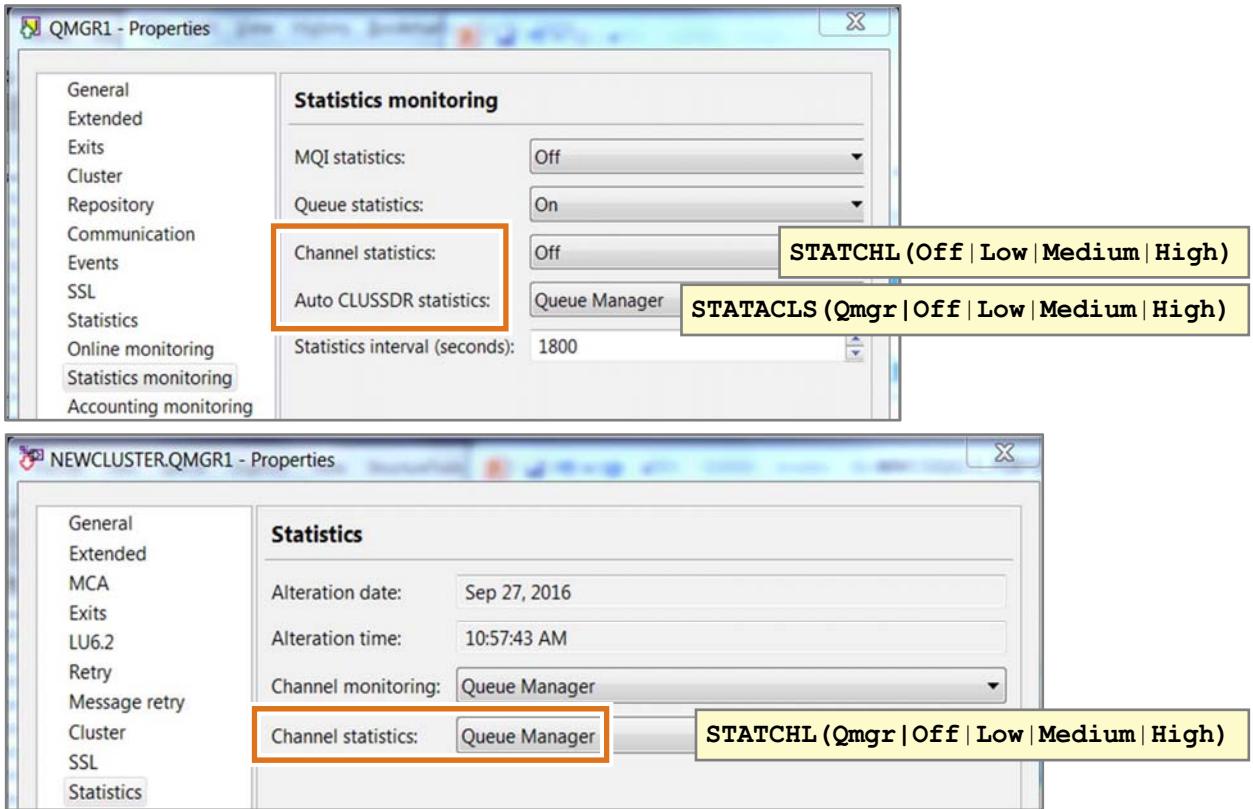
Channel statistics messages contain information about the activity of a channel during a configured interval. For example, the message can include the number of messages that the channel transferred, or the number of bytes that the channel transferred.

Each channel statistics message contains up to 100 records. Each record contains the activity for each channel for which statistics were collected.

Channel statistics information collection can be set to one of the three monitoring levels: low, medium, or high. Collecting statistics information data might require that the process runs some instructions that can affect performance. To reduce the impact of channel statistics collection, the “medium” and “low” monitoring options measure a sample of the data at regular intervals rather than collecting data all the time.

Automatically defined cluster-sender channels are not MQ objects, so they do not have attributes. To control automatically defined cluster-sender channels, use the queue manager attribute **STATACLS**. This attribute determines whether automatically defined cluster-sender channels within a queue manager are enabled or disabled for channel statistics information collection.

Controlling channel statistics



Monitoring and configuring IBM MQ for performance

© Copyright IBM Corporation 2017

Figure 13-12. Controlling channel statistics

By using MQ Explorer or MQSC commands, you can enable queue and channel statistics at the queue manager level or at the individual queue or channel level.

If the **Channel statistics** property on the channel is set to **Queue Manager**, the queue manager controls statistics collection. If the **Channel statistics** property on the channel is set to **Off**, no statistics information is collected for this channel.

When you use MQSC, the channel attribute **STATCHL** on the **ALTER CHL** command controls individual channels. The queue manager attribute **STATCHL** on the **ALTER QMGR** command controls many channels together.

IBM MQ accounting data collection

- Can collect connection-level and queue information for each connection
 - MQI accounting data that is controlled by queue manager
 - Queue accounting that is controlled by queue manager and queue
 - Accounting interval in seconds
 - Applications can override the MQI accounting attribute and the Queue accounting attribute by using the Connect options in MQCONN calls
- Connection-level information includes:
 - Context details such as application name, process ID, connection type, and connect time
 - MQI counts
 - Message details (counts, bytes) for MQPUT, MQGET, MQGET (browse)
- Written in the form of a PCF monitoring message to SYSTEM.ADMIN.ACOUNTING.QUEUE

[Monitoring and configuring IBM MQ for performance](#)

© Copyright IBM Corporation 2017

Figure 13-13. IBM MQ accounting data collection

Accounting messages record information about the MQI operations that MQ applications complete. An accounting message is a PCF message that contains a number of PCF structures.

When an application disconnects from a queue manager, an accounting message is generated and delivered to the system accounting queue SYSTEM.ADMIN.ACOUNTING.QUEUE. For long-running MQ applications, intermediate accounting messages are generated as follows:

- When the time since the connection was established exceeds the configured interval
- When the time since the last intermediate accounting message exceeds the configured interval

The information that is contained within accounting messages can be used for the following purposes:

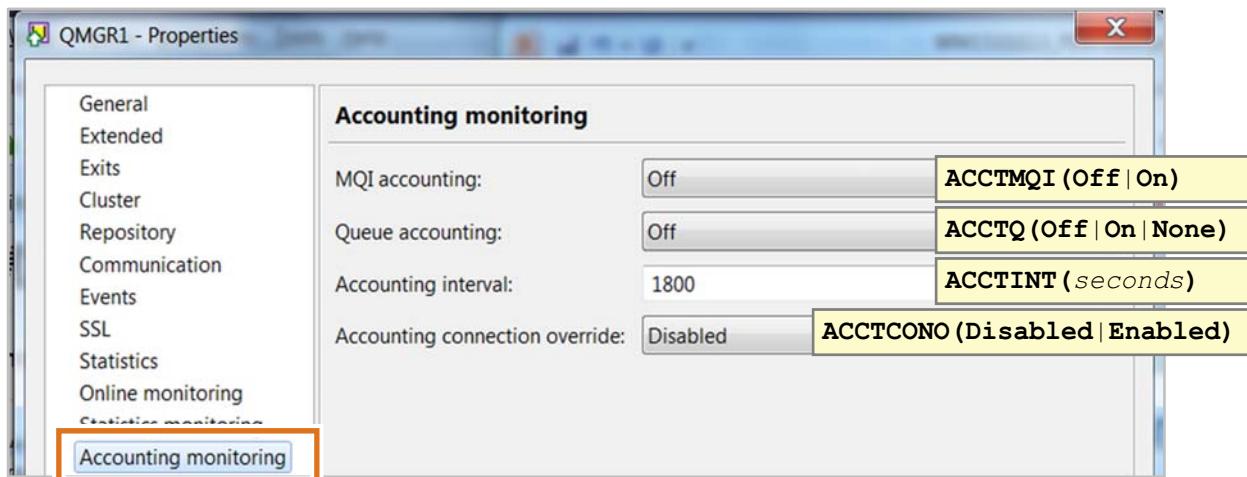
- Accounting for application resource use
- Recording application activity
- Detecting problems in your queue manager network
- Helping to determine the causes of problems in your queue manager network
- Improving the efficiency of your queue manager network
- Familiarizing yourself with the running of your queue manager network
- Confirming that your queue manager network is running correctly

MQI accounting messages contain information that is related to the number of MQI requests that were run by using a connection to a queue manager.

The application can also modify the collection of both MQI and queue statistics at the connection level by specifying the `ConnectOpts` parameter on the MQCONN call.



Controlling queue manager accounting



- Can set properties for both local and remote queue managers

Figure 13-14. Controlling queue manager accounting

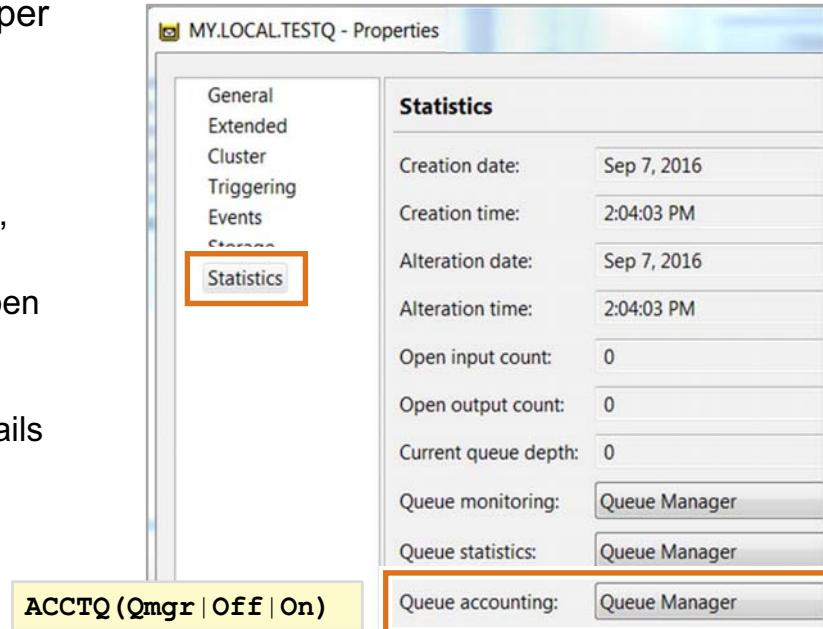
You can enable MQI and queue accounting on the queue manager by using MQSC commands or the queue manager **Accounting monitoring** properties in MQ Explorer. The figure shows the MQ Explorer **Accounting monitoring** properties and the associated MQSC command.

The queue manager **MQI accounting** property (**ACCTMQI**) controls the collection of MQI accounting information. When **MQI accounting** is set to **On**, accounting information is collected for every connection to the queue manager.

The queue manager **Queue accounting** property (**ACCTQ**) controls the collection of queue accounting information for any queues with the **Queue accounting** property set to **Queue Manager**.

Controlling queue accounting

- Queue data might be up to 100 queue details per message
- Queue information includes:
 - Queue details: Name, type
 - Open details: First open time, last close time
 - MQPUT, MQGET, MQGET(browse) details
- Control at queue manager level or at queue level



Monitoring and configuring IBM MQ for performance

© Copyright IBM Corporation 2017

Figure 13-15. Controlling queue accounting

Queue accounting messages contain information about the number of MQI requests that were processed by using connections to a queue manager, concerning specific queues.

Each queue accounting message can contain up to 100 records. Every record contains information about application activity for a specific queue.

Displaying statistics and accounting data

- Use **amqsmon** sample program to display the information that is contained within accounting and statistics messages in a formatted form
 - Accounting messages are read from the accounting queue, SYSTEM.ADMIN.ACCTOUNTING.QUEUE
 - Statistics messages are read from the statistics queue, SYSTEM.ADMIN.STATISTICS.QUEUE
- Source code is provided
- Examples:

```
amqsmon -m QMGR1 -t statistics
amqsmon -m QMGR1 -t accounting
```

Monitoring and configuring IBM MQ for performance

© Copyright IBM Corporation 2017

Figure 13-16. Displaying statistics and accounting data

Accounting and statistics messages are written to the system accounting and statistics queues. To use the information that is recorded in these messages, you must use an application to transform the recorded information into a format that is suitable for use.

The MQ sample program, **amqsmon**, processes messages from the accounting and statistics queues and shows the information in a readable format.

The supplied source code for the **amqsmon** program can be used as a template for writing your own application to process accounting or statistics messages. You can also modify the **amqsmon** source code to meet your own particular requirements.

The required parameter is the type (**-t**) of messages to process:

- If **-t** is set to **accounting**, accounting record messages are processed from the system queue SYSTEM.ADMIN.ACCTOUNTING.QUEUE
- If **-t** is set to **statistics**, statistics records are processed. Messages are read from the system queue SYSTEM.ADMIN.STATISTICS.QUEUE

For a complete description the **amqsmon** parameters, see the IBM MQ product documentation.

Examples of accounting data

```
QueueManager: QMGR1
IntervalEndDate: 2016-08-10
IntervalEndTime: 14:39:50
CommandLevel: 900
SeqNumber: 0
ApplName: amqsput.exe
ApplicationPid: 9408
ApplicationTid: 1
UserId: 'admin01'
ObjectCount: 1
```

```
OBJECTS:
QueueName: 'APP.QUEUE.X'
QueueType: Predefined
QueueDefType: Local
OpenCount: 1
OpenDate: 2016-08-10
OpenTime: 14:39:49
CloseCount: 1
CloseDate: 2016-08-10
CloseTime: 14:39:50
```

```
PutCount: [0, 1]*
PutFailCount: 0
Put1Count: [0, 0]
Put1FailCount: 0
PutBytes: [0, 4]
PutMinBytes: [0, 4]
PutMaxBytes: [0, 4]
*array shows nonpersistent, persistent
```

Figure 13-17. Examples of accounting output

The figure includes three examples of accounting information that the `amqsmon` sample program generates.

The first example shows information about the queue manager. The second example shows information about the objects that the queue manager controls, such as a queue. The third example shows more information about the queue.

Example of statistics data

```
MonitoringType: MQIStatistics
QueueManager: 'QM01'
IntervalStartDate: '2016-10-27'
IntervalStartTime: '11.41.38'
IntervalEndDate: '2016-10-27'
IntervalEndTime: '11.44.38'
CommandLevel: 900
ConnCount: 0
ConnFailCount: 0
ConnHighwater: 23
DiscCount: [0, 0, 0]
OpenCount: [0, 753, 0, 0, 0, 127, 0, 0, 0, 0, 0, 0, 0, 0, 0]
OpenFailCount: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
CloseCount: [0, 753, 0, 0, 0, 127, 0, 0, 0, 0, 0, 0, 0, 0, 0]
CloseFailCount: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
InqCount: [0, 829, 0, 0, 0, 204, 0, 0, 0, 0, 0, 0, 0, 0, 0]
InqFailCount: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
SetCount: [0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
SetFailCount: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
PutCount: [829, 2]
PutFailCount: 0
Put1Count: [0, 0]
Put1FailCount: 0
PutBytes: [671324, 316]
GetCount: [829, 1]
. . .
```

Monitoring and configuring IBM MQ for performance

© Copyright IBM Corporation 2017

Figure 13-18. Example of statistics data

Subscribing to statistics with the `amqsrua` program

Syntax for interactive mode: `amqsrua [-n MaxPubs] -m QmgrName`

- `MaxPubs` specifies number of reports to return before the command ends
- Enter the class (`-c`) of data to return
 - **CPU** returns information about CPU usage
 - **DISK** returns information about disk usage
 - **STATMQI** returns information about MQI usage
 - **STATQ** returns information about per-queue MQI usage
- Enter the type (`-t`) of data to return
 - For **CPU**: SystemSummary or QMgrSummary
 - For **DISK**: SystemSummary, QMgrSummary, or Log
 - For **STATMQI**: CONNDISC, OPENCLOSE, PUT, GET, SYNCPOINT, SUBSCRIBE, and PUBLISH
 - For **STATQ**: OPENCLOSE, INQSET, PUT, and GET

Figure 13-19. Subscribing to statistics with the `amqsrua` program

The `amqsrua` command reports metadata that is published by queue managers. This data can include information about the CPU, memory, and disk usage. You can also see data equivalent to the STATMQI PCF statistics data. The data is published every 10 seconds and is reported while the command runs.

The class of data that you select (CPU, DISK, STATMQI, or STATQ) determines the type of data available to return.

Example: Using the `amqsrua` sample program

```
$ amqsrua -n 2 -m QMA
CPU : Platform central processing units
DISK : Platform persistent data stores
STATMQI : API usage statistics
STATQ : API per-queue usage statistics

Enter Class selection
==> CPU
SystemSummary : CPU performance - platform wide
QMngrSummary : CPU performance - running queue manager

Enter Type selection
==> QMngrSummary
Publication received PutDate:20151014 PutTime:09175398
User CPU time - percentage estimate for queue manager 0.02%
System CPU time - percentage estimate for queue manager 0.04%
RAM total bytes - estimate for queue manager 200MB

Publication received PutDate:20151014 PutTime:09180405
User CPU time - percentage estimate for queue manager 0.00%
System CPU time - percentage estimate for queue manager 0.00%
RAM total bytes - estimate for queue manager 200MB
```

[Monitoring and configuring IBM MQ for performance](#)

© Copyright IBM Corporation 2017

Figure 13-20. Example: Using the `amqsrua` sample program

The example shows the result of using `amqsrua` to view CPU performance data for the running queue manager over a 20-second period.

Real-time monitoring

- Determine the current state of queues and channels within a queue manager
 - Help to understand the steady state of the IBM MQ system
 - Determine condition of a queue manager at any moment, even if no specific event or problem was detected
 - Assist with determining the cause of a problem in the IBM MQ system
- Configure by using MQSC commands or IBM MQ Explorer queue manager properties
 1. Enable real-time monitoring at the queue manager
 2. Enable or disable real-time monitoring for individual queues or channels, or for multiple queues or channels

Real-time monitoring might negatively affect performance

With real-time monitoring you can determine the current state of queues and channels within a queue manager. The information that is returned is accurate at the moment the command is sent.

Information can be returned for one or more queues or channels and can vary in quantity.

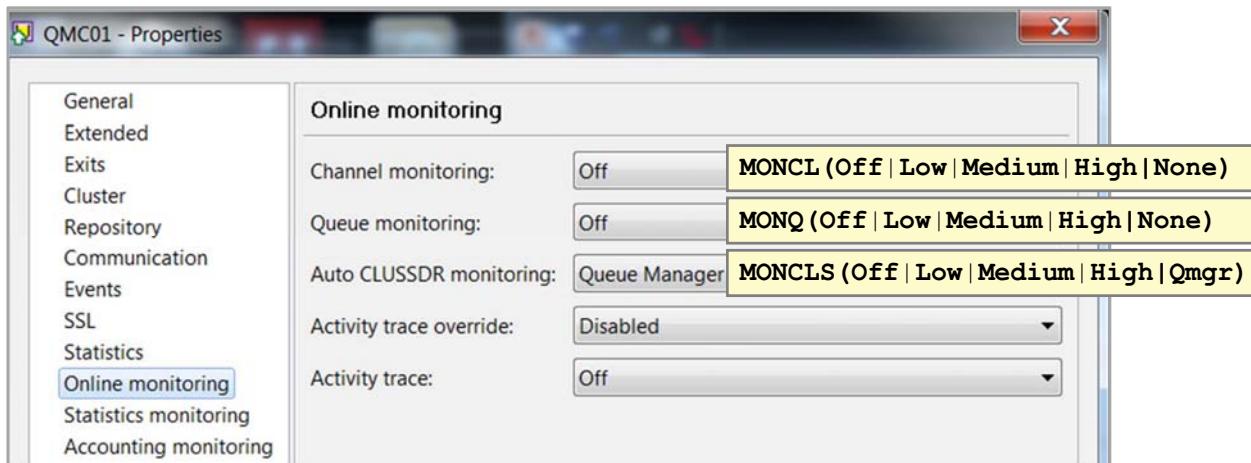
Real-time monitoring can be used in the following tasks:

- Helping system administrators understand the steady state of their MQ system to help with problem diagnosis when a problem occurs in the system.
- Determining the condition of your queue manager at any moment, even if no specific event or problem was detected.
- Assisting with determining the cause of a problem in your system.

With real-time monitoring, information can be returned for either queues or channels. Queue manager, queue, and channel attributes control the amount of real-time information that is returned.



Controlling queue manager real-time monitoring



Monitoring and configuring IBM MQ for performance

© Copyright IBM Corporation 2017

Figure 13-22. Controlling queue manager real-time monitoring

The **Online monitoring** queue manager properties specify whether to collect real-time monitoring data about the current performance of channels that the queue manager hosts.

- To disable real-time monitoring data collection for the queue manager's channels that have the value **Queue Manager** in their **Channel monitoring** attribute, select **Off**.
- To disable online monitoring data collection for all queue manager channels regardless of the setting of the channel's **Channel monitoring** attribute, select **None**.

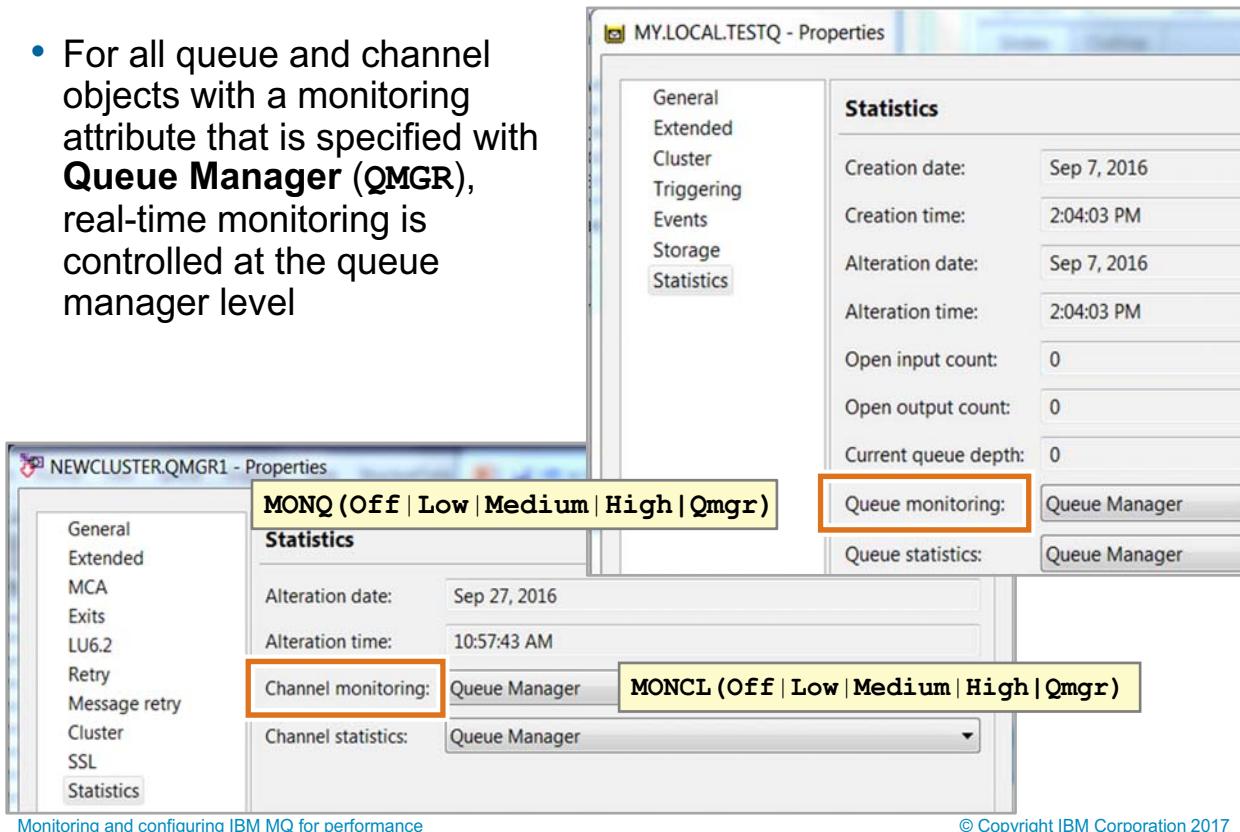
The **Queue monitoring** property specifies whether to collect online monitoring data about the current performance of queues that the queue manager hosts.

The **Auto CLUSSDR monitoring** property specifies whether to collect online monitoring data about the current performance of automatically defined cluster-sender channels.

The **Activity trace** property specifies whether to enable an activity trace. Activity trace was described in detail in Unit 9.

Controlling channel and queue real-time monitoring

- For all queue and channel objects with a monitoring attribute that is specified with **Queue Manager (QMGR)**, real-time monitoring is controlled at the queue manager level



Monitoring and configuring IBM MQ for performance

© Copyright IBM Corporation 2017

Figure 13-23. Controlling channel and queue real-time monitoring

You can enable or disable real-time monitoring for individual queues or channels, or for multiple queues or channels.

To control individual queues, set the queue attribute **MONQ** to enable or disable real-time monitoring. To control many queues, enable or disable real-time monitoring at the queue manager level by using the queue manager attribute **MONQ**. For all queue objects with a monitoring attribute that is specified with the default value **QMGR**, real-time monitoring is controlled at the queue manager level.

To control individual channels, set the channel attribute **MONCHL** to enable or disable real-time monitoring. To control many channels, enable or disable real-time monitoring at the queue manager level by using the queue manager attribute **MONCHL**. For all channel objects with a monitoring attribute that is specified with the default value **QMGR**, real-time monitoring is controlled at the queue manager level.

Displaying queue and channel monitoring data

- Use IBM MQ Explorer content views or MQSC **DISPLAY QSTATUS** and **DISPLAY CHSTATUS** with **MONITOR** option

Example: On queue manager, MONCHL = MEDIUM

On sender channel QM1.QM2, MONCHL = QMGR

DISPLAY CHSTATUS (QM1.QM2) MONITOR

```
CHSTATUS (QM1. QM2)
XMITQ (Q1)
CONNNAME (127.0.0.1)
CURRENT
CHLTYPE (SDR)
STATUS (RUNNING)
SUBSTATE (MQGET)
MONCHL (MEDIUM)
XQTIME (755394737, 755199260)
NETTIME (13372, 13372)
EXITTIME (0,0)
XBATCHSZ (50,50)
COMPTIME (0,0)
STOPREQ (NO)
RQMNAME (QM2)
```

Figure 13-24. Displaying queue and channel monitoring data

To display real-time monitoring information for a queue or channel, use either the MQ Explorer or the appropriate MQSC command. Some monitoring fields display a comma-separated pair of indicator values, which help you to monitor the operation of your queue manager.

Configuring and tuning IBM MQ for performance

- Default properties are configured to produce a fully functioning queue manager by using reasonable amounts of memory and disk space
 - For on-premises installations, IBM MQ is not optimized for performance
 - For IBM MQ Appliance, IBM MQ is optimized for performance
- Apply tuning to all connected queue managers because messaging performance by using more than one queue manager depends on the performance of those other queue managers
- Tuning options:
 - Queue manager logs
 - Queue manager channels
 - Queue manager listeners

[Monitoring and configuring IBM MQ for performance](#)

© Copyright IBM Corporation 2017

Figure 13-25. Configuring and tuning IBM MQ for performance

Unless the MQ queue manager runs on an MQ Appliance, an MQ queue manager with default properties is configured to produce a fully functioning queue manager that uses reasonable amounts of memory and disk space. It is not optimized for performance.

You can make some configuration changes to improve the performance of message processing with MQ.



Attention

It might not be necessary to implement the tuning guidelines that are described in this unit, especially if the message throughput and response time of the queue manager already meet the required level. If applied inappropriately, implementation of some of the tuning options can degrade the performance of a previously balanced system.

Carefully monitor the results of tuning the queue manager to ensure that no adverse effects are evident.

Always thoroughly test your environment after you change any tuning parameters.

Queue manager logs

- Might require performance tuning when the queue manager processes persistent messages, which are stored in logs
- Default settings for new queue managers:
 - **Default log settings** in IBM MQ Explorer properties
 - **LogDefaults** stanza in the `mqsc.ini` file
 - Can override default settings by using `qm.ini` or queue manager **Properties** in IBM MQ Explorer
- Performance factors:
 - Log type*
 - Log file path*
 - Log file pages*
 - Level log write integrity
 - Log buffer pages
 - Number of primary and secondary log files
 - Number of concurrent applications
 - Application processing within a unit of work

* Must be specified when the queue manager is created and cannot be changed

[Monitoring and configuring IBM MQ for performance](#)

© Copyright IBM Corporation 2017

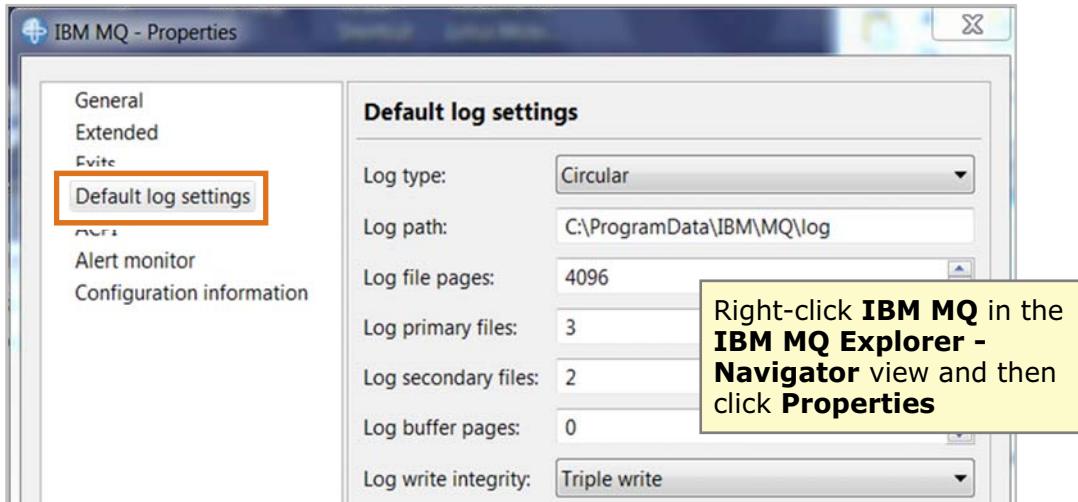
Figure 13-26. Queue manager logs

The location, size, and number of queue manager logs can affect performance. Some performance factors, such as log file location and the type of logging, must be specified when the queue manager is created. Other performance factors such as level of log integrity, log file pages, and log buffer pages can be changed at any time but might require a restart.

Queue manager log settings can be configured in the MQ Explorer queue manager properties and the **LogDefaults** stanza in the `mqsc.ini` file.

Default log settings in IBM MQ Explorer

- Specifies log settings to use for new queue managers that are created in IBM MQ Explorer



Monitoring and configuring IBM MQ for performance

© Copyright IBM Corporation 2017

Figure 13-27. Default log settings in IBM MQ Explorer

The queue manager log properties that can be configured in the MQ Explorer are:

- Log type
- Log path
- Log file pages
- Log primary files
- Log secondary files
- Log buffer pages
- Log write integrity

To access the **Default log settings** in MQ Explorer:

- Right-click the queue manager in the **MQ Explorer - Navigator** view and then click **Properties**.
- From the **Properties** list, click **Default log settings**.

Log type

- Linear logging file extents are continually allocated as required
- Circular logging log file extents are reused after they no longer contain active log data
- For performance, choose circular logging if linear logging is not required for re-creating lost or damaged data by replaying the contents of the log
- Type of logging cannot be changed after a queue manager is created

Figure 13-28. Log type

The type of logging can affect performance.

Circular logging is more efficient because log files are reused when they no longer contain active log data. If circular logging is enabled, ensure that **Log primary files** are greater than 20. Ensure that primary logs satisfy circular logging because secondary logs are formatted each time that they are reused.

Log path

- Specifies directory for the queue manager log files
- Locate the queue manager log on its own disk, particularly when processing large messages, or high message volumes (> 50 messages per second)
 - On Windows, create a directory on fastest local disk available and then specify the directory by using the `-1d` attribute when creating the queue manager
 - On UNIX and Linux, allocate and mount a file system for queue manager files and log before creating the queue manager
- When possible, allocate the log on a device with a battery-backed write cache or use fastest local disk available

Figure 13-29. Log path

The log file path and location can affect performance.

Nonpersistent messages are held in main memory and then saved to the file system as the queue depth increases. Persistent messages are synchronously written to the log.

Queue and log I/O contention can occur due to the queue manager simultaneously updating a queue file and log extent on the same disk.

To avoid potential queue and log I/O contention, put queues and logs on separate and dedicated physical devices.

Log file pages

- Specified on queue manager creation and cannot be changed
- Defines the size of one physical disk extent in units of 4 KB pages
- For Windows:
 - Default number of log file pages is 4096, giving a log file size of 16 MB
 - Minimum number of log file pages is 32
 - Maximum number of log file pages is 65536
- For UNIX and Linux:
 - Default number of log file pages is 4096, giving a log file size of 16 MB
 - Minimum number of log file pages is 64
 - Maximum number of log file pages is 65,535
- For performance, allocate maximum size if disk space is available

Monitoring and configuring IBM MQ for performance

© Copyright IBM Corporation 2017

Figure 13-30. Log file pages

The **Log file pages** option in the queue manager log properties defines the size of one physical disk extent.

The size of the disk extent is directly related to the elapsed times between changing disk extents. It is better to have a smaller number of large extents. The largest size reduces the frequency of switching extents.

Log file page size can be configured in the MQ Explorer queue manager properties.

Log primary and secondary files

- Queue manager allocates and formats primary log file extents when it is first started or when extra extents are added
 - Minimum = 2
 - Maximum = 254 on Windows, or 510 on UNIX and Linux
 - Default = 3
- Queue manager dynamically allocates secondary log file extents when the primary files are exhausted
 - Minimum = 1
 - Maximum = 253 on Windows, or 509 on UNIX and Linux
 - Default = 2
- Total number of primary and secondary log files:
 - Must not exceed 255 on Windows, or 511 on UNIX
 - Must not be less than 3
- Change in value requires queue manager restart
- For performance, ensure that a reasonable number of secondary extents exist for system activity

[Monitoring and configuring IBM MQ for performance](#)

© Copyright IBM Corporation 2017

Figure 13-31. Log primary and secondary files

Records are written into the log buffer with each put, get, and commit. This information is moved onto the log disk. Periodically the checkpoint process decides how many of these log file extents that are in the *active* log must be kept online for recovery purposes. The log extents no longer in the active log are available for achieving when the queue manager uses linear logs.

Ensure that sufficient primary logs are available to hold the active log plus the new log extents that are used until the next checkpoint; otherwise, some secondary logs are temporarily included in the log set. Secondary logs must be instantly formatted, which is an unnecessary delay when the queue manager uses circular logging.

The value for the number of logs is examined when the queue manager is started. You can change this value, but changes do not become effective until the queue manager is restarted, and even then the effect might not be immediate.

Log buffer pages

- Log buffer is a circular piece of main memory where the log records are concatenated so that multiple log records can be written to the log file in a single I/O operation
- Specify the size of the buffers in units of 4 KB pages
 - Minimum = 18
 - Maximum = 512
 - If you specify 0, queue manager selects size
- Improve persistent message throughput of large messages (message size > 1 MB) by increasing **Log buffer pages** to improve likelihood of messages that need only one I/O to get to the disk
- Requires queue manager restart to recognize any changes

Figure 13-32. Log buffer pages

To improve persistent message throughput of large messages, increase the value in the **Log buffer pages** property to improve the likelihood of messages that need only one I/O to get to the disk. A large message is a message greater than 1 MB.

You can reduce the memory by using a smaller log buffer page without affecting throughput in environments that process a few (under 100) small persistent messages. A small message is a message less than 10 KB.

Log write integrity level

- Method that the queue manager logger uses to reliably write log records
 - **SingleWrite**: Safe for the logger to write log records in a single write because the hardware assures full write integrity
 - **TripleWrite**: Default method that assures full write integrity when write integrity hardware is not available
- On Windows, change **LogWriteIntegrity** string value under `IBMMQSeries\CurrentVersion\Configuration\QueueManager` in the Windows registry
- On UNIX and Linux, change **LogWriteIntegrity** value in **Log** stanza of `qm.ini` file
- Level change requires queue manager restart

Monitoring and configuring IBM MQ for performance

© Copyright IBM Corporation 2017

Figure 13-33. Log write integrity level

MQ provides full write integrity of logs where hardware that assures write integrity is not available.

Some hardware promises if a write operation writes a page and fails for any reason, a subsequent read of the same page shows that each byte in the buffer is the same as before the write.

On this type of hardware, it is safe for the logger to write log records in a single write as the hardware assures full write integrity. This method provides the highest level of performance.

The log write integrity level can be configured in the MQ Explorer queue manager properties and in the **Log** stanza of the queue manager configuration file.

Queue manager Log settings in qm.ini

```

#* Module Name: qm.ini                                     *#
#* Type : IBM MQ queue manager configuration file        *#
# Function : Define the configuration of a single queue manager *#
#*                                                       *#
#*****                                                       *#
#* Notes :                                                 *#
#* 1) This file defines the configuration of the queue manager *#
#*                                                       *#
#*****                                                       *#
ExitPath:
  ExitsDefaultPath=/var/mqm/exits
  ExitsDefaultPath64=/var/mqm/exits64

. . .

Log:
  LogPrimaryFiles=3
  LogSecondaryFiles=2
 LogFilePages=4096
  LogType=CIRCULAR
  LogBufferPages=0 1
  LogWriteIntegrity = TripleWrite
  LogDefaultPath=/var/mqm/log/saturn!queue!manager/

```

Figure 13-34. Queue manager Log settings in qm.ini

This figure shows an example of the **Log** stanza in the **qm.ini** file.

Queue manager channels performance tuning

- If environment is stable, run channels as trusted or fast path IBM MQ applications to give a performance benefit through reduced code pathlength
- Configure by using one of two options:
 - Specify a value of **MQIBindType=FASTPATH** in the **Channels** stanza of the **qm.ini** or registry file
 - Set the environment variable **MQ_CONNECT_TYPE = FASTPATH** in the environment in which the channel is started
- Do not use trusted or fast path channels:
 - If channel exits are used, a potential exists for the exit to corrupt the queue manager if the exits are not correctly written and thoroughly tested
 - If **STOP CHANNEL MODE (FORCE)** command is used
 - If the environment is unstable with regular component failure

[Monitoring and configuring IBM MQ for performance](#)

© Copyright IBM Corporation 2017

Figure 13-35. Queue manager channels performance tuning

If your MQ environment is stable, fast path channels can increase throughput for both nonpersistent and persistent messaging. For persistent messages, the improvement is for the path through the queue manager, and does not affect performance when writes to the log disk occur.



Note

Because the greater proportion of time for persistent messages is while writing to the log disk, the performance improvement for fast path channels is less apparent with persistent messages than with nonpersistent messages.

Setting the **MQIBindType** attribute in the **qm.ini** file to **FASTPATH** causes the channel to run in “trusted” mode. Trusted applications do not use a thread in the agent process. No interprocess communication (IPC) between the channel and agent exists because the agent does not exist in this connection.

If the channel is run in **STANDARD** mode, then any messages that are passed between the channel and agent use IPC memory that is dynamically obtained and held only for the lifetime of the **MQGET**. Standard channels each require an extra 80 KB of memory. When the message rate increases, more IPC memory is used in parallel.

Queue manager listeners performance tuning

- Run as trusted or fast path IBM MQ applications to give a performance benefit through reduced code pathlength
- Set the environment variable `MQ_CONNECT_TYPE=FASTPATH` in the environment in which the listener is started

- Works for listeners that are started when the `runmqlsr` command is run manually or in a script
 - `MQ_CONNECT_TYPE=FASTPATH` needs to be present only in the shell from which the `runmqlsr` command is entered

- Works for listeners that were defined by using the `DEFINE LISTENER` MQSC command
 - `MQ_CONNECT_TYPE=FASTPATH` must be set in the environment in which the queue manager is started

[Monitoring and configuring IBM MQ for performance](#)

© Copyright IBM Corporation 2017

Figure 13-36. Queue manager listeners performance tuning

Running queue manager listeners as trusted or fast path applications might give a performance benefit through reduced code path length.

More resource savings are available by using the `runmqlsr` listener command rather than InetD, including a reduced requirement on virtual memory, number of processes, and file handles.



Note

The performance improvement for fast path channels is less apparent with persistent messages than with nonpersistent messages.

Unit summary

- Describe the statistics and accounting data that IBM MQ provides
- View and generate accounting and statistical data
- Subscribe to IBM MQ statistic topics
- Interpret statistics and accounting data to identify possible system performance benefits
- Configure and tune IBM MQ for improved performance

Monitoring and configuring IBM MQ for performance

© Copyright IBM Corporation 2017

Figure 13-37. Unit summary

Review questions

1. Which of the following are types of statistics that are collectable by IBM MQ?
 - A. Queue
 - B. Process
 - C. Channel
 - D. Queue manager

2. Select all answers that are correct for the following statement.
Accounting data includes:
 - A. Application data
 - B. Queue data
 - C. Publish/subscribe data
 - D. Byte counts
 - E. All of the above



Monitoring and configuring IBM MQ for performance

© Copyright IBM Corporation 2017

Figure 13-38. Review questions

Write your answers down here:

- 1.

- 2.

Review answers

1. Which of the following are types of statistics that are collectable by IBM MQ?
 - A. Queue
 - B. Process
 - C. Channel
 - D. Queue manager

The answer is A, C, and D.
2. Select all answers that are correct for the following statement.
Accounting data includes:
 - A. Application data
 - B. Queue data
 - C. Publish/subscribe data
 - D. Byte counts
 - E. All of the above

The answer is E.



Exercise: Monitoring IBM MQ for performance

Monitoring and configuring IBM MQ for performance

© Copyright IBM Corporation 2017

Figure 13-40. Exercise: Monitoring IBM MQ for performance

In this exercise, you enable and configure the MQ online monitoring, statistics, and accounting.

Exercise objectives

- Enable accounting and statistics collection in IBM MQ
- View accounting and statistics data
- Configure a queue manager for online monitoring
- Monitor system resource usage



[Monitoring and configuring IBM MQ for performance](#)

© Copyright IBM Corporation 2017

Figure 13-41. Exercise objectives

See the *Course Exercises Guide* for detailed instruction.

Unit 14. Course summary

Estimated time

00:30

Overview

This unit summarizes the course and provides information for future study.

Unit objectives

- Explain how the course met its learning objectives
- Access the IBM Training website
- Identify other IBM Training courses that are related to this topic
- Locate appropriate resources for further study

Course summary

© Copyright IBM Corporation 2017

Figure 14-1. Unit objectives

Course objectives

- Describe the IBM MQ deployment options
- Plan for the implementation of IBM MQ on-premises or in the Cloud
- Use IBM MQ commands and the IBM MQ Explorer to create and manage queue managers, queues, and channels
- Use the IBM MQ sample programs and utilities to test the IBM MQ network
- Enable a queue manager to exchange messages with another queue manager
- Configure client connections to a queue manager
- Use a trigger message and a trigger monitor to start an application to process messages

Course summary

© Copyright IBM Corporation 2017

Figure 14-2. Course objectives

Course objectives

- Implement basic queue manager restart and recovery procedures
- Use IBM MQ troubleshooting tools to identify the cause of a problem in the IBM MQ network
- Plan for and implement basic IBM MQ security features
- Use accounting and statistics messages to monitor the activities of an IBM MQ system
- Define and administer a simple queue manager cluster

Course summary

© Copyright IBM Corporation 2017

Figure 14-3. Course objectives

IBM Training



To learn more on the subject

- IBM Training website:
www.ibm.com/training
- IBM Knowledge Center for IBM MQ V9.0:
www.ibm.com/support/knowledgecenter/SSFKSJ_9.0.0/com.ibm.mq.helphome.v90.doc/WelcomePagev9r0.htm

Course summary

© Copyright IBM Corporation 2017

Figure 14-4. To learn more on the subject



Enhance your learning with IBM resources

Keep your IBM Cloud skills up-to-date

- IBM offers resources for:
 - Product information
 - Training and certification
 - Documentation
 - Support
 - Technical information



- To learn more, see the IBM Cloud Education Resource Guide:
 - www.ibm.biz/CloudEduResources

Course summary

© Copyright IBM Corporation 2017

Figure 14-5. Enhance your learning with IBM resources

Unit summary

- Explain how the course met its learning objectives
- Access the IBM Training website
- Identify other IBM Training courses that are related to this topic
- Locate appropriate resources for further study

Course summary

© Copyright IBM Corporation 2017

Figure 14-6. Unit summary

Appendix A. List of abbreviations

ACL	access control list
AIX	Advanced IBM UNIX
API	application programming interface
CCDT	client channel definition table
CCSID	coded character set identifier
CDR	Continuous Delivery Release
COBOL	Common Business Oriented Language
CPU	central processing unit
DLH	dead-letter header
DNS	Domain Name System
EOF	end of file
FFDC	first failure data capture
FFST	First Failure Support Technology
FIPS	Federal Information Processing Standard
FTP	File Transfer Protocol
GPFS	General Parallel File System
HP-UX	Hewlett-Packard UNIX
HTTP	Hypertext Transfer Protocol
IaaS	infrastructure as a service
IBM	International Business Machines Corporation
IP	Internet Protocol
IPC	interprocess communication
IPLA	International Program License Agreement
JMS	Java Message Service
JNDI	Java Naming and Directory Interface
LSN	log sequence number
LTSR	Long Term Support Release
MCA	message channel agent
MQ	message queue
MQDLH	MQ dead-letter queue header

MQFB	MQ feedback code
MQI	Message Queue Interface
MQMD	MQ message descriptor
MQRC	MQ reason code
MQSC	MQ script commands
MQTMC2	MQ trigger message 2 character format
MQTT	MQ Telemetry Transport
MQXQH	MQ transmission queue header
NetBIOS	Network Basic Input/Output System
OAM	object authority manager
QoS	quality of service
PaaS	platform as a service
PCF	programmable command format
PI	program isolation
PID	process ID
RAM	random access memory
RCP	rich client platform
SMIT	System Management Interface Tool
SNA	Systems Network Architecture
SPX	Sequenced Packet Exchange
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
TCP/IP	Transmission Control Protocol/Internet Protocol
TLS	Transport Layer Security
UNIX	Uniplexed Information and Computing System
URL	Uniform Resource Locator
z/OS	zSeries operating system



IBM Training



© Copyright International Business Machines Corporation 2017.