

Course Guide

IBM MQ V9.1 System Administration

Course code WM156 / ZM156 ERC 1.1



April 2020 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 2020.

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	xviii
Course description	xix
Agenda	xxi
Unit 1. Introducing IBM MQ	1-1
How to check online for course material updates	1-2
Unit objectives	1-3
Topics	1-4
1.1. Introducing IBM MQ	1-5
Introducing IBM MQ	1-6
Problem: How can independent programs communicate?	1-7
Solution: Message queuing	1-8
What is IBM MQ? (1 of 2)	1-9
What is IBM MQ? (2 of 2)	1-10
IBM MQ enterprise messaging features	1-11
Benefits of message queuing	1-13
1.2. IBM MQ components and features	1-14
IBM MQ components and features	1-15
Anatomy of an IBM MQ system	1-16
Messages	1-18
Queues	1-19
Queue managers	1-20
IBM MQ clients	1-21
Channels	1-22
1.3. What forms of IBM MQ are available	1-23
What forms of IBM MQ are available	1-24
Deployment models	1-25
IBM MQ Advanced	1-26
IBM MQ on Cloud managed service	1-27
IBM MQ in containers	1-28
1.4. Administrative role and tools	1-29
Administrative role and tools	1-30
IBM MQ administrative tasks	1-31
IBM MQ administration interfaces	1-32
Working with IBM MQ Explorer	1-33
Working with IBM MQ Console	1-34
1.5. Installing IBM MQ	1-35
Installing IBM MQ	1-36
Long Term Support and Continuous Delivery	1-37
Multiple version installations	1-38
Choosing an installation path	1-39
Choosing what to install	1-40
Review your installation	1-41
Validate your installation with MQ Explorer	1-42
Uninstalling IBM MQ	1-43
Finding more information	1-44
1.6. Configuring the mqweb server for IBM MQ Console and REST administration	1-45
Configuring the mqweb server for IBM MQ Console and REST administration	1-46

Configuring mqweb server for IBM MQ Console and REST administration	1-47
Commands to control the mqweb server	1-48
Working with IBM MQ Console	1-49
Unit summary	1-50
Review questions	1-51
Review answers	1-52
Exercise: Getting started with IBM MQ	1-53
Exercise introduction	1-54
Unit 2. Working with IBM MQ administration tools	2-1
Unit objectives	2-2
Topics	2-3
2.1. IBM MQ administration tools	2-4
IBM MQ administration tools	2-5
Administration interfaces	2-6
2.2. IBM MQ control commands	2-7
IBM MQ control commands	2-8
IBM MQ command modes	2-9
Creating a queue manager	2-11
Queue manager default and system objects	2-13
Queue manager logs	2-14
Naming IBM MQ objects	2-16
Basic queue manager control commands	2-17
Stopping a queue manager	2-18
Stopping and starting the IBM MQ service on Windows	2-20
Queue manager configuration file	2-21
When do you edit the queue manager configuration file?	2-22
2.3. IBM MQ script commands (MQSC)	2-23
IBM MQ script commands (MQSC)	2-24
Using MQSC to configure IBM MQ objects	2-25
Starting the MQSC command interface	2-26
Stopping the MQSC command interface	2-28
MQSC command rules	2-29
Using filters in MQSC commands	2-30
MQSC filtering example	2-31
MQSC script files	2-32
Running MQSC script files	2-34
Defining a local queue	2-35
Defining a local queue examples	2-36
Defining other queue types	2-37
Clearing and deleting queues	2-39
Displaying queue manager and queue attributes	2-40
Modifying attributes	2-41
2.4. Special queues	2-42
Special queues	2-43
Special queues	2-44
Defining a dead-letter queue	2-46
2.5. Sample programs	2-47
Sample programs	2-48
Testing with IBM MQ sample programs	2-49
Sample program example	2-51
2.6. Working with queue manager sets	2-52
Working with queue manager sets	2-53
Queue manager sets (1 of 2)	2-54
Queue manager sets (2 of 2)	2-55
2.7. Stopping and removing a queue manager	2-56

Stopping and removing a queue manager	2-57
Stopping and deleting queue managers	2-58
Deleting queue managers	2-59
Unit summary	2-60
Review questions (1 of 2)	2-61
Review questions (2 of 2)	2-62
Review answers (1 of 2)	2-63
Review answers (2 of 2)	2-64
Exercise: Using commands to create queue managers and queues	2-65
Exercise objectives	2-66
Unit 3. Configuring distributed queuing.....	3-1
Unit objectives	3-2
Topics	3-3
3.1. Distributed queuing.....	3-4
Distributed queuing	3-5
Message-queuing styles	3-6
3.2. Point-to-point topology	3-8
Point-to-point topology	3-9
Create a point-to-point topology	3-10
Point-to-point: Required queues	3-11
Moving and transport services	3-12
Channel initiators and listeners	3-14
Point-to-point: How does it work?	3-15
Point-to-point: Manual versus clustering (1 of 2)	3-17
Point-to-point: Manual versus clustering (2 of 2)	3-18
3.3. Working with channels	3-19
Working with channels	3-20
Message channels	3-21
Message channel types	3-22
Sender to receiver message channel	3-23
Requester to server message channel	3-24
Requester to sender message channel	3-25
DEFINE CHANNEL command	3-26
Defining channels example	3-27
Minimum channel definitions for remote communication	3-28
Distributed queuing configuration example	3-29
Starting a message channel	3-30
Displaying channel status	3-32
Channel states (1 of 2)	3-33
Channel states (2 of 2)	3-35
Channel initiators and listeners	3-36
Controlling the listener process	3-37
Listener object	3-38
Channel-exit programs for message channels (1 of 2)	3-39
Channel-exit programs for message channels (2 of 2)	3-40
Reasons why a channel might fail to start	3-41
3.4. Message data conversion.....	3-42
Message data conversion	3-43
Data representation and conversion	3-44
Data conversion example	3-45
Application data conversion configuration	3-46
What data conversion can be done	3-47
3.5. Message delivery	3-48
Message delivery	3-49
When a message arrives at a queue manager	3-50

What happens when a message cannot be delivered?	3-51
Dead-letter header (MQDLH)	3-52
Guidelines for handling undelivered messages	3-53
Unit summary	3-54
Review questions	3-55
Review answers	3-56
Exercise: Implementing distributed queuing	3-57
Exercise introduction	3-58
Unit 4. Managing clients and client connections	4-1
Unit objectives	4-2
Topics	4-3
4.1. IBM MQ client introduction	4-4
IBM MQ client introduction	4-5
IBM MQ client application options	4-6
IBM MQ redistributable client	4-7
IBM MQ MQI client	4-8
IBM MQ MQI client component	4-9
IBM MQ application	4-10
MQI channel	4-11
4.2. Client configurations	4-12
Client configurations	4-13
Defining connections between the server and client	4-14
Defining an MQI channel	4-15
Defining client-connection channels in IBM MQ Explorer	4-16
Client configuration methods	4-17
Method 1: Using MQSERVER	4-18
Method 2: Client configuration file	4-19
Method 3: Using a client channel definition table (CCDT)	4-20
Using a CCDT example	4-21
Accessing a CCDT	4-22
URL support for CCDT files	4-23
Precedence order for finding the CCDT	4-24
Weighted selection on client channels	4-25
MQI channel instances	4-26
Instance limits on SVRCONN channels	4-27
Auto-definition of channels	4-28
Testing client connectivity	4-29
Testing client-to-server connectivity	4-30
4.3. Client modes for MQSC	4-31
Client modes for MQSC	4-32
MQSC client modes	4-33
MQSC client-local mode (-n)	4-34
MQSC client-connect mode (-c)	4-35
4.4. Troubleshooting tips	4-36
Troubleshooting tips	4-37
Troubleshooting IBM MQ channels	4-38
Listener problems	4-39
Channel configuration problems	4-40
Network connectivity problems	4-41
In-doubt channels	4-42
MQSC connection and channel status commands	4-43
Channel monitoring	4-44
Channel monitoring example	4-45
Troubleshooting IBM MQ clients	4-46
Unit summary	4-47

Review questions	4-48
Review answers	4-49
Exercise: Connecting an IBM MQ client	4-50
Exercise introduction	4-51
Unit 5. Advanced IBM MQ client features.....	5-1
Unit objectives	5-2
Topics	5-3
Extended transactional client	5-4
Extended transactional client	5-5
Units of work and sync points	5-6
Control flow	5-7
5.1. Sharing conversations	5-8
Sharing conversations	5-9
Sharing conversations overview	5-10
Sharing conversation requirements	5-11
SHARECNV channel property	5-12
Sharing conversations configuration: Server-connection	5-14
Sharing conversations configuration: Client-connection	5-15
Sharing conversations example	5-16
Sharing conversations channel status fields	5-17
Sharing conversations and existing channel status fields	5-18
Sharing conversations and channel exit considerations	5-19
Sharing conversation performance	5-20
5.2. Read ahead	5-21
Read ahead	5-22
Standard MQGET processing	5-23
Read ahead overview	5-24
Read ahead advantages	5-25
Read ahead considerations	5-26
Default read ahead configuration (1 of 2)	5-27
Read ahead configuration (2 of 2)	5-28
Read ahead memory buffer size	5-29
Read ahead buffer updates	5-30
Using IBM MQ Explorer to get read ahead status	5-31
Using MQSC to get read ahead status	5-32
5.3. Asynchronous put	5-33
Asynchronous put	5-34
Asynchronous put introduction	5-35
MQPUT processing	5-36
MQPUT in client mode with asynchronous put	5-37
When asynchronous put applies	5-38
Sync point	5-39
Setting the default put response type on a queue	5-40
Unit summary	5-41
Review questions	5-42
Review answers	5-43
Unit 6. Working with queue manager clusters.....	6-1
Unit objectives	6-2
Topics	6-3
6.1. Introducing clusters	6-4
Introducing clusters	6-5
Clusters	6-6
Benefits of clustering	6-7
Overview of cluster components	6-8

What makes a cluster work?	6-9
Considerations for a full repository	6-10
Steps to set up a basic cluster	6-12
Define cluster-receiver channels (CLUSRCVR)	6-13
Define cluster-sender channel (CLUSSDR)	6-14
Define local queues to the cluster	6-15
System cluster queues	6-17
Cluster administration considerations	6-18
Troubleshooting clusters	6-19
Cluster troubleshooting	6-20
6.2. Cluster workload management.	6-21
Cluster workload management	6-22
Cluster workload management options	6-23
Channel workload balancing	6-24
When workload balancing can occur	6-25
Cluster workload management algorithm: Summary	6-27
Queue manager cluster scenario	6-28
Use-queue basics	6-29
Use-queue example	6-30
Rank basics	6-31
Channel rank example	6-32
Channel and queue rank example	6-33
Channel and queue priority basics	6-34
Using CLWLPTY to identify primary and backup servers	6-35
Priority example (1 of 2)	6-36
Priority example (2 of 2)	6-37
Most recently used channels	6-38
Most recently used channels basics	6-39
Using processing power to control the workload	6-40
Channel weight basics	6-41
Channel weight example	6-42
Unit summary	6-43
Review questions	6-44
Review answers	6-45
Exercise: Implementing a basic cluster	6-46
Exercise introduction	6-47
Unit 7. Publish/subscribe messaging.	7-1
Unit objectives	7-2
Topics	7-3
7.1. Introducing publish/subscribe	7-4
Introducing publish/subscribe	7-5
What is publish/subscribe?	7-6
Example: Stock market	7-7
Publish/subscribe patterns	7-8
7.2. Topic trees and topic strings	7-9
Topic trees and topic strings	7-10
It's all about the topic tree!	7-11
Pattern matching	7-12
Designing the topic tree: Make it extendible	7-13
Designing the topic tree: Shape	7-15
Why worry about the size and shape of the topic tree?	7-16
7.3. Topic objects	7-18
Topic objects	7-19
Topic objects	7-20
Topic object attributes	7-22

Topic security	7-23
7.4. Managing subscriptions	7-24
Managing subscriptions	7-25
How do you create subscriptions	7-26
Subscription lifetime	7-27
Subscription queue management	7-28
7.5. Managing publication	7-29
Managing publication	7-30
Publication success	7-31
Retained publications	7-32
7.6. Publish/subscribe topologies	7-33
Publish/subscribe topologies	7-34
Distributed publish/subscribe	7-35
Distributed publish/subscribe topologies	7-36
Publish/subscribe hierarchies	7-38
Publish/subscribe hierarchy configuration	7-39
Publish/subscribe clusters	7-40
Cluster topics	7-42
Cluster modes for routing publications	7-44
Cluster topic routing	7-45
Routing behavior for publish/subscribe clusters	7-47
Topic host routing requirements	7-48
Subscription propagation	7-49
7.7. Comparing topologies	7-50
Comparing topologies	7-51
Proxy subscription in a hierarchy	7-52
Proxy subscriptions in a cluster	7-53
Scaling	7-54
Publication flows	7-55
Configuration	7-56
Availability	7-57
Comparison summary	7-58
7.8. Troubleshooting	7-59
Troubleshooting	7-60
Tips	7-61
Problem determination (1 of 2)	7-62
Problem determination (2 of 2)	7-63
Loop detection	7-65
Stopping a queue manager	7-66
Unit summary	7-67
Review questions	7-68
Review answers	7-69
Exercise: Configuring publish/subscribe message queuing	7-70
Exercise introduction	7-71
Unit 8. Implementing basic security in IBM MQ	8-1
Unit objectives	8-2
Security mechanisms	8-3
IBM MQ security implementations	8-4
Planning for security	8-5
IBM MQ access control overview	8-6
OAM installable service	8-7
Principals and groups	8-8
Access control with the OAM	8-9
OAM access control lists	8-10
Set or reset authorization	8-12

Display authorization	8-14
Create authorization report	8-15
Using MQSC to manage authorization	8-16
Using IBM MQ Explorer to manage authorization	8-17
Queue manager authorization in IBM MQ Explorer	8-18
Queue authorization in IBM MQ Explorer	8-19
Channel authorization in IBM MQ Explorer	8-20
Access control for IBM MQ control commands	8-21
Security and distributed queuing	8-22
Security authorization for remote queues	8-23
Authorization checking in the MQI	8-24
Authority events (1 of 2)	8-25
Authority events (2 of 2)	8-26
Channel authentication control	8-27
Channel authentication commands	8-28
Channel authentication example	8-29
IBM MQ security exits	8-30
Connection authentication	8-31
Enabling connection authentication on a queue manager	8-32
Enabling connection authentication examples	8-33
Unit summary	8-34
Review questions	8-35
Checkpoint answers	8-36
Exercise: Controlling access to IBM MQ	8-37
Exercise introduction	8-38
Unit 9. Securing IBM MQ channels with TLS	9-1
Unit objectives	9-2
Topics	9-3
9.1. TLS overview	9-4
TLS overview	9-5
The security problems	9-6
Data encryption and signing	9-7
Elementary encryption: Symmetric key	9-8
Elementary encryption: Asymmetric keys	9-9
Digital signature	9-10
Relative authentication	9-11
The key distribution problem	9-12
Digital certificates	9-13
Trusting a digital certificate	9-14
Certificate revocation	9-15
Digital certificate details	9-16
Distinguished name	9-17
Transport Layer Security (TLS) concepts	9-18
9.2. Implementing TLS in IBM MQ	9-19
Implementing TLS in IBM MQ	9-20
TLS support in IBM MQ	9-21
Cipher specification support in IBM MQ	9-22
IBM MQ connection procedure with TLS	9-23
TLS and IBM MQ certificate exchange	9-24
Configuring TLS on queue managers	9-25
Managing certificates	9-26
Creating certificates	9-27
Creating a key repository with IBM Key Management (1 of 2)	9-28
Creating a key repository with IBM Key Management (2 of 2)	9-29
Using IBM Key Management to create a self-signed personal certificate	9-30

Using IBM Key Management to add a CA certificate	9-31
Command options for managing certificates and keys	9-32
Using runmqakm to create and initialize a key database	9-33
Using runmqakm to generate a self-signed certificate	9-34
Key repository	9-35
Labeling the certificate	9-37
Benefits of labeling the certificate	9-38
Configuring TLS on queue manager with IBM MQ Explorer	9-39
Configuring TLS on channels	9-40
Distinguished Name matching support in IBM MQ	9-41
Defining certificates for channels (1 of 2)	9-42
Defining certificates for channels (2 of 2)	9-43
Channel attributes for TLS	9-44
SSLPEER	9-45
SSLPEER mapping	9-46
Displaying certificate details	9-47
Checking certificate status with OCSP	9-48
Setting OSCP properties in IBM MQ Explorer (1 of 2)	9-49
Setting OSCP properties in IBM MQ Explorer (2 of 2)	9-50
Accessing CRLs and ARLs	9-51
Accessing CRLs and ARLs by using IBM MQ Explorer	9-53
Possible authentication failures	9-54
Using secret key reset	9-55
Refreshing TLS on IBM MQ	9-56
Security administration tasks	9-57
TLS implementation scenario (1 of 3)	9-58
TLS implementation scenario (2 of 3)	9-59
TLS implementation scenario (3 of 3)	9-60
Security problems and solutions	9-61
IBM MQ Advanced Message Security	9-62
Unit summary	9-63
Review questions	9-64
Review answers	9-65
Exercise: Securing channels with TLS	9-66
Exercise introduction	9-67
Unit 10. Authenticating channels and connections	10-1
Unit objectives	10-2
Topics	10-3
10.1. Authentication and authorization overview	10-4
Authentication and authorization overview	10-5
Authentication	10-6
Authorization	10-7
Identification and authentication in IBM MQ	10-8
Approach to planning security	10-9
Secure connections in IBM MQ Explorer	10-10
Setting client connection default values	10-11
Setting passwords that IBM MQ Explorer uses	10-12
Security for remote queue manager connection	10-13
Security settings for remote queue managers (1 of 4)	10-14
Security settings for remote queue managers (2 of 4)	10-15
Security settings for remote queue managers (3 of 4)	10-16
Security settings for remote queue managers (4 of 4)	10-17
User ID when using bindings mode	10-18
Queue manager to queue manager	10-19
Why using TLS can lead to a false sense of security	10-20

IBM MQ Advanced Message Security	10-21
10.2. Object authorizations review	10-22
Object authorizations review	10-23
Object authority manager (OAM)	10-24
Managing authority records in MQSC	10-25
Application to queue manager	10-26
Blank MCAUSER: Main cause of security exposures	10-27
MCAUSER remedy	10-28
Secure remote administration	10-29
Authorizing groups and principals	10-31
Interactive users with OAM	10-32
Considerations for applications with OAM	10-33
OAM and queue manager to queue manager	10-34
A note about transmission queues	10-35
PUTAUT: Put Authority for channels	10-36
Security guidelines	10-37
10.3. Connection authentication	10-38
Connection authentication	10-39
Connection authentication	10-40
Enabling connection authentication on a queue manager	10-41
Connection authentication configuration	10-42
Connection authentication error notification	10-44
Connection authentication and authorization	10-45
Connection authentication user repositories	10-47
LDAP user repository	10-48
Secure connection to an LDAP server	10-50
Connection authentication and authorization: LDAP	10-52
User ID and password for IBM MQ Explorer (1 of 2)	10-53
User ID and password for IBM MQ Explorer (2 of 2)	10-54
User IDs and passwords in IBM MQ programs	10-55
Connection authentication summary	10-56
10.4. Channel authentication	10-57
Channel authentication	10-58
Channel authentication rules	10-59
Channel access blocking points	10-60
Channel authentication example	10-61
Precedence order	10-62
Precedence order example	10-63
Using IP addresses in channel authentication rules	10-64
Using hostnames in channel authentication rules	10-65
Getting the hostname	10-66
Connection authentication configuration granularity	10-68
Diagnosing hostname lookup failures	10-69
Using MATCH(RUNCHECK) example	10-70
More security options	10-71
Dynamic mapping of MCAUSER	10-72
Dynamic MCAUSER mapping example	10-73
User ID blocking with CHLAUTH BLOCKUSER	10-74
Filtering by IP address	10-75
Filtering by IP address: Example	10-76
Filtering by IP address: Guidelines	10-77
Step-by-step guide to a basic secure setup	10-78
Step 1: Set MCAUSER	10-79
Step 2: Provision user IDs and groups	10-80
Step 3: Set channels that are not SYSTEM channels	10-81
Step 4: Authorize groups	10-82

Step 5: Set up strong authentication	10-83
10.5. Channel exits	10-84
Channel exits	10-85
Channel exit programs	10-86
Channel exits for link level security	10-87
Channel auto-definition exit	10-88
Channel exits on message channels	10-89
Channel exit configuration	10-90
Location of exit modules	10-91
Client-side security exit	10-92
Unit summary	10-93
Review questions	10-94
Review answers	10-95
Exercise: Implementing connection authentication	10-96
Exercise introduction	10-97
Unit 11. Supporting JMS with IBM MQ	11-1
Unit objectives	11-2
Topics	11-3
11.1. IBM MQ and JMS	11-4
IBM MQ and JMS	11-5
What is JMS?	11-6
JMS elements	11-7
JMS administered objects	11-8
JMS providers	11-9
IBM MQ classes for Java or JMS	11-10
IBM MQ classes for JMS and IBM MQ classes for Java	11-11
JMS message properties	11-12
Support for JMS 2.0 in IBM MQ	11-13
11.2. Managing JMS resources with IBM MQ Explorer	11-14
Managing JMS resources with IBM MQ Explorer	11-15
Configuring JMS administered objects in IBM MQ	11-16
IBM MQ Explorer and JMS	11-17
Adding an initial context	11-18
Connecting the initial context	11-19
Creating a connection factory (1 of 3)	11-20
Creating a connection factory (2 of 3)	11-21
Creating a connection factory (3 of 3)	11-22
Creating a JMS destination (1 of 3)	11-23
Creating a destination (2 of 3)	11-24
Creating a destination (3 of 3)	11-25
Mapping between IBM MQ and JMS objects (1 of 2)	11-26
Mapping between IBM MQ and JMS objects (2 of 2)	11-27
JMS object creation wizards	11-28
IBM MQ classes for JMS problem determination	11-29
Unit summary	11-30
Review questions	11-31
Review answers	11-32
Unit 12. Diagnosing problems	12-1
Unit objectives	12-2
Topics	12-3
12.1. Finding a missing message	12-4
Finding a missing message	12-5
Where is the message?	12-6
Dead-letter queue	12-7

Missing message checklist	12-8
Finding the dead-letter reason in IBM MQ Explorer	12-9
Finding the dead-letter reason with amqsbcg	12-10
Checking IBM MQ reason codes	12-11
Checking for IBM MQ network problems	12-12
Checking message persistence	12-14
Checking message expiration	12-15
Uncommitted messages	12-16
Displaying the queue status	12-17
Checking for channel problems	12-18
Resolving channel triggering problems	12-19
Remote administration failures	12-20
Checking configuration files	12-21
12.2. Cluster rerouting	12-22
Cluster rerouting	12-23
Cluster Workload Management (WLM)	12-24
Cluster Queue Monitoring sample program (AMQSCLM)	12-25
Normal operation with AMQSCLM	12-26
Application fails	12-27
AMQSCLM detects application failure	12-28
AMQSCLM redirects messages	12-29
AMQSCLM redistributes messages	12-30
12.3. First Failure Support Technology (FFST)	12-31
First Failure Support Technology (FFST)	12-32
First Failure Support Technology (FFST)	12-33
FFST files	12-34
FFST report example	12-35
12.4. Error logs	12-36
Error logs	12-37
IBM MQ error logs	12-38
Example error log contents	12-39
IBM MQ AMQ messages	12-40
AMQ message example	12-41
12.5. IBM MQ trace	12-42
IBM MQ trace	12-43
Application activity trace	12-44
The mqat.ini file	12-45
The mqat.ini file	12-46
Subscribing to activity trace data	12-47
Activity trace topics and subscriptions	12-48
Example activity trace topic strings	12-50
Activity trace sample program	12-51
Tracing IBM MQ components	12-52
Trace commands	12-53
Enabling trace in IBM MQ Explorer	12-54
Example trace on Windows	12-55
Unit summary	12-56
Exercise: Running an IBM MQ trace	12-57
Exercise introduction	12-58
Unit 13. Backing up and restoring IBM MQ messages and object definitions	13-1
Unit objectives	13-2
Topics	13-3
13.1. Logs and recovery	13-4
Logs and recovery	13-5
Circular logs	13-6

Linear logs	13-7
Types of logs	13-8
Circular versus linear	13-9
Dumping the log	13-10
Recovering persistent messages	13-12
Damaged objects and media recovery	13-13
Recording media image to a log	13-14
Re-creating an object from an image	13-15
Valid object types for recording and re-creating images	13-16
13.2. Backing up the IBM MQ file system	13-17
Backing up the IBM MQ file system	13-18
The need to back up	13-19
Backing up the IBM MQ file system	13-20
13.3. Backing up object definitions	13-21
Backing up object definitions	13-22
Overview of IBM MQ objects	13-23
Why back up IBM MQ object definitions	13-24
Capturing IBM MQ object definitions	13-25
Capturing resource definitions with the DISPLAY command	13-26
Save IBM MQ configuration: <code>dmpmqcfg</code>	13-27
Backing up queue manager configuration	13-28
Syntax examples of <code>dmpmqcfg</code>	13-29
Using <code>dmpmqcfg</code> : Output MQSC	13-30
Using <code>dmpmqcfg</code> : Output <code>setmqaut</code>	13-31
Overview of security definitions	13-32
Need to back up security definitions	13-33
Backing up security definitions	13-34
<code>dmpmqaut</code> command	13-35
Backing up security definitions in IBM MQ Explorer	13-36
Unit summary	13-37
Review questions	13-38
Review answers	13-39
Exercise: Using a media image to restore a queue	13-40
Exercise introduction	13-41
Exercise: Backing up and restoring IBM MQ object definitions	13-42
Exercise introduction	13-43
Unit 14. High availability	14-1
Unit objectives	14-2
Topics	14-3
14.1. HA overview for IBM MQ	14-4
HA overview for IBM MQ	14-5
Messaging system availability in IBM MQ	14-6
Application client connectivity (redundancy)	14-7
Application client connectivity (redundancy)	14-8
Client Channel Definition Tables (CCDT)	14-9
IBM MQ Cluster Routing (routing)	14-10
Message High Availability	14-11
IBM MQ Distributed HA solutions	14-12
System-managed HA	14-13
Microsoft Cluster Service (MSCS)	14-14
IBM MQ Appliance	14-15
Replicated Data Queue Managers (RDQM)	14-16
14.2. Multi-instance Queue Managers	14-17
Multi-instance Queue Managers	14-18
Multi-instance queue managers	14-19

Creating multi-instance queue managers	14-20
Standby configuration	14-21
Standby failover	14-22
Handling multiple IP addresses in multi-instance	14-23
Multi-instance queue manager administration	14-24
Managing multi-instance queue manager in IBM MQ Explorer	14-25
Unit summary	14-26
Unit 15. Monitoring and configuring IBM MQ for performance	15-1
Unit objectives	15-2
Topics	15-3
IBM MQ statistics	15-4
15.1. IBM MQ statistics	15-5
Collecting statistics and accounting data	15-6
Subscribing to statistics data (1 of 2)	15-7
Subscribing to statistics data (2 of 2)	15-8
Benefits of subscribing to statistics topics	15-9
IBM MQ statistics that are published to topics only	15-10
Subscribing to statistics with the amqsrua program	15-12
Example: Using the amqsrua sample program	15-13
Queue manager statistics monitoring properties	15-14
MQI statistics	15-15
Queue statistics	15-16
Controlling queue statistics	15-17
Channel statistics	15-18
Controlling channel statistics	15-19
15.2. IBM MQ accounting data	15-20
IBM MQ accounting data	15-21
IBM MQ accounting data collection	15-22
Controlling queue manager accounting	15-24
Controlling queue accounting	15-25
15.3. Displaying statistics and accounting data	15-26
Displaying statistics and accounting data	15-27
Displaying statistics and accounting data	15-28
Examples of accounting output	15-29
Example of statistics data	15-30
15.4. Real-time monitoring	15-31
Real-time monitoring	15-32
Real-time monitoring	15-33
Controlling queue manager real-time monitoring	15-34
Controlling channel and queue real-time monitoring	15-35
Displaying queue and channel monitoring data	15-36
15.5. Configuring and tuning IBM MQ for performance	15-37
Configuring and tuning IBM MQ for performance	15-38
Configuring and tuning IBM MQ for performance	15-39
Queue manager logs	15-40
Default log settings in IBM MQ Explorer	15-41
Log type	15-42
Log path	15-43
Log file pages	15-44
Log primary and secondary files	15-45
Log buffer pages	15-46
Log write integrity level	15-47
Queue manager Log settings in qm.ini	15-48
Queue manager channels performance tuning	15-49
Queue manager listeners performance tuning	15-50

15.6. Monitoring with IBM MQ Console	15-51
Monitoring with IBM MQ Console	15-52
Configuring system resource monitoring	15-53
Monitoring resources	15-54
IBM MQ Console roles	15-55
Unit summary	15-56
Review questions	15-57
Checkpoint answers	15-58
Exercise: Monitoring IBM MQ for performance	15-59
Exercise introduction	15-60
Exercise: Monitoring resources with the IBM MQ Console	15-61
Exercise introduction	15-62
Unit 16. Course summary	16-1
Unit objectives	16-2
Course objectives	16-3
Course objectives	16-4
IBM badge	16-5
IBM Professional Certifications	16-6
Other learning resources (1 of 4)	16-7
Other learning resources (2 of 4)	16-8
Other learning resources (3 of 4)	16-9
Other learning resources (4 of 4)	16-10
Unit summary	16-11
Course completion	16-12
Appendix 17. List of abbreviations	17-1

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®

First Failure Support
Technology™

PureApplication®

Redbooks®

WebSphere®

z/OS®

Intel, Intel Xeon and Xeon are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

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

Windows is a trademark 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 is a registered trademark or trademark of VMware, Inc. or its subsidiaries in the United States and/or other jurisdictions.

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

Course description

IBM MQ V9.1 System Administration

Duration: 5 days

Purpose

This course teaches you how to customize, operate, administer, and monitor IBM MQ on-premises on distributed operating systems. The course covers configuration, day-to-day administration, problem recovery, security management, and performance monitoring. In addition to the instructor-led lectures, the hands-on exercises provide practical experience with distributed queuing, working with MQ clients, and implementing clusters, publish/subscribe messaging. You also learn how to implement authorization, authentication, and encryption, and you learn how to monitor performance.

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.

Prerequisites

- Basic knowledge of IBM MQ concepts and features
- Some knowledge of TCP/IP configuration
- Basic experience with Windows 2016 system administration

Objectives

- Describe the IBM MQ deployment options
- Create and manage queue managers, queues, and channels
- Use the IBM MQ sample programs and utilities to test the IBM MQ network
- Configure distributed queuing
- Configure MQ client connections to a queue manager
- Define and administer a queue manager cluster
- Administer Java Message Service (JMS) in MQ
- 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
- Manage IBM MQ security

- Monitor the activities and performance of an IBM MQ system

Agenda

**Note**

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

Day 1

- (00:15) Course introduction
- (01:30) Unit 1. Introducing IBM MQ
- (01:00) Exercise 1. Getting started with IBM MQ
- (01:30) Unit 2. Working with IBM MQ administration tools
- (01:00) Exercise 2. Working with IBM MQ administration tools
- (01:30) Unit 3. Configuring distributed queuing
- (00:30) Exercise 3. Implementing distributed queuing

Day 2

- (01:30) Unit 4. Managing clients and client connections
- (01:00) Exercise 4. Connecting an IBM MQ client
- (01:00) Unit 5. Advanced IBM MQ client features
- (01:30) Unit 6. Working with queue manager clusters
- (02:00) Exercise 5. Implementing a basic cluster

Day 3

- (02:00) Unit 7. Publish/subscribe messaging
- (01:00) Exercise 6. Configuring publish/subscribe message queuing
- (01:30) Unit 8. Implementing basic security in IBM MQ
- (01:00) Exercise 7. Controlling access to IBM MQ
- (01:30) Unit 9. Securing IBM MQ channels with TLS
- (00:30) Exercise 8. Securing channels with TLS

Day 4

- (01:00) Exercise 8. Securing channels with TLS
- (01:30) Unit 10. Authenticating channels and connections
- (01:30) Exercise 9. Implementing connection authentication
- (00:30) Unit 11. Supporting JMS with IBM MQ
- (01:00) Unit 12. Diagnosing problems
- (00:30) Exercise 10. Running an IBM MQ trace
- (00:30) Unit 13. Backing up and restoring IBM MQ messages and object definitions

Day 5

- (00:30) Unit 13. Backing up and restoring IBM MQ messages and object definitions
- (00:30) Exercise 12. Using a media image to restore a queue
- (00:30) Exercise 11. Backing up and restoring IBM MQ object definitions
- (01:00) Unit 14. High availability
- (01:30) Unit 15. Monitoring and configuring IBM MQ for performance
- (01:30) Exercise 13. Monitoring IBM MQ for performance
- (01:00) Exercise 14. Monitoring resources with the IBM MQ Console
- (00:30) Unit 16. Course summary

Unit 1. Introducing IBM MQ

Estimated time

01:30

Overview

This unit describes IBM MQ basic concepts and components that are applicable to the administrator role. It also describes installation and deployment options for IBM MQ on-premises and in the Cloud.

How you will check your progress

- Review questions
- Lab exercise

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 a course corrections document is available, it is found here.
5. If you want to download an attachment, such as a course corrections document, click the **Attachments** tab at the bottom of the page.

Comments (0) Versions (1) **Attachments (1)** About
6. To save the file to your computer, click the document link and follow the prompts.

Introducing IBM MQ

© Copyright IBM Corporation 2020

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

Unit objectives

- Describe IBM MQ features
- Identify the IBM MQ components and their functions
- Describe the Administrator role and tools
- Outline IBM MQ installation options
- Configure IBM MQ Console

Introducing IBM MQ

© Copyright IBM Corporation 2020

Figure 1-2. Unit objectives

Topics

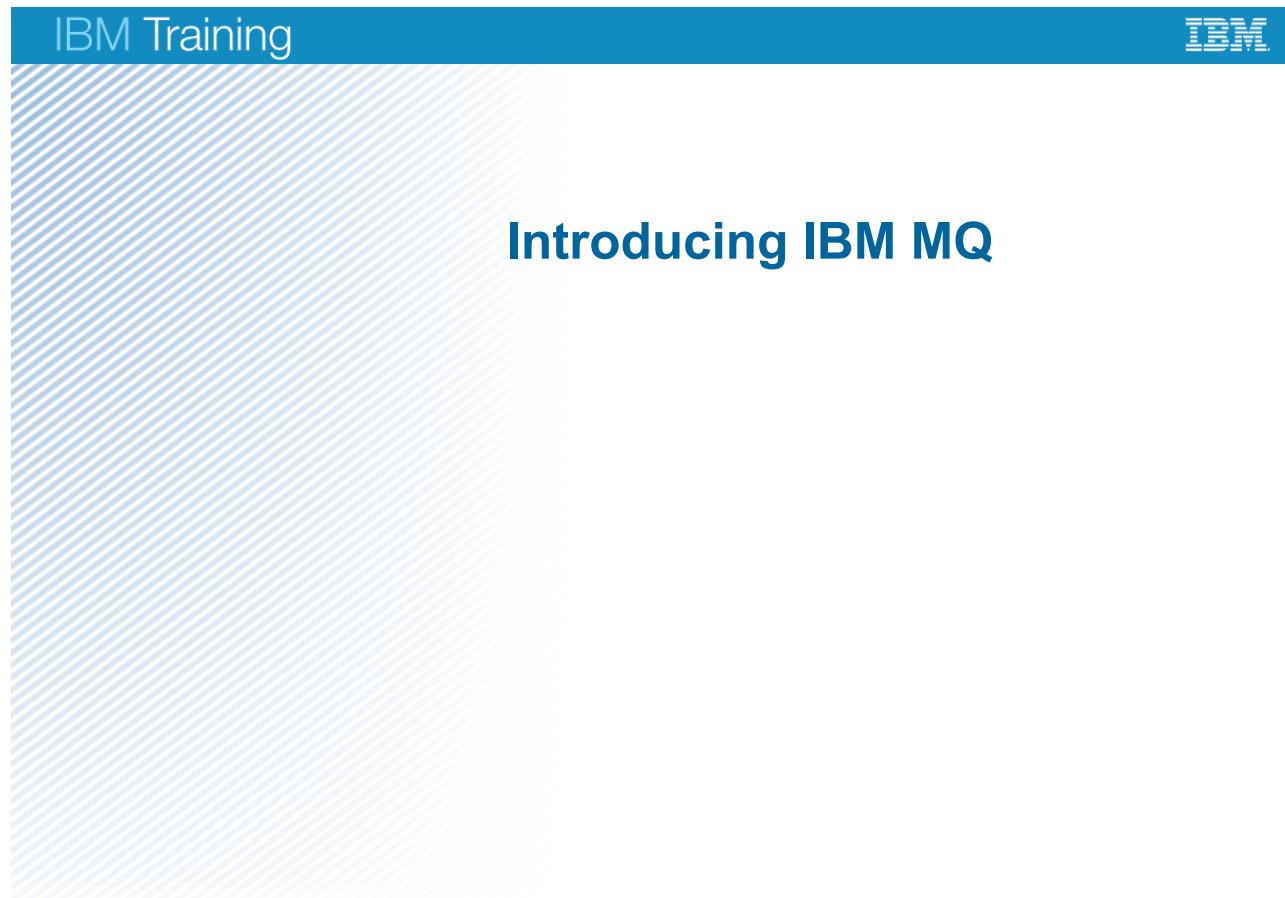
- Introducing IBM MQ
- IBM MQ components and features
- What forms of IBM MQ are available
- Administrator role and tools
- Installing IBM MQ

Introducing IBM MQ

© Copyright IBM Corporation 2020

Figure 1-3. Topics

1.1. Introducing IBM MQ



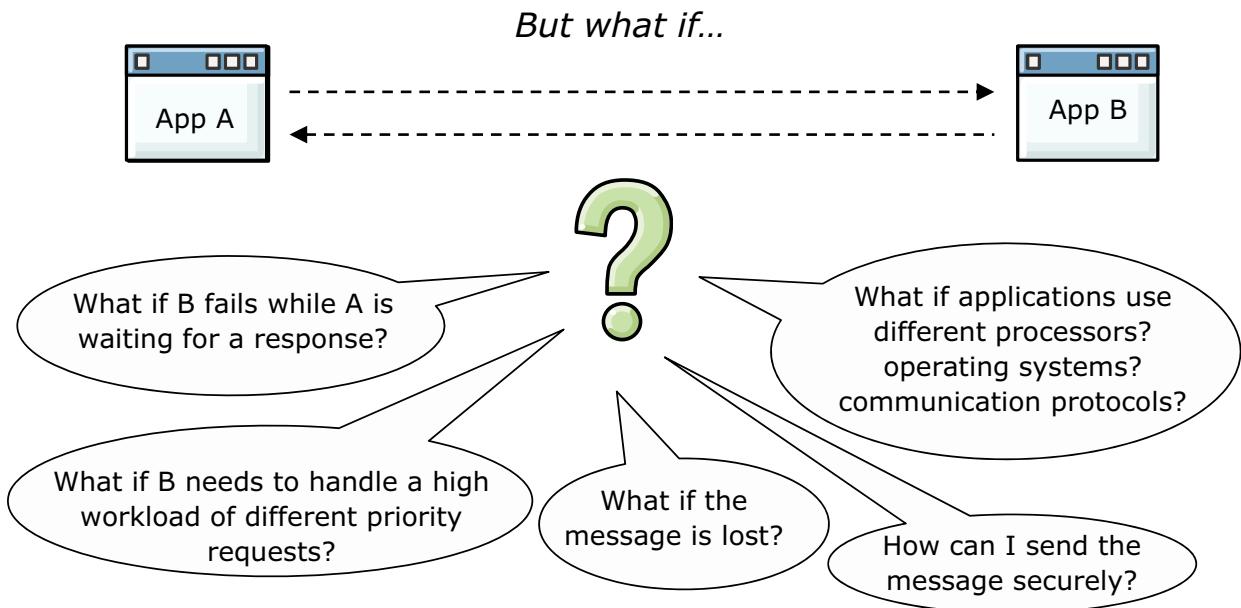
Introducing IBM MQ

© Copyright IBM Corporation 2020

Figure 1-4. Introducing IBM MQ

Problem: How can independent programs communicate?

Applications can communicate by sending each other messages



Introducing IBM MQ

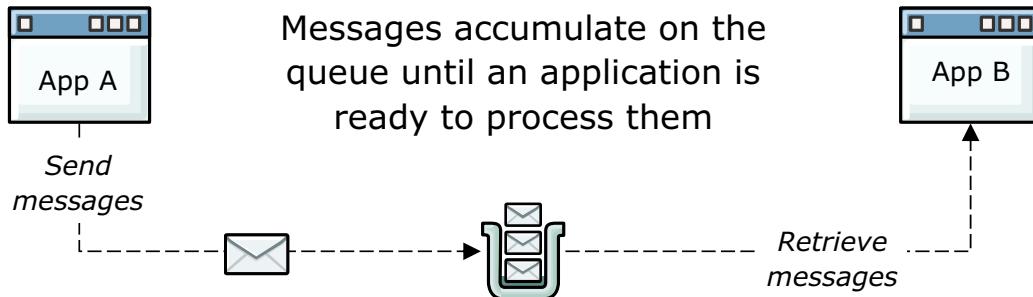
© Copyright IBM Corporation 2020

Figure 1-5. Problem: How can independent programs communicate?

Applications can communicate by sending each other messages. But what if the applications run on different processors, or different operating systems, or use different communication protocols?

As systems become more tightly coupled, their reliance on each other increases. The cost of a failure of a process increases. Scaling systems independently to respond to requirements becomes unmanageable.

Solution: Message queuing



- A can continue working without waiting for B
- B can process the queue when it is ready
- Availability of A and B does not affect messaging

[Introducing IBM MQ](#)

© Copyright IBM Corporation 2020

Figure 1-6. Solution: Message queuing

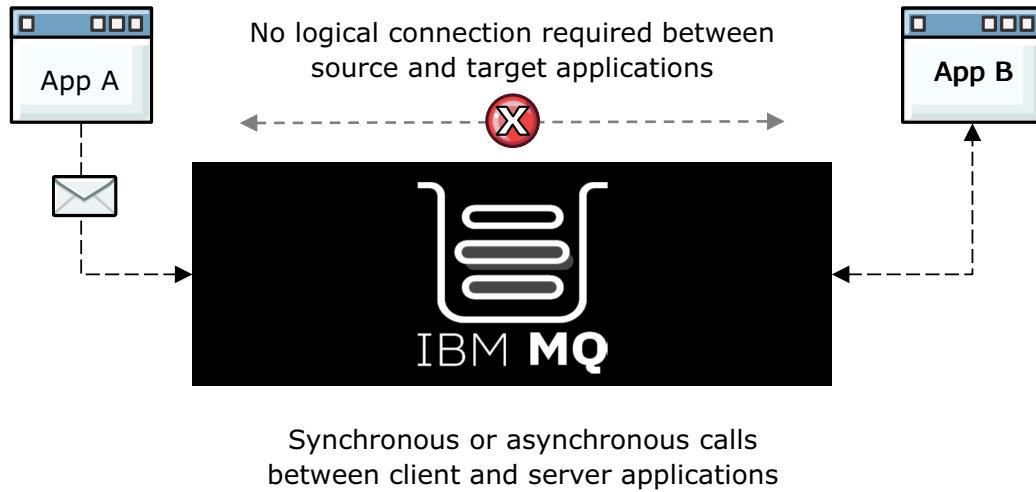
Message queuing allows applications to communicate indirectly through queues, which means that they can continue to work together even when they aren't available at the same time.

Applications that are designed and written with this interface are known as message queuing applications.

- Messaging means that programs communicate by sending each other data in messages rather than calling each other directly.
- Queuing means that messages are stored on queues. Programs can communicate while running independently of each other, at different speeds and times, in different locations, and without having a logical connection between them.

What is IBM MQ? (1 of 2)

- Messaging middleware that enables independent applications to send messages to each other



Introducing IBM MQ

© Copyright IBM Corporation 2020

Figure 1-7. What is IBM MQ? (1 of 2)

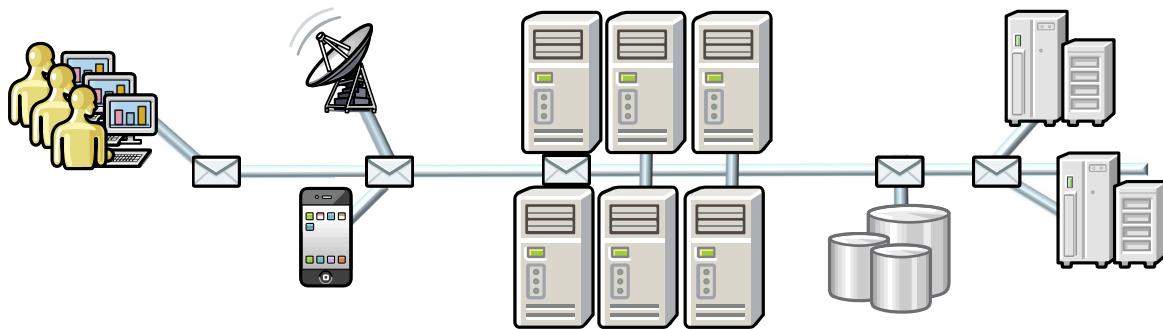
IBM MQ is like email for applications. It sends messages across networks of diverse components. Your application connects to IBM MQ to send or receive a message.

IBM MQ connects applications and services together with valuable qualities-of-service. Applications can exchange information without tying themselves up—just like email where people communicate asynchronously.

IBM MQ handles the different processors, operating systems, subsystems, and communication protocols it encounters in transferring the message. If a connection or a processor is temporarily unavailable, IBM MQ queues the message until the connection is back online and it receives a request to deliver the message.

What is IBM MQ? (2 of 2)

- IBM MQ supports the exchange of information between applications, systems, services, and files by sending and receiving message data via messaging queues



- Connects even the simplest applications to the most complex business processes

Introducing IBM MQ

© Copyright IBM Corporation 2020

Figure 1-8. What is IBM MQ? (2 of 2)

IBM MQ products enable programs to communicate with one another across a network of unlike components (processors, operating systems, subsystems, and communication protocols) by using a consistent application programming interface.

IBM MQ connects virtually any commercial IT system, enabling communication across a broad range of computing platforms, application environments and APIs, and communications protocols for security-rich message delivery.

IBM MQ enterprise messaging features

Reliability	Scalability	Ubiquity	Security
Assured message delivery: “Once and once only” Resiliency and high availability of the infrastructure Continued support and interoperability of systems for over 20 years	High-performance solution Incremental growth of applications and infrastructure	Breadth of support for platforms and environments Multiple application environments and APIs to suit many styles	Data encryption and integrity User authentication and authorization Audit trails for configuration and data flows

Introducing IBM MQ

© Copyright IBM Corporation 2020

Figure 1-9. IBM MQ enterprise messaging features

Reliability

- Other messaging systems use **at-least-once** or **at-most-once** delivery, which can mean *duplicate* or *lost* messages.
- By combining persistent messages with transactions, MQ provides **once-and-once-only** delivery of messages from an application’s point of view.

Scalability

- IBM MQ has proven scalability that allows you to grow your adoption as and when your business requires
- Many customers process hundreds of messages a second and hundreds of millions for messages a day
- No other messaging-oriented middleware product supports as many commercial IT systems as IBM MQ
- Clustering transparently allows dynamic work load distribution and failover without altering your IBM MQ applications
- IBM MQ is designed to leverage threading models on different operating systems

Security

- Rich support for security includes:
 - Authentication of applications when they use a queue manager
 - Authorization checks when they use resources such as a queue on the queue manager
 - Encryption of message data as it travels over the network and as it resides on queues

Benefits of message queuing

- Transports any type of data as messages, enabling business to build flexible, reusable architectures
- Supports 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
- 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

1.2. IBM MQ components and features

IBM MQ components and features

Introducing IBM MQ

© Copyright IBM Corporation 2020

Figure 1-11. IBM MQ components and features

Anatomy of an IBM MQ system

Applications use **IBM MQ clients** to connect to an IBM MQ queue manager, either on the same system or remotely over a network

Channels provide a communication path between queue managers, and between IBM MQ clients and queue managers

Messages are chunks of data that are used to transfer information from one application to another

Queue managers are runtimes that host messaging resources, such as queues and messages

Queues hold messages

Introducing IBM MQ

© Copyright IBM Corporation 2020

Figure 1-12. Anatomy of an IBM MQ system

Messages contain application data

Queues hold messages

Channels connect queue managers and client applications. **Queue manager** manages messages, queues, channels, and other IBM MQ services and resources

What is a message queue?

A message queue is a named destination to which messages can be sent. Messages accumulate on queues until they are retrieved by programs that service those queues.

What is a queue?

A **queue** is a data structure that is used to store messages.

Each queue is owned by a **queue manager**. The queue manager is responsible for maintaining the queues that it owns, and for storing all the messages it receives onto the appropriate queues. The messages might be put on the queue by application programs, or by a queue manager as part of its normal operation.

What is a message?

A *message* is a string of bytes that is meaningful to the applications that use it. Messages are used to transfer information from one application program to another (or between different parts of the same application). The applications can be running on the same platform, or on different platforms.

https://www.ibm.com/support/knowledgecenter/en/SSFKSJ_9.1.0/com.ibm.mq.pro.doc/q002650_.htm

Retrieving messages from queues

Suitably authorized applications can retrieve messages from a queue according to the following retrieval algorithms: First-in-first-out (FIFO).

Message priority, as defined in the message descriptor. Messages that have the same priority are retrieved on a FIFO basis.

A program request for a specific message.

What is a channel?

Channels are logical communication links:

- Between distributed queue managers
- Between an IBM MQ MQI client and an IBM MQ server
- Between two IBM MQ servers

Channels are objects that provide a communication path from one queue manager to another. Channels are used in distributed queuing to move messages from one queue manager to another and they shield applications from the underlying communications protocols. The queue managers might exist on the same, or different, platforms.

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

Message descriptor (MQMD)	Additional headers (optional)	Message data (Payload)
---------------------------	-------------------------------	------------------------

- Persistent messages
 - Stored to disk
 - Queued messages are recovered following a server failure
 - Recovered from logged data after a queue manager restart if disks are intact
- Non-persistent messages
 - Stored in system memory as much as possible (better performance)
 - Queued messages are lost when a server fails or restarts
 - Discarded when a queue manager stops for any reason

Introducing IBM MQ

© Copyright IBM Corporation 2020

Figure 1-13. Messages

Messages are just chunks of data. Applications build messages to send and receive.

Message persistence

- Persistent messages are stored to disk. If the disks are intact, the messages will be also, regardless of the failure
 - Use persistent messages for critical business data that must be reliably maintained and not lost in a failure
- Non-persistent messages are discarded when a queue manager stops for any reason
 - Use non-persistent messages where loss of data is not crucial and in cases where performance is considered more important than data integrity
- Applications can specify message persistence in MQMD

Queues



- Defined endpoint destination for messages
- Types of queues

Local	QLOCAL		Owned by queue manager to which the application is connected. You can get messages from and put messages on local queues.
Remote	QREMOTE		Defined as a local queue on one queue manager, but redirects messages to a local queue on a different queue manager. An application does not need to know whether a queue is local or remote.
Alias	QALIAS		A pointer to a local queue or a locally owned remote queue
Model	QMODEL		A template to create a dynamic local queue

Introducing IBM MQ

© Copyright IBM Corporation 2020

Figure 1-14. Queues

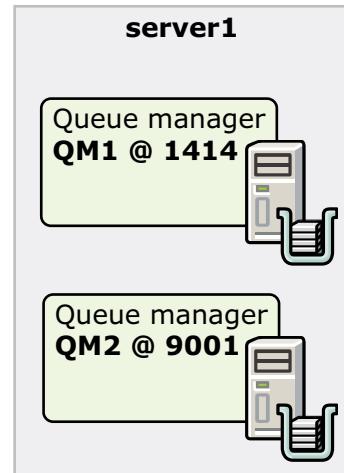
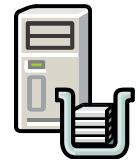
A message queue is a named destination to which messages can be sent. Messages accumulate on queues until they are retrieved by programs that service those queues.

Queue managers

- Provide queuing services to applications and manage the queues that belong to them
 - Each queue belongs to a single queue manager

- Ensure:
 - Messages are put on the correct queue
 - Object attributes are changed according to the commands received
 - Special events such as trigger events or instrumentation events are generated when the appropriate conditions are met

- Multiple queue managers can be hosted on the same machine
 - Queue managers that share a server require different names and listener ports



[Introducing IBM MQ](#)

© Copyright IBM Corporation 2020

Figure 1-15. Queue managers

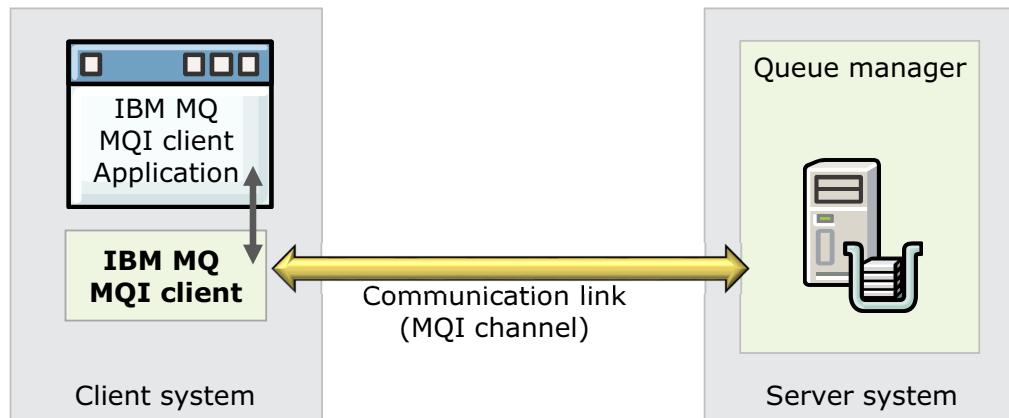
Queues reside in, and are managed by, a queue manager.

Programs access queues through the external services of the queue manager to do these tasks:

- Open a queue
- Put messages on the queue
- Get messages from the queue
- Close the queue
- Set or query the attributes of queues

IBM MQ clients

- Implement messaging without full IBM MQ implementation on client machines
 - Reduces hardware requirements on the client system
 - Reduces system administration requirements
- Application uses message queue interface (MQI) calls to server
- Wide platform support



Introducing IBM MQ

© Copyright IBM Corporation 2020

Figure 1-16. IBM MQ clients

An IBM MQ MQI client is a component of the IBM MQ product that can be installed on a system on which no queue manager runs. IBM MQ clients are an efficient way of implementing IBM MQ messaging and queuing without the need for a full IBM MQ implementation on the client machine.

Using an IBM MQ MQI client, an application running on the same system as the client can connect to a queue manager that is running on another system. The application can issue MQI calls to that queue manager. Such an application is called an IBM MQ MQI client application and the queue manager is called a *server queue manager*.

An IBM MQ MQI client application and a server queue manager communicate with each other by using a bidirectional *MQI channel*. The input parameters of an MQI call flow in one direction on an MQI channel, and the output parameters flow in the opposite direction.

https://www.ibm.com/support/knowledgecenter/SSFKSJ_9.1.0/com.ibm.mq.pro.doc/q003460_.htm

Channels

- Configurable processes that send or receive messages
- Queue manager to queue manager
 - Unidirectional
 - Defined in pairs: 1 sender, 1 receiver
 - Asynchronous
 - Message channel agents (MCA) control the sending and receiving of messages between queue managers



- Client MQI channel
 - Bidirectional
 - Defined as a single channel
 - Synchronous



[Introducing IBM MQ](#)

© Copyright IBM Corporation 2020

Figure 1-17. Channels

What are channels?

Message channel agents (MCA) control the sending and receiving of messages between queue managers. Using channels requires transmission queues and remote queues.

For client connections, channels connect an application (MQI client) to a queue manager.

1.3. What forms of IBM MQ are available

What forms of IBM MQ are available

Introducing IBM MQ

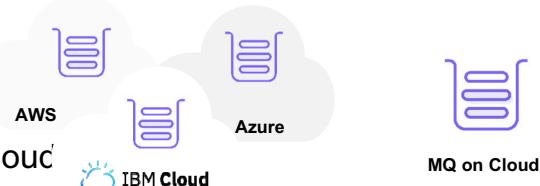
© Copyright IBM Corporation 2020

Figure 1-18. What forms of IBM MQ are available

Deployment models

- IBM MQ and IBM MQ Advanced on-premises software
 - Unix, Linux, Windows, and IBMi

- IBM MQ on Cloud
 - Any public or private cloud
 - Managed IBM MQ service on IBM Cloud
 - IBM MQ in containers



- IBM MQ Appliance
 - Physical appliance that runs IBM MQ Advanced firmware
- IBM MQ for z/OS



Introducing IBM MQ

© Copyright IBM Corporation 2020

Figure 1-19. Deployment models

You have many options for deploying your IBM MQ solutions. IBM MQ is flexible, and can be run as software, on an appliance, or on the cloud.

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

- Managed File Transfer
- Advanced Message Security
- MQ Telemetry Transport

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

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

IBM MQ also runs on the z/OS mainframe.

IBM MQ Advanced

- Advanced entitlement adds the following features:
 - Managed File Transfer
 - Advanced Message Security
 - IBM MQ Telemetry Transport
 - Replicated Data Queue Manager
 - Bridge to blockchain
 - IBM MQ Advanced certified container

Introducing IBM MQ

© Copyright IBM Corporation 2020

Figure 1-20. IBM MQ Advanced

IBM MQ on Cloud managed service

- Available in both IBM Cloud and Amazon Web Services (AWS)
- Removes much of the responsibility of running and maintaining an IBM MQ environment, including
 - Operating System Patching
 - IBM MQ patching
 - IBM MQ upgrades
 - Availability
 - Failover

Figure 1-21. *IBM MQ on Cloud managed service*

IBM MQ in containers

- Deploy a production-ready IBM MQ image into
 - Red Hat OpenShift
 - IBM Cloud Private
 - IBM Cloud Kubernetes Service
- Three options:
 - Use the pre-packaged IBM MQ Advanced certified container
 - Use the pre-packaged container in IBM Cloud Pak for Integration (a separate IBM product)
 - Develop a self-built container (formerly referred to as the "Docker container image")

Figure 1-22. IBM MQ in containers

1.4. Administrative role and tools

Administrative role and tools

Introducing IBM MQ

© Copyright IBM Corporation 2020

Figure 1-23. Administrative role and tools

IBM MQ administrative tasks

- Install, apply maintenance, and configure:
 - IBM MQ
 - IBM MQ Managed File Transfer
 - IBM MQ Advanced Message Security
 - IBM MQ Telemetry
- Create queue managers
- Create queues, clusters, channels, or any objects that users request
- Administer publish/subscribe topologies
- Administer security and SSL keystores and truststores
- Back up file systems and object definitions
- Monitor disk space
- Identify problems
- Participate in infrastructure and capacity planning sessions
- Establish naming standards
- Analyze performance

Introducing IBM MQ

© Copyright IBM Corporation 2020

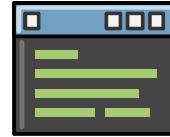
Figure 1-24. IBM MQ administrative tasks

As an administrator, must have administrative ID with IBM MQ privileges. You are responsible for the installation, configuration, and maintenance of IBM MQ. You create and monitor the welfare of the queue managers, and other IBM MQ objects.

Some of the main administrator tasks are listed on this slide.

IBM MQ administration interfaces

- IBM MQ Explorer
 - Eclipse interface
- IBM MQ script commands (MQSC)
- IBM MQ programmable commands (PCF)
- IBM MQ Console
 - Web-based interface
 - Simple way to access, create, and manage MQ resources
- REST API
 - Administrative APIs over HTTPS



Introducing IBM MQ

© Copyright IBM Corporation 2020

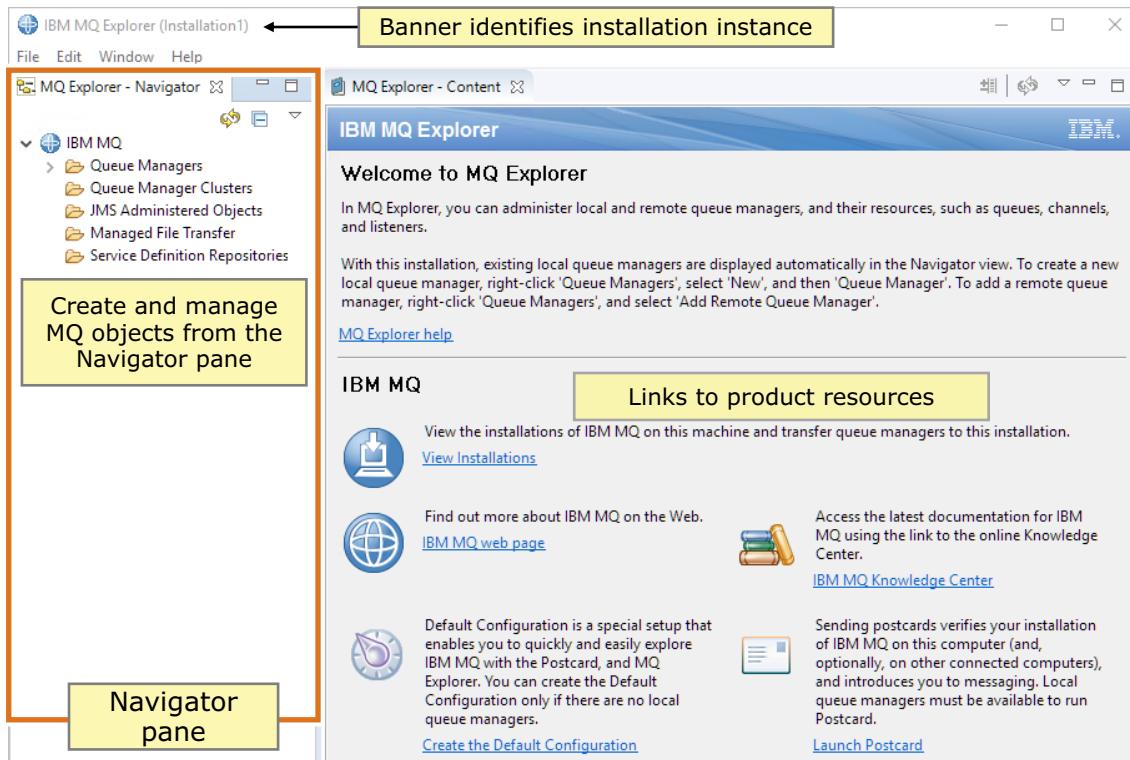
Figure 1-25. IBM MQ administration interfaces

IBM MQ offers several options for administration. Each offers equivalent options to perform common actions on IBM MQ objects. Which option you use depends on what is most appropriate for the task at hand and your personal preference. For example, when creating many objects that might be associated with a new application, you could enter these definitions in a file and define them by running an *IBM MQ script command* with the *runmqsc* utility. In the lab exercises for this course, you use the *runmqsc* utility.

During this course, you work with each of these tools.



Working with IBM MQ Explorer



Introducing IBM MQ

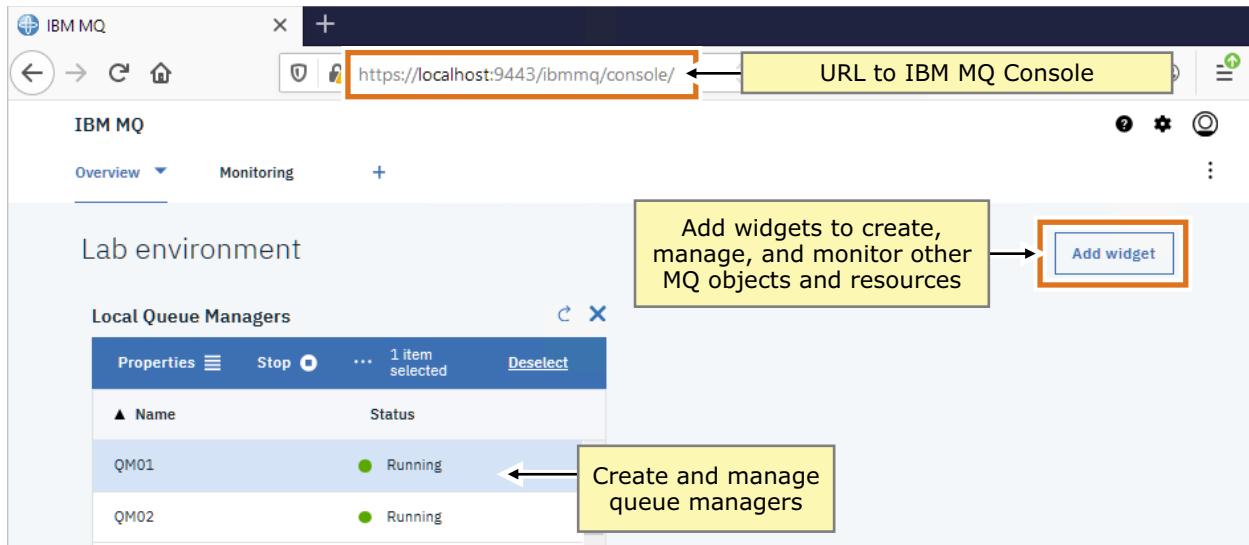
© Copyright IBM Corporation 2020

Figure 1-26. Working with IBM MQ Explorer

During this course, you use MQ Explorer for most administrative functions, such as creating queue managers, queues, and channels.

IBM Training

Working with IBM MQ Console



Introducing IBM MQ

© Copyright IBM Corporation 2020

Figure 1-27. Working with IBM MQ Console

During this course, you also use IBM MQ Console to manage IBM MQ objects.

1.5. Installing IBM MQ

Installing IBM MQ

Introducing IBM MQ

© Copyright IBM Corporation 2020

Figure 1-28. Installing IBM MQ

Long Term Support and Continuous Delivery

- Long-Term Support (LTS)
 - Provides maximum stability over the life of each major version
 - New functions are added only at major version boundaries
 - During the life of each major version, only software fixes and security updates are applied
- Continuous Delivery (CD)
 - Provides earliest access to new capabilities of IBM MQ
 - New functions are added continuously, along with software fixes and security updates
- LTS and CD releases are realigned at the first release of each major version
 - New functions that were applied incrementally to the CD release, are all applied to the new LTS version
- See *IBM MQ FAQ for Long-Term Support and Continuous Delivery releases*: <http://www.ibm.com/support/docview.wss?uid=swg27047919>

Introducing IBM MQ

© Copyright IBM Corporation 2020

Figure 1-29. Long Term Support and Continuous Delivery

The two IBM MQ release versions for IBM MQ 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 IBM MQ.

Multiple version installations

- On all platforms except IBM i, you can have more than one installation of IBM MQ on a system
 - Each installation has a unique name that associates queue managers and configuration files with an installation
 - On Windows, you choose the installation name during installation
 - On UNIX and Linux
 - The first IBM MQ installation is named `Installation1`
 - You use the `crtmqinst` command for subsequent installation names before installing the product
- Optional. During or after installation, define the primary installation (and the level of IBM MQ) that IBM MQ system-wide locations refer to
 - Example: When migrating to a new IBM MQ version
 - On UNIX, Linux and Windows, use the `setmqinst` command
 - On z/OS, use the `STEPLIB` command to control the level of IBM MQ that is used

Introducing IBM MQ

© Copyright IBM Corporation 2020

Figure 1-30. Multiple version installations

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.

The installation name cannot be changed after IBM MQ is installed.

Choosing an installation path

- Default installation paths

AIX	/usr/mqm
IBM i	/QIBM/ProdData/mqm
Linux	/opt/mqm
Solaris	/opt/mqm
Windows (software)	C:\Program Files\IBM\MQ
Windows (data files and logs)	C:\ProgramData\IBM\MQ

- Custom installation paths

- On IBM i, you can install only in the default location
- For the other platforms, you can install to a custom location
- See the IBM Knowledge Center for installation path restrictions, as described in *Installation location on Multiplatforms*:
https://www.ibm.com/support/knowledgecenter/SSFKSJ_9.1.0/com.ibm.mq.ins.doc/q008310.htm

Figure 1-31. Choosing an installation path

You can install IBM 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.

Choosing what to install

- Select components and features that you require when you install IBM MQ, for example:
 - 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 and entitlement

[Introducing IBM MQ](#)

© Copyright IBM Corporation 2020

Figure 1-32. Choosing what to install

When you install IBM MQ on premises, you choose which IBM MQ components you are going to install. Is this IBM MQ installation for a developer or for production? Does this installation require IBM MQ Managed File Transfer or an IBM MQ Telemetry server?

IBM 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.

Review your installation

- After installation, run the `dspmqver` command to view the DataPath and InstPath
 - Example: On Windows, DataPath is set to C:\ProgramData

```

Select Administrator: Command Prompt
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

C:\Users\Administrator>dspmqver
Name: IBM MQ
Version: 9.1.0.1
Level: p910-001-181108
BuildType: IKAP - (Production)
Platform: IBM MQ for Windows (x64 platform)
Mode: 64-bit
O/S: Windows Server 2016 Server Standard Edition, Build 14393
InstName: Installation1
InstDesc:
InstPath: C:\Program Files\IBM\MQ
DataPath: C:\ProgramData\IBM\MQ
LicenseType: Production
  
```

This PC > Local Disk (C:) >

Name
labfiles
PerfLogs
Program Files
Program Files (v86)
ProgramData
temp
Users
Windows

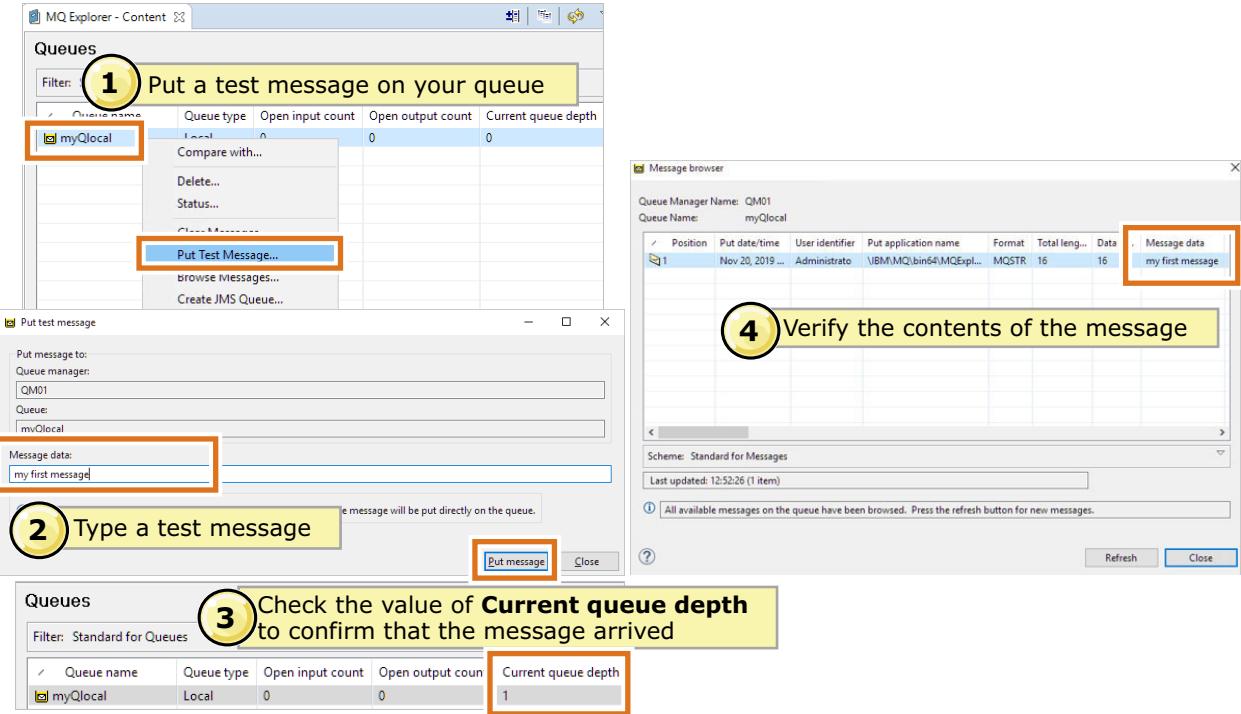
Make sure that the **ProgramData** folder is not hidden

Figure 1-33. Review your installation



Validate your installation with MQ Explorer

- Create a queue manager and a queue to test sending a message



Introducing IBM MQ

© Copyright IBM Corporation 2020

Figure 1-34. Validate your installation with MQ Explorer

You can also use IBM MQ Console or IBM MQ script commands (MQSC) and IBM MQ control commands to do these tasks.

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

Introducing IBM MQ

© Copyright IBM Corporation 2020

Figure 1-35. Uninstalling IBM MQ

If you need to uninstall IBM 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 IBM MQ components and services.

Finding more information

- System requirements
 - <http://www.ibm.com/support/docview.wss?uid=swg27006467>
- Planned maintenance
 - <https://www.ibm.com/support/pages/ibm-mq-planned-maintenance-release-dates#9>
- Product support
 - 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
 - <https://www.ibm.com/support/pages/ibm-mq-websphere-mq-and-mqseries-product-readmes>
- Product documentation
 - https://www.ibm.com/support/knowledgecenter/SSFKSJ_9.1.0/com.ibm.mq.pro.doc/q001020.htm

Introducing IBM MQ

© Copyright IBM Corporation 2020

Figure 1-36. Finding more information

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

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

=====

YouTube videos:

<https://www.youtube.com/playlist?list=PLzpeuWUENMK1qVc24p14guzd3L30Nv6Fa>

1.6. Configuring the mqweb server for IBM MQ Console and REST administration

Configuring the mqweb server for IBM MQ Console and REST administration

Introducing IBM MQ

© Copyright IBM Corporation 2020

Figure 1-37. Configuring the mqweb server for IBM MQ Console and REST administration

Configuring mqweb server for IBM MQ Console and REST administration

- Configure users and roles
 - By default, the user registry does not contain any users
 - Users configured in
MQDATA_PATH/web/installations/installationName/servers/mqweb/ mqwebuser.xml
- Sample registry files in
MQINSTALLATION_PATH/web/mq/samp/configuration
- Main user roles for IBM MQ Console
 - **MQWebAdmin** (mqadmin)
 - **MQWebAdminRO** (mqreader)
 - **MQWebUser** (mquser)

[Introducing IBM MQ](#)

© Copyright IBM Corporation 2020

Figure 1-38. Configuring mqweb server for IBM MQ Console and REST administration

You must configure basic security to allow users and groups to access the IBM MQ Console.

First, copy one of the sample registry files from the

MQ_INSTALLATION_PATH/web/mq/samp/configuration directory to the IBM MQ Console web server directory for the installation. On UNIX, Linux, and Windows the installation directory is MQ_DATA_DIRECTORY/web/installations/installationName/servers/mqweb. Then, rename the sample XML file to mqwebuser.xml.

The IBM MQ installation includes three sample registry files for the IBM MQ Console.

You can use one of the sample files or customize one of the sample files. Before you start the IBM MQ web server, replace the mqwebuser.xml file in the IBM MQ Console web server installation directory with your registry file.

During the exercise that follows this unit, you set up basic security to access IBM MQ Console, and sign in with the MQWebAdmin role (mqadmin).

Commands to control the mqweb server

- Start mqweb server: `strmqweb`
- Stop mqweb server: `endmqweb`
- Display the status of mqweb server and the URL: `dspmqweb`
 - Example output:

```
Server mqweb is running.  
URLs:  
https://localhost:9443/ibmmq/console
```

Figure 1-39. Commands to control the mqweb server

To start the IBM MQ Console web server, type `strmqweb`. The user that starts the IBM MQ Console web server must be an IBM MQ privileged user. A privileged user is a user with full administrative authorities for IBM MQ.

You can stop the mqweb server by typing `endmqweb`.

The mqweb server must be running to use the IBM MQ Console or the administrative REST API. You can use the `dspmqweb` command to view the status of the mqweb server. If the server is running, then the URLs that are used by the IBM MQ Console and administrative REST API are displayed.



Working with IBM MQ Console

The screenshot shows the IBM MQ Console interface. At the top, there's a navigation bar with tabs for 'Overview' (highlighted with an orange box) and 'Monitoring'. Below this is a dashboard area titled 'Lab environment' containing a 'Local Queue Managers' widget. This widget lists two queue managers: 'QM01' and 'QM02', both marked as 'Running'. To the right of the dashboard, there are several callout boxes with arrows pointing to specific UI elements:

- A yellow box labeled 'URL to IBM MQ Console' points to the URL field in the browser's address bar, which contains 'https://localhost:9443/ibmmq/console/'.
- A yellow box labeled 'Add tabs to the dashboard' points to the '+' button in the top navigation bar.
- A yellow box labeled 'Create and manage queue managers in the **Local Queue Managers** widget' points to the 'Local Queue Managers' section of the dashboard.
- A yellow box labeled 'Add widgets to create, manage, and monitor other WebSphere MQ objects and resources' points to a blue 'Add widget' button located in the top right corner of the dashboard area.

[Introducing IBM MQ](#)

© Copyright IBM Corporation 2020

Figure 1-40. Working with IBM MQ Console

You can use the Local Queue Manager widget to create and manage queue managers and queue manager properties. You can also manage queue managers that are associated with the IBM MQ installation from which the IBM MQ Console is running.

Unit summary

- Describe IBM MQ features
- Identify the IBM MQ components and their functions
- Describe the Administrator role and tools
- Outline IBM MQ installation options
- Configure IBM MQ Console

Introducing IBM MQ

© Copyright IBM Corporation 2020

Figure 1-41. Unit summary

Review questions

1. True or False: IBM MQ supports asynchronous messaging only.
2. IBM MQ provides which of the following features? Select all that apply.
 - a. Security
 - b. Reliability
 - c. Scalability
 - d. All of the above
3. True or False: Only one copy of IBM MQ can be installed on a system at a time.
4. True or False: To use the IBM MQ Console, you must first configure security on the mqweb server to enable user access.



Introducing IBM MQ

© Copyright IBM Corporation 2020

Figure 1-42. Review questions

Write your answers here:

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

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 provides which of the following features? Select all that apply.
 - a. Security
 - b. Reliability
 - c. Scalability
 - d. All of the above**The answer is d.**
3. True or False: Only one copy of IBM MQ can be installed on a system at a time.
The answer is False. On all platforms except IBM i, you can have more than one installation of IBM MQ on a system.
4. True or False: To use the IBM MQ Console, you must first configure security on the mqweb server to enable user access.
The answer is True.

Introducing IBM MQ

© Copyright IBM Corporation 2020

Figure 1-43. Review answers

Exercise: Getting started with IBM MQ

Introducing IBM MQ

© Copyright IBM Corporation 2020

Figure 1-44. Exercise: Getting started with IBM MQ

Exercise introduction

- Explore your IBM MQ installation by creating a queue and testing messaging
- Validate your installation by creating a queue and testing messaging
- Configure the mqweb server for user access to IBM MQ Console



Introducing IBM MQ

© Copyright IBM Corporation 2020

Figure 1-45. Exercise introduction

Unit 2. Working with IBM MQ administration tools

Estimated time

01:30

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
- Lab exercise

Unit objectives

- Create queue managers and objects with IBM MQ control commands and script commands
- Describe queue manager sets
- Stop and delete queue managers

Working with IBM MQ administration tools

© Copyright IBM Corporation 2020

Figure 2-1. Unit objectives

Topics

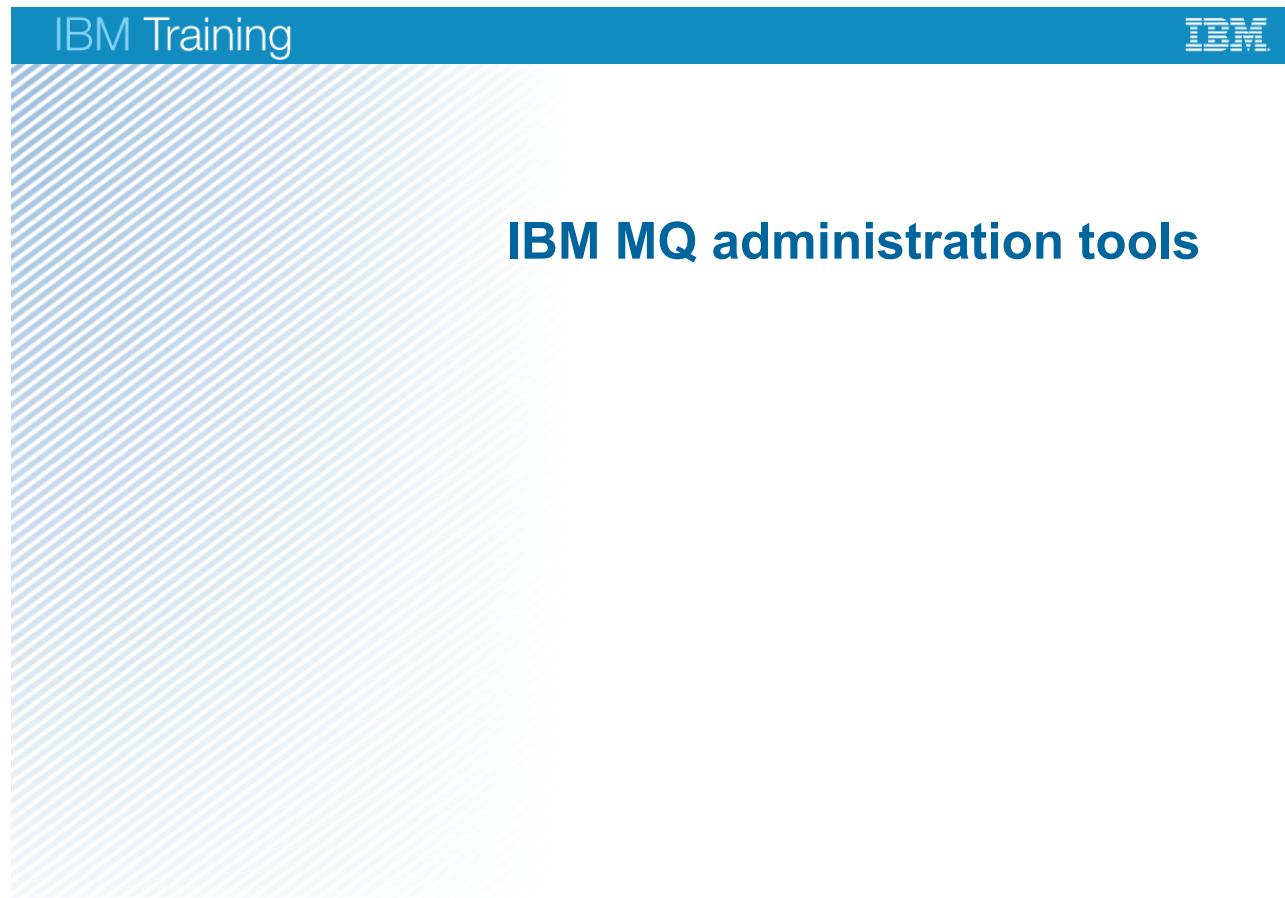
- IBM MQ control commands
- IBM MQ script commands (MQSC)
- Special queues
- Sample programs to test configuration
- Working with queue manager sets

Working with IBM MQ administration tools

© Copyright IBM Corporation 2020

Figure 2-2. Topics

2.1. IBM MQ administration tools



Working with IBM MQ administration tools

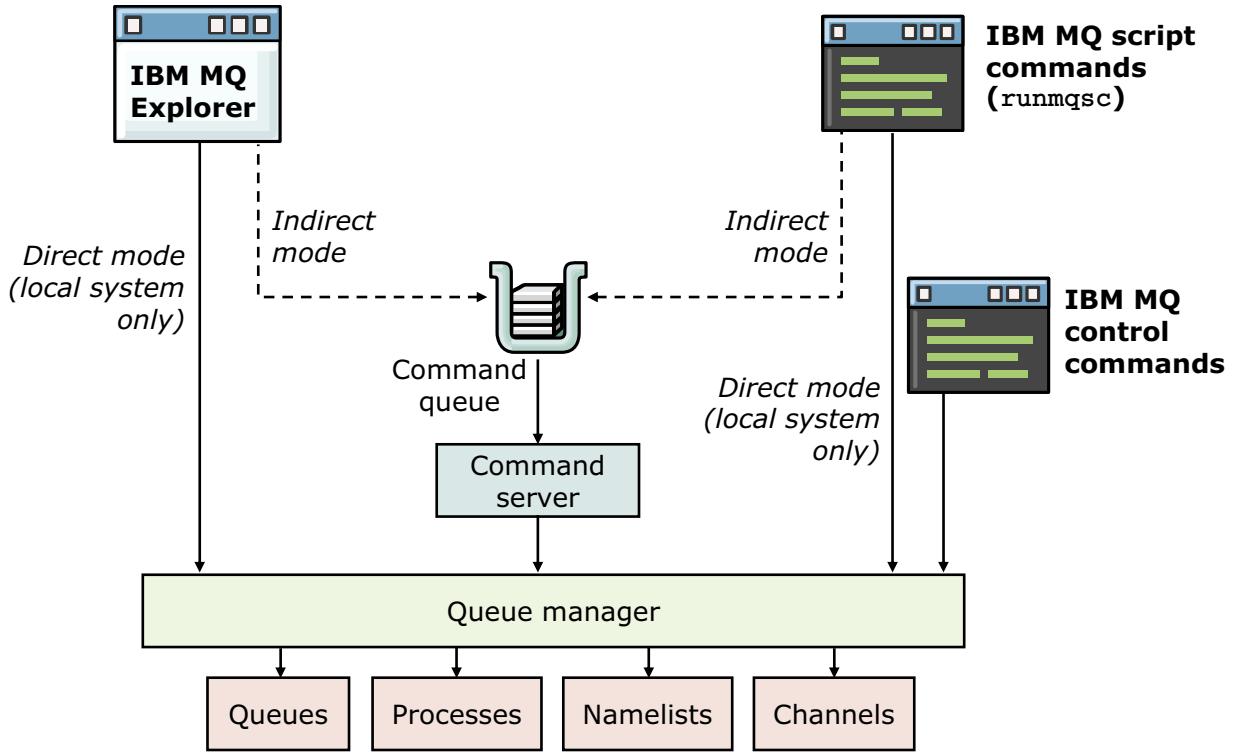
© Copyright IBM Corporation 2020

Figure 2-3. IBM MQ administration tools

IBM Training



Administration interfaces



Working with IBM MQ administration tools

© Copyright IBM Corporation 2020

Figure 2-4. Administration interfaces

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

- IBM MQ Explorer
- IBM MQ script commands
- IBM 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.

2.2. IBM MQ control commands

IBM MQ control commands

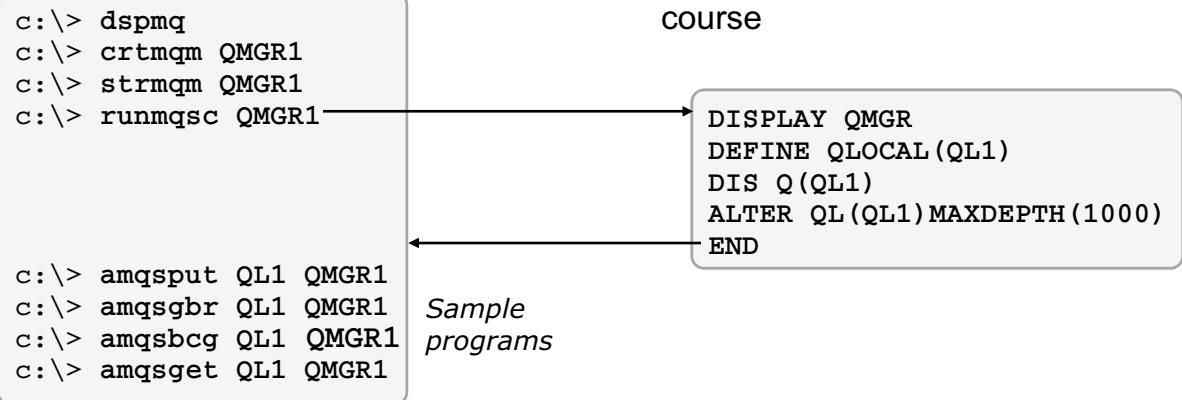
Working with IBM MQ administration tools

© Copyright IBM Corporation 2020

Figure 2-5. IBM MQ control commands

IBM MQ command modes

- Control commands
 - Entered at an operating system command prompt
 - Require authorization
 - Lowercase when referenced in this course
- IBM MQ script commands (MQSC)
 - Entered in MQSC mode (no command prompt)
 - Run against specified queue manager
 - Started by typing `runmqsc` and queue manager name
 - Uppercase when referenced in this course



Working with IBM MQ administration tools

© Copyright IBM Corporation 2020

Figure 2-6. 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, QMR01, on port 1415 that uses a local queue, QMR01.DLQ, as its dead-letter queue

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

Working with IBM MQ administration tools

© Copyright IBM Corporation 2020

Figure 2-7. 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 hostnames as queue manager names when possible. If the hostnames change or frequent server reassignments occur, then the use of hostnames 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 the SYSTEM prefix
- Examples
 - SYSTEM.ADMIN.ACCTING.QUEUE holds accounting monitoring data
 - SYSTEM.DEFAULT.LOCAL.QUEUE defines default attributes for a local queue

Figure 2-8. 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

Figure 2-9. Queue manager logs

IBM 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 IBM 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, IBM IBM 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

[Working with IBM MQ administration tools](#)

© Copyright IBM Corporation 2020

Figure 2-10. Naming IBM MQ objects

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

In general, IBM 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 IBM 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 IBM 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 2-11. 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 2-12. 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 2-13. Stopping and starting the IBM MQ service on Windows

IBM 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 `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 2-14. 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.

When do you 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
- When you might need to edit a configuration file:
 - 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 when you accidentally delete the existing queue manager
 - Your IBM Support Center advises you to do so
- Always create a backup of the configuration file before editing

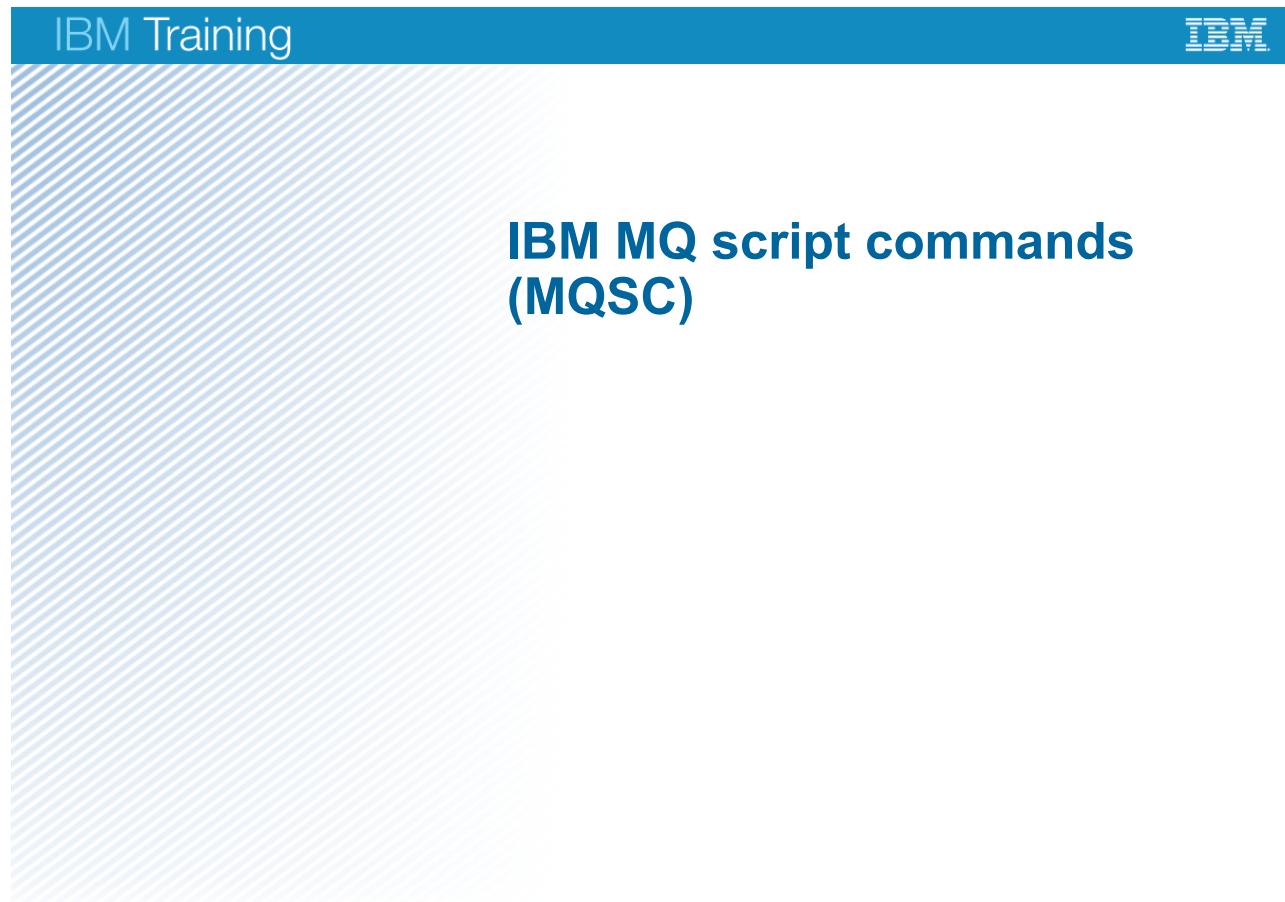
Figure 2-15. When do you edit the queue manager configuration file?

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.

2.3. IBM MQ script commands (MQSC)



Working with IBM MQ administration tools

© Copyright IBM Corporation 2020

Figure 2-16. *IBM MQ script commands (MQSC)*

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

Figure 2-17. 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 IBM 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

[Working with IBM MQ administration tools](#)

© Copyright IBM Corporation 2020

Figure 2-18. 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 IBM 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:\>
```

Figure 2-19. 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 2-20. 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 IBM 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 IBM 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.

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

Working with IBM MQ administration tools

© Copyright IBM Corporation 2020

Figure 2-21. 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 2-22. 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')
```

Working with IBM MQ administration tools

© Copyright IBM Corporation 2020

Figure 2-23. 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 IBM 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 2-24. 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

[Working with IBM MQ administration tools](#)

© Copyright IBM Corporation 2020

Figure 2-25. 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 IBM 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 IBM 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 2-26. 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 2-27. Defining other queue types

IBM 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, you cannot clear or delete the queue

Figure 2-28. 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 2-29. 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**.

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

Figure 2-30. 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.

2.4. Special queues

Special queues

Working with IBM MQ administration tools

© Copyright IBM Corporation 2020

Figure 2-31. Special queues

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

[Working with IBM MQ administration tools](#)

© Copyright IBM Corporation 2020

Figure 2-32. Special queues

Some local queues have special purposes in IBM 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 IBM 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)`

Working with IBM MQ administration tools

© Copyright IBM Corporation 2020

Figure 2-33. 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.

2.5. Sample programs

Sample programs

Working with IBM MQ administration tools

© Copyright IBM Corporation 2020

Figure 2-34. Sample programs

Testing with IBM MQ sample programs

- IBM MQ sample programs
 - Linux: /opt/mqm/samp/bin
 - Windows: C:\Program Files\IBM\MQ\Tools\c\Samples\Bin64

- **amqspput QName QMgrName**
 - Read text from the standard input device and put messages
- **amqsgget QName QMgrName**
 - Get messages and write to the standard output device
- **amqsbcg QName QMgrName**
 - Browse messages and show both application data and message descriptor
- **amqsgbr 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

Figure 2-35. Testing with IBM MQ sample programs

IBM MQ provides many ways to test your MQ configuration. You saw how to put and browse messages with MQ Explorer.

IBM MQ also contains sample programs to put, get, and browse messages on queues. MQ sample programs are available as C source, COBOL source, and C runtime.

This slide describes some of the sample programs.

The Put sample program, **amqspput**, 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, **amqsgget**, 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.

IBM 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 QMGR1
Sample AMQSPUT0 start
Target queue is QL.TEST
This is my test message [Press Enter]
This is another test message [Press Enter]
[Press Enter] ← A blank line indicates the
Sample AMQSPUT0 end end of message input and
ends the program

C:\Users\MQ_ADMIN>amqsgget QL.TEST QMGR1
Sample AMQSGET0 start
message <This is my test message>
message <This is another test message>
C:\Users\MQ_ADMIN>
```

Working with IBM MQ administration tools

© Copyright IBM Corporation 2020

Figure 2-36. Sample program example

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

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

To send messages to the queue with the **amqspput** sample program, you press Enter after each message. You can send multiple messages after the program starts. To end the program, press Enter twice after your last message.

When you run the **amqsgget** program, it returns a list of all the messages that are in the queue, which empties the queue. The **amqsgget** sample program waits for 15 seconds after returning the last message, in case any more messages are written to the queue.

2.6. Working with queue manager sets

Working with queue manager sets

Working with IBM MQ administration tools

© Copyright IBM Corporation 2020

Figure 2-37. Working with queue manager sets

Queue manager sets (1 of 2)

- Useful for administering large numbers of queue managers with MQ Explorer
 - Group queue managers that belong to particular applications, departments, or companies
 - Organize sets according to 'test' and 'production' sets, or based on the operating system of the platform
- Apply actions to all queue managers in the set
 - Show or hide all
 - Connect or disconnect all
 - Start or stop all local
 - Run default or custom tests
- Group queue managers in folders
 - Queue managers can be members of none, one, or many sets
 - Sets cannot contain other sets

Figure 2-38. Queue manager sets (1 of 2)

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.

Queue manager sets (2 of 2)

- Group queue managers *manually* or *automatically* by using filters
 - Manually: Create a set and manually choose which queue managers to include
 - Automatically: Create a filter; all queue managers that match the test in the filter are included in the set
- Sets can be exported and imported with MQ Explorer
- Deleting a set does not delete the queue managers

Working with IBM MQ administration tools

© Copyright IBM Corporation 2020

Figure 2-39. Queue manager sets (2 of 2)

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.

2.7. Stopping and removing a queue manager

Stopping and removing a queue manager

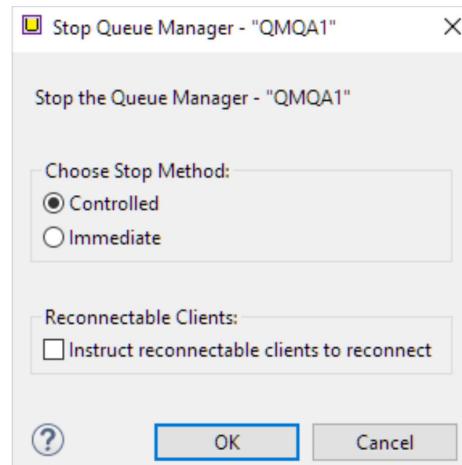
Working with IBM MQ administration tools

© Copyright IBM Corporation 2020

Figure 2-40. Stopping and removing a queue manager

Stopping and deleting queue managers

- You must stop a queue manager before deleting it
- Stop a queue manager
 - Use IBM MQ Console, MQ Explorer, or the `endmqm QMgrName` control command
- Controlled shutdown (`endmqm`)
 - Performs a quiesced shutdown
 - Waits for connected applications to disconnect
- Immediate shutdown
 - Stops queue manager processes without waiting for applications to disconnect
- Manually force shutdown
 - Manually stop processes in Windows Task Manager



[Working with IBM MQ administration tools](#)

© Copyright IBM Corporation 2020

Figure 2-41. Stopping and deleting queue managers

Some system administration tasks require that you stop a queue manager. Use a controlled shutdown, which is equivalent the `endmqm` control command. The controlled shutdown might take a few moments to complete because it waits until all connected applications have disconnected.

In MQ Explorer, you can choose Controlled or Immediate shutdown methods. Immediate shutdown does not wait for applications to disconnect from the queue manager.

If the controlled stop fails, it might be necessary to force the queue manager to shut down by stopping the queue manager processes. Use this option only when all other options to stop the queue manager failed. For more information about stop methods and manual shutdown, see:

https://www.ibm.com/support/knowledgecenter/en/SSFKSJ_9.1.0/com.ibm.mq.adm.doc/q020600_.htm

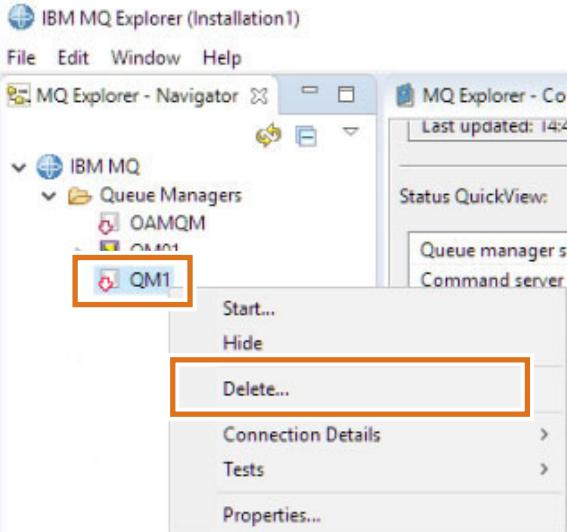


Deleting queue managers

- Options for deleting queue managers

MQ control command: `dltmqm QMgrName`

MQ Explorer



IBM MQ Console

Local Queue Managers	
Name	Status
QM01	Running
QM02	Stopped

Working with IBM MQ administration tools

© Copyright IBM Corporation 2020

Figure 2-42. Deleting queue managers

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.

After you stop the queue manager, you can delete it by using MQ Explorer, IBM MQ Console or with the `dltmqm` control command.

Unit summary

- Create queue managers and objects with IBM MQ control commands and script commands
- Describe queue manager sets
- Stop and delete queue managers

Working with IBM MQ administration tools

© Copyright IBM Corporation 2020

Figure 2-43. Unit summary

Review questions (1 of 2)

1. True or False: Queue manager names can contain a maximum of 48 characters.
2. True or False: You must stop a queue manager before you delete it.
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.



Figure 2-44. 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 2-45. Review questions (2 of 2)

Write your answer here:

5.

Review answers (1 of 2)

1. True or False: Queue manager names can contain a maximum of 48 characters.
The answer is True.

2. True or False: You must stop a queue manager before you delete it.
The answer is True.

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.



Figure 2-46. 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 managerThe answer is B.



Figure 2-47. Review answers (2 of 2)

Exercise: Using commands to create queue managers and queues

© Copyright IBM Corporation 2020
Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Figure 2-48. 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

Working with IBM MQ administration tools

© Copyright IBM Corporation 2020

Figure 2-49. Exercise objectives

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

Unit 3. Configuring distributed queuing

Estimated time

01:30

Overview

In this unit, you learn how to set up a distributed topology with all the required components, including message channels.

How you will check your progress

- Review questions
- Lab exercise

Unit objectives

- Diagram the components of a distributed topology
- Explain how point-to-point messaging works
- 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

Configuring distributed queuing

© Copyright IBM Corporation 2020

Figure 3-1. Unit objectives

Topics

- Distributed queuing
- Working with transmission queues
- Working with channels
- Accessing remote queue managers

Configuring distributed queuing

© Copyright IBM Corporation 2020

Figure 3-2. Topics

3.1. Distributed queuing

Distributed queuing

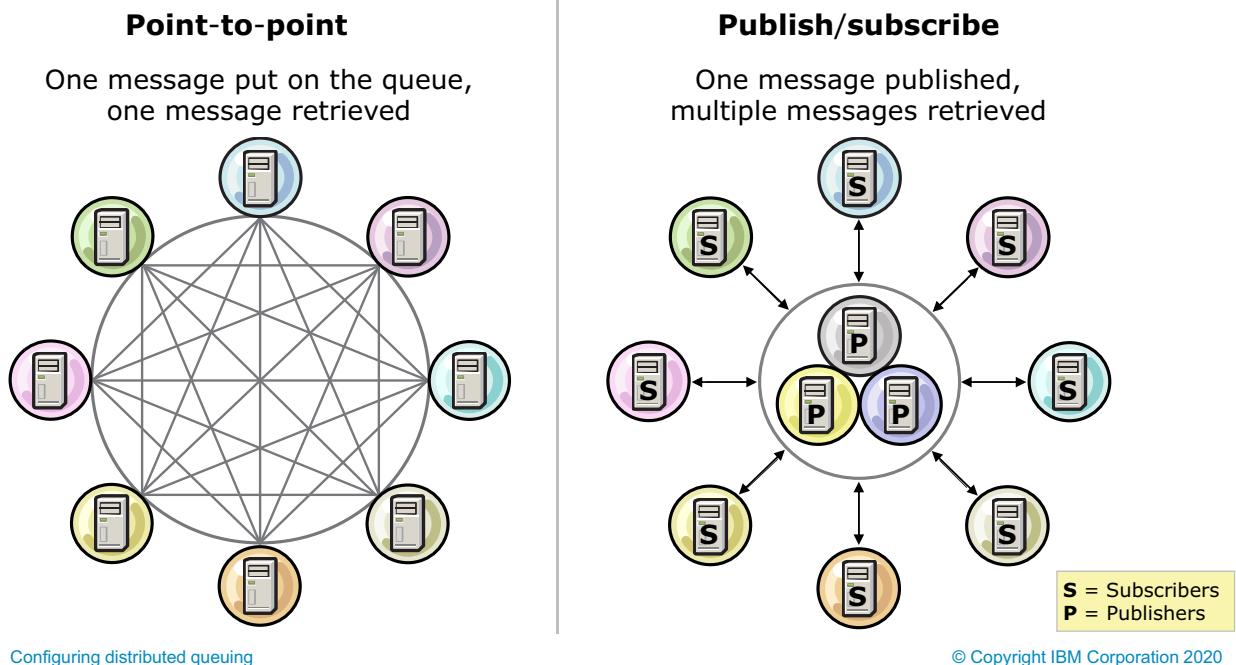
Configuring distributed queuing

© Copyright IBM Corporation 2020

Figure 3-3. Distributed queuing

Message-queuing styles

- Distributed queuing
 - Messages sent from one queue manager to another queue manager



Configuring distributed queuing

© Copyright IBM Corporation 2020

Figure 3-4. Message-queuing styles

Distributed queuing means sending messages from one queue manager to another. The receiving queue manager can be on the same system, on another nearby system, or on the other side of the world. It can be running on the same platform as the local queue manager, or can be on any of the platforms that are supported by IBM MQ. You can manually define all the connections in a distributed queuing environment, or you can create a cluster and let IBM MQ define much of the connection detail for you.

IBM MQ supports two message-queuing styles:

- Point-to-point
- Publish/subscribe

Point-to-point messaging is the ability to define queues, have message producers put messages on a queue, and have the messages retrieved by a specific consumer. One message comes in, one message goes out.

Publish/subscribe also involves queues, message producers, and consumers, along with the additional concepts of topics and subscriptions. A subscription is a way of matching messages to interested consumers. Instead of working directly with a queue, a message producer publishes messages to a topic. IBM MQ works out which subscriptions are interested in that topic, and delivers copies of the message to each subscription queue. One message comes in, multiple messages go out.

You learn about publish/subscribe later.

For more information, see:

- IBM Knowledge Center:
https://www.ibm.com/support/knowledgecenter/SSFKSJ_9.1.0/com.ibm.mq.pro.doc/q002660.htm#q002660_clusters
- Presentation by Dave Ware: <https://www.youtube.com/watch?v=szqdtlEgTR4>

3.2. Point-to-point topology

Point-to-point topology

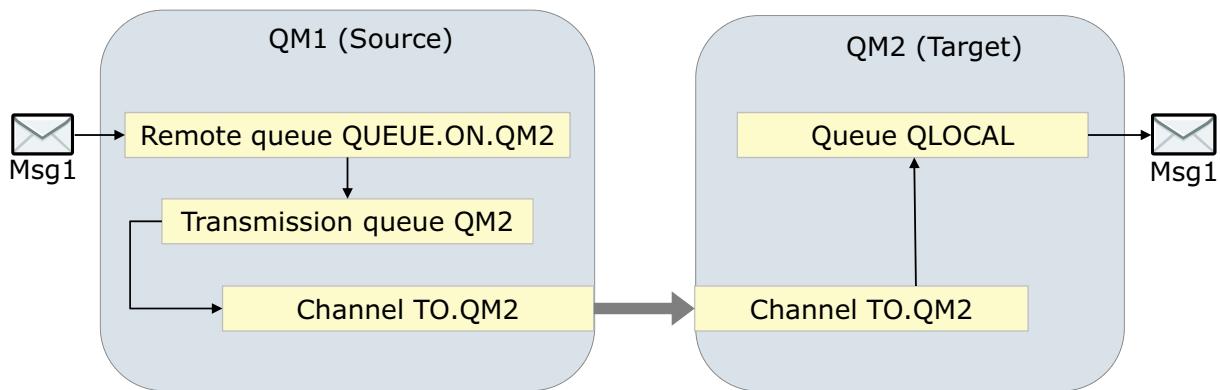
Configuring distributed queuing

© Copyright IBM Corporation 2020

Figure 3-5. Point-to-point topology

Create a point-to-point topology

- Define source queue manager objects:
 - Remote queue definition
 - Transmission queue
 - Sender channel
- Define target queue manager objects:
 - Receiver channel
 - Local queue



Configuring distributed queuing

© Copyright IBM Corporation 2020

Figure 3-6. Create a point-to-point topology

A distributed topology includes a local or source queue manager and a target queue manager.

The local queue manager needs:

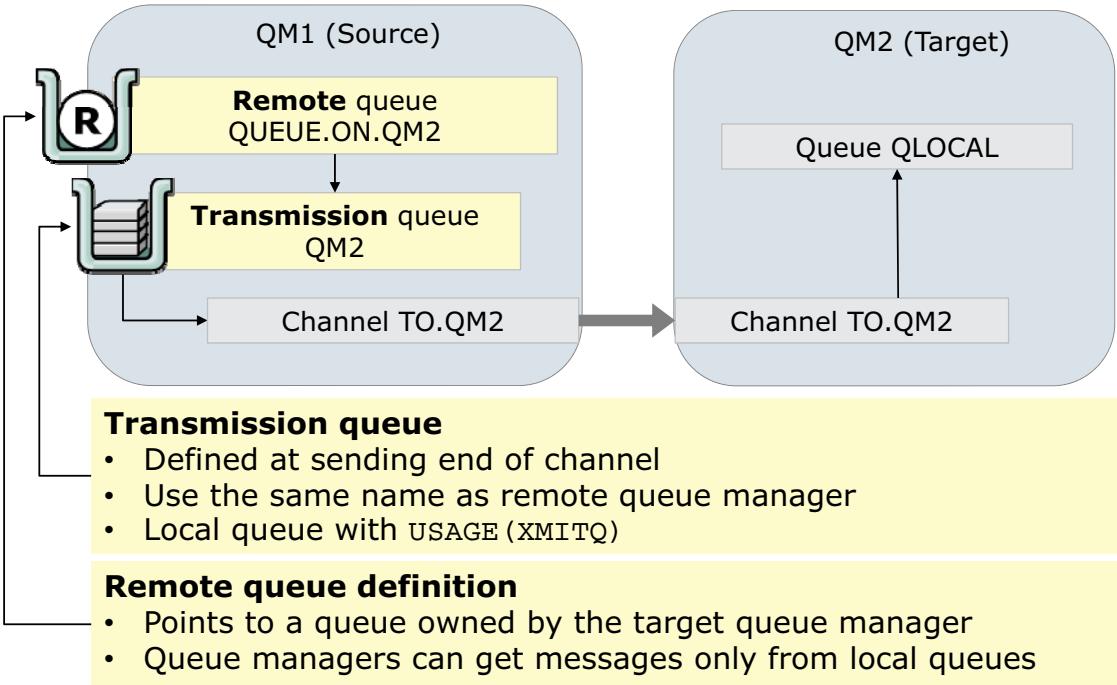
- A transmission queue, which is a local queue that is used to store messages until they can be forwarded
- A remote queue definition, which is a pointer to a queue on the target queue manager.
- A sender channel

The target queue manager needs:

- A receiver channel, which is compatible with the sender channel on the source queue manager
- A local queue, from which the queue manager can retrieve messages

Point-to-point: Required queues

- Two special queues required on the source queue manager



Configuring distributed queuing

© Copyright IBM Corporation 2020

Figure 3-7. Point-to-point: Required queues

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

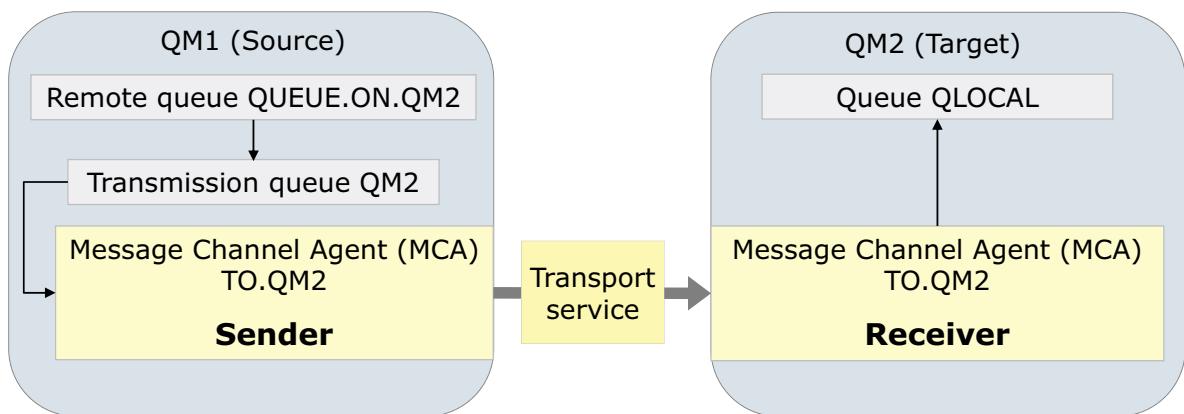
The **transmission queue** is a special type of local queue that stores messages before they are sent to the remote queue manager. You define one transmission queue for each sending channel. For clarity, give a transmission queue the same name as the remote queue manager.

The **remote queue definition** points to the local queue that is owned by the target queue manager. Queue managers can retrieve messages from local queues only. In the example on the slide, QM2 **cannot get** any messages that are stored on QM1's queues. However, queue managers can retrieve messages from both local or remote queues. Remote queue definitions allow applications to put a message on a queue that belongs to another queue manager. So QM2 **can get** a message that was put on the remote queue QUEUE.ON.QM2.

In the remote queue definition, you include the name of the transmission queue and the name of queue manager that owns the remote queue. If you do not specify a transmission queue, the queue manager looks for a transmission queue with the same name as the remote queue manager.

Moving and transport services

- Message channel agents move messages from one queue manager to another
 - A message channel agent is one end of a channel
 - A pair of message channel agents (sending and receiving) makes up a channel
- Transport service connects to target queue manager and opens queue
 - Can be: TCP/IP, SNA, SPX, NetBIOS



Configuring distributed queuing

© Copyright IBM Corporation 2020

Figure 3-8. Moving and transport services

How does the message get from QM1 to QM2? By the *message channel*. A channel carries messages from one queue manager to another. Each end of a message channel can be a sender or receiver. As you see in the example, the pair of channels uses the same name.

Each queue manager contains communication software that is called the *moving service* component by which the queue manager can communicate with other queue managers. The moving service is handled by message channel agents (MCA). A message channel agent is one end of a channel. A pair of message channel agents, one sending and one receiving, make up a channel. One MCA takes messages from the transmission queue and puts them on the communication link. The other MCA receives messages and delivers them onto a queue on the remote queue manager.

The sending MCA splits large messages before sending them across the channel. They are reassembled at the remote queue manager, which is hidden from the user.

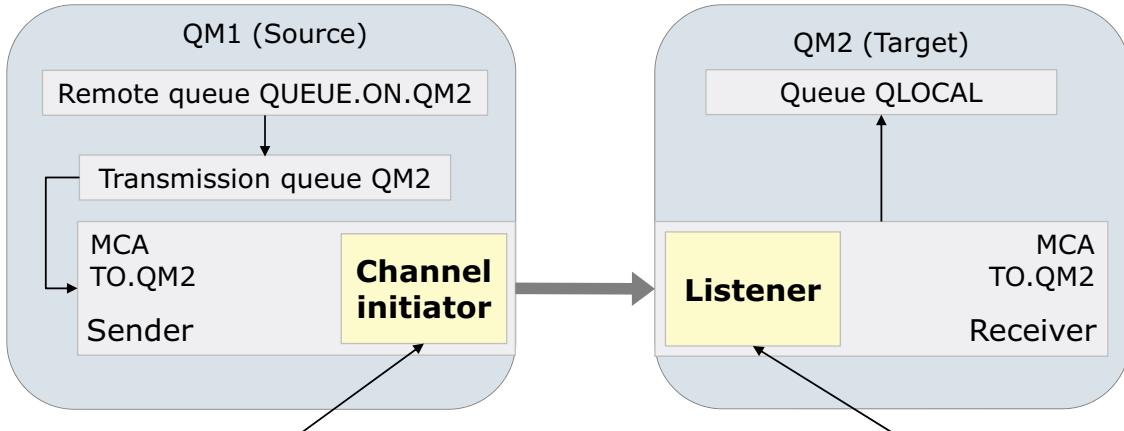
An MCA can transfer messages by using multiple threads. This process, called *pipelining* enables the MCA to transfer messages more efficiently, with fewer wait states. Pipelining improves channel performance. For more information, see:

https://www.ibm.com/support/knowledgecenter/en/SSFKSJ_9.1.0/com.ibm.mq.con.doc/q015580_.htm

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)

Channel initiators and listeners



Channel initiator

- When a message arrives on a transmission queue, the channel initiator is triggered to start the sender channel

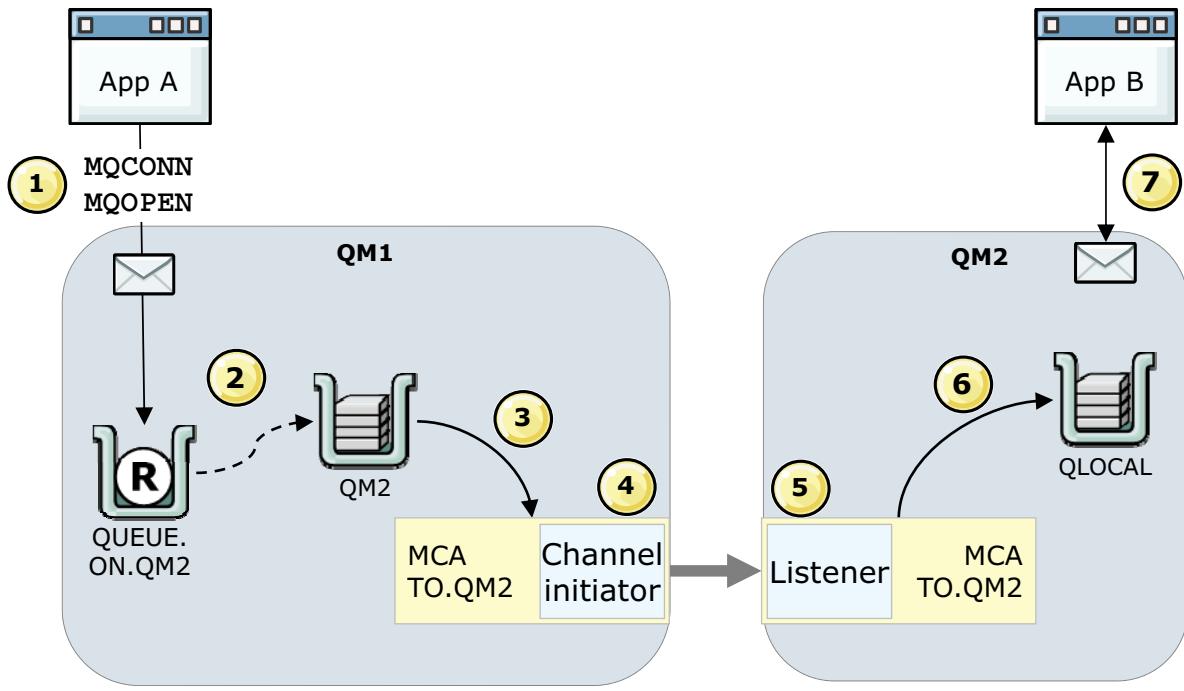
Channel listener

- Listens for incoming network requests
- Starts the receiver channel when it is needed

Figure 3-9. Channel initiators and listeners

- 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, a message is sent to the initiation queue, which triggers the channel initiator to start the sender channel.
- A *channel listener* program listens for incoming network requests and starts the appropriate receiver channel when it is needed.
- IBM MQ calls *channel-exit programs* at defined places in the processing, which is carried out by the MCA.

Point-to-point: How does it work?



Configuring distributed queuing

© Copyright IBM Corporation 2020

Figure 3-10. Point-to-point: How does it work?

This figure reviews the IBM 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. An application connects to a queue manager and opens a queue to transmit messages to another queue manager.

1. App A sends MQCONN call to connect to QM1 and sends MQOPEN call to open the queue and puts a message on QUEUE.ON.QM2, which is the remote queue definition that points to QLOCAL on QM2.
2. QM1 puts the message on the QM2 transmission queue, and the transmission queue stores the message while checking whether the channel is active.
3. The channel initiator recognizes that a message was put on the transmission queue and triggers the TO.QM2 sender channel to start.
4. QM1 transmits the message to queue manager QM2 by the TO.QM2 sender channel.
5. The **listener** on queue manager QM2 starts the TO.QM2 receiver channel and the message is received.
6. QM2 puts the message on the target queue QLOCAL.
7. App B contacts the queue manager to check for messages. App B can use one of these protocols:

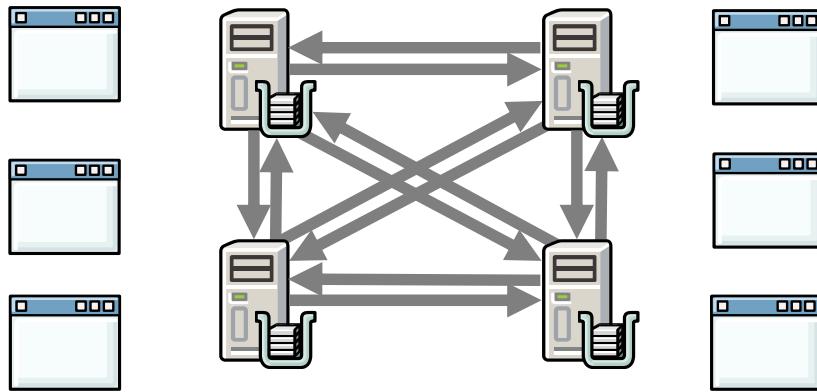
- Get/wait: “Get me any messages that you have for me, I’ll wait”
- Register callback: “Check whether you have any messages for me. If you find any, call me back.”

This implementation provides a level of decoupling. If necessary, you can change the remote queue definition to point to a different queue manager and 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.

Point-to-point: Manual versus clustering (1 of 2)

- Scaling up your IBM MQ network adds complexity
- Manual configuration can become unmanageable
 - Channel pairs to connect each of the queue managers
 - Other resources (remote queue definitions, transmission queues, listeners)



Configuring distributed queuing

© Copyright IBM Corporation 2020

Figure 3-11. Point-to-point: Manual versus clustering (1 of 2)

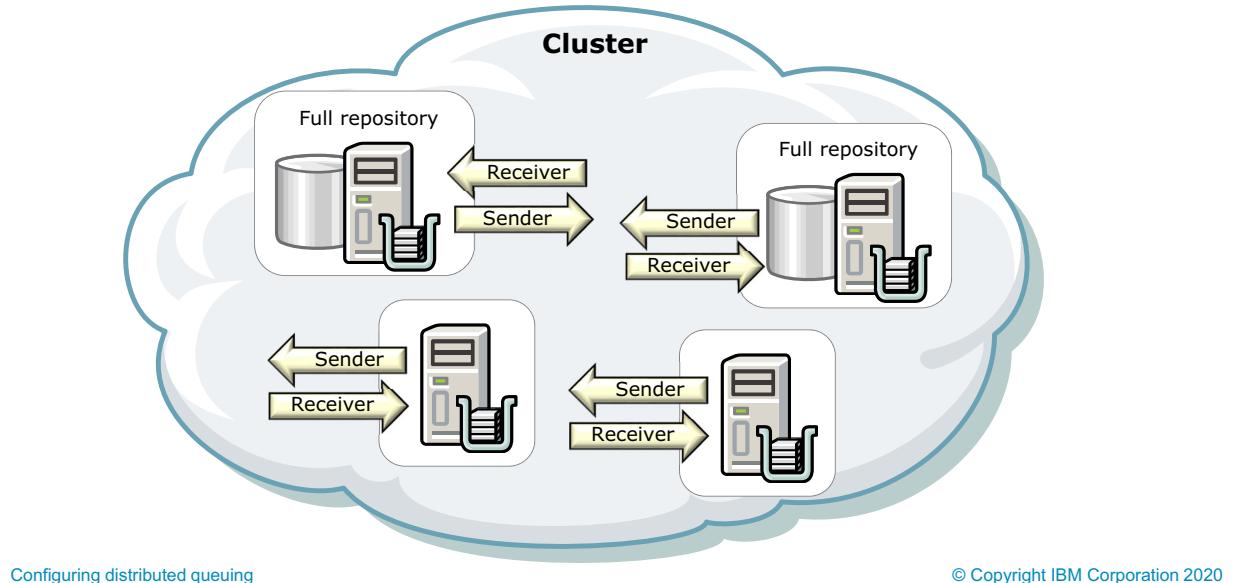
When you start with IBM MQ, your configuration might be very simple. You might have a single queue manager that uses a queue to put messages, retrieve and process them, and send a reply with a client/server application.

But as you start to use multiple queue managers, perhaps on multiple systems or in multiple locations, you need to join queue managers together. You connect them by defining MQ channels, point them at each other, and set up transmission queues and remote queue definitions.

As you scale up with more applications and more instances of those applications, adding extra systems and applications, you might need more queue managers to service them, either for scale or to service a larger network of systems. To have more queue managers, you must interconnect them with more MQ channels, plus all the transmission queues and remote queue definitions. If the interactions between the queue managers, channel definitions, and other resources are not tightly controlled, the MQ network could quickly become unmanageable.

Point-to-point: Manual versus clustering (2 of 2)

- IBM MQ clusters simplify the network
- Each queue manager requires only two channel definitions:
 - Receiver: All other queue managers communicate
 - Sender: Used to connect to one full-repository queue manager



Configuring distributed queuing

© Copyright IBM Corporation 2020

Figure 3-12. Point-to-point: Manual versus clustering (2 of 2)

[NOTES from David Ware's cluster presentation]

IBM MQ clusters simplify your MQ estate. An MQ cluster resolves down to every queue manager needing two channel definitions:

- Receiver channel: "This is how everyone in the cluster is going to talk to me"
- Sender channel: To bootstrap you into one of the full repository queue managers

After the cluster is set up, you don't have to worry about remote queue definitions or specific channels between any pairs of queue managers. The cluster works that out for you automatically, and creates resources for you without you even knowing that they are there.

You learn more about clusters in a later unit.

3.3. Working with channels

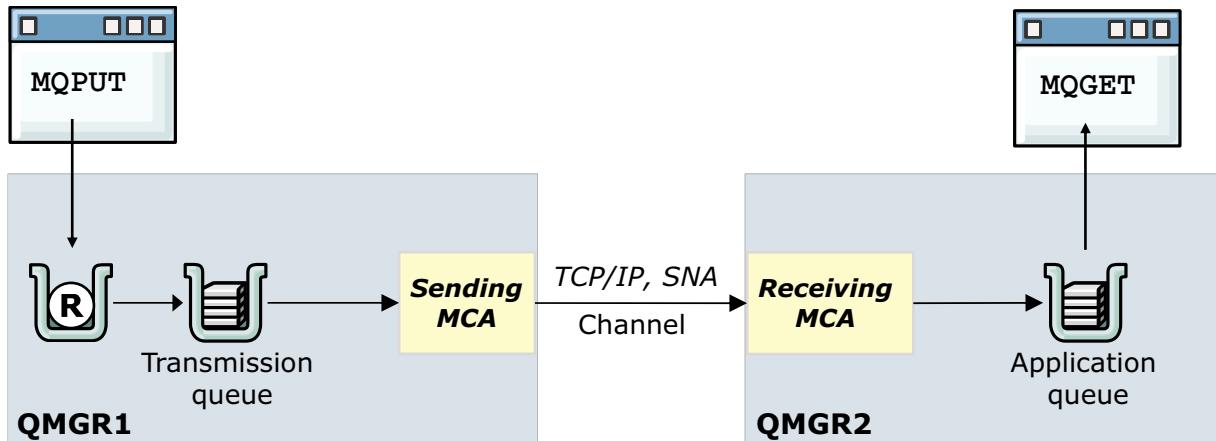
Working with channels

Configuring distributed queuing

© Copyright IBM Corporation 2020

Figure 3-13. Working with channels

Message channels



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

Configuring distributed queuing

© Copyright IBM Corporation 2020

Figure 3-14. Message channels

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 them. The communications connection might be, for example, an SNA LU6.2 conversation or a TCP connection.

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

Because a transmission queue is required at the sending end of a message channel, only the sender channel definition includes the name of the transmission queue. To ensure that the message is sent to the correct destination, give the transmission queue the same name as 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)
- Define a message channel with compatible pairs of channel types:
 - Sender + receiver
 - Requester + server
 - Requester + sender (for callback)
 - Server + receiver (server is used as a sender)
 - Cluster-sender + cluster-receiver

Configuring distributed queuing

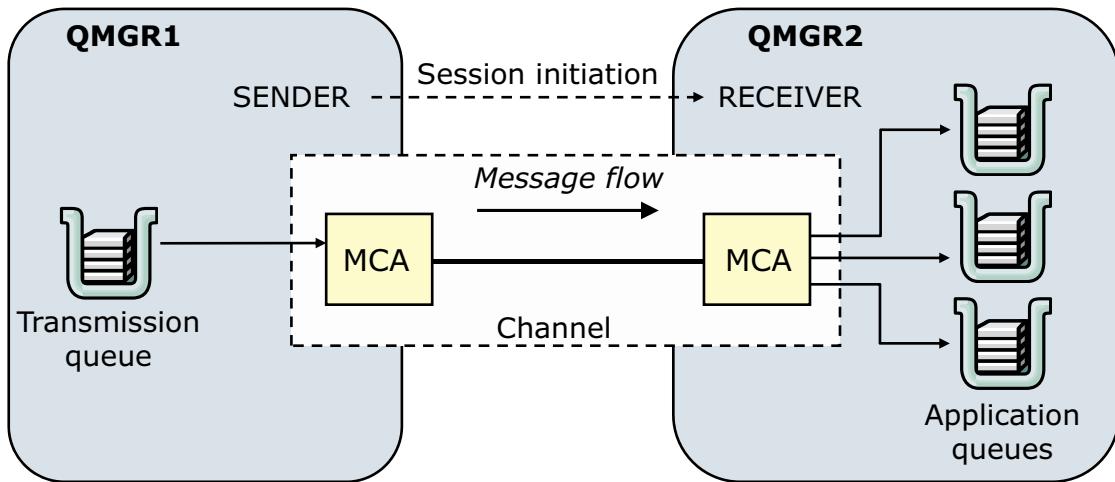
© Copyright IBM Corporation 2020

Figure 3-15. Message channel types

The slide 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 IBM 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 the transmission queue to the receiver
- **Receiver puts the messages on the destination queue**

Configuring distributed queuing

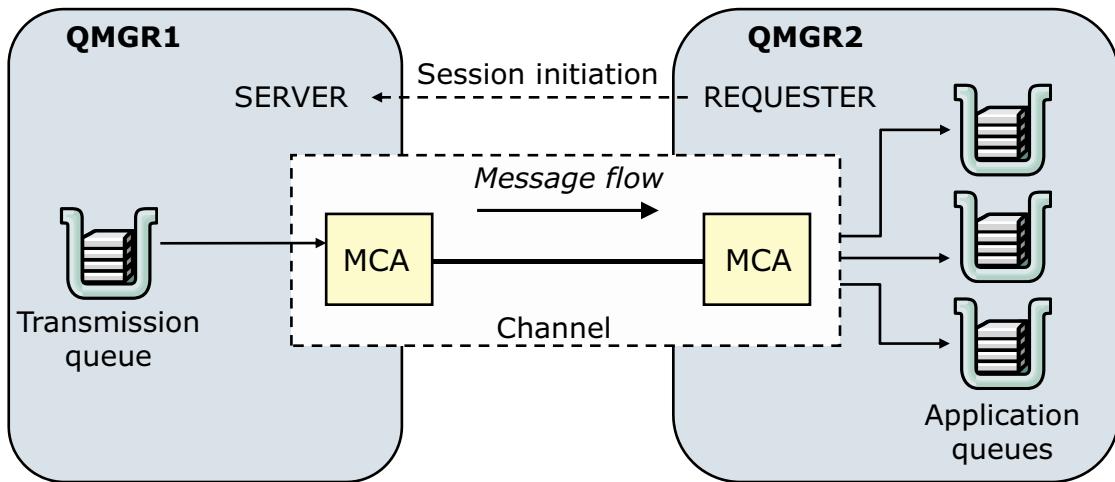
© Copyright IBM Corporation 2020

Figure 3-16. 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

Configuring distributed queuing

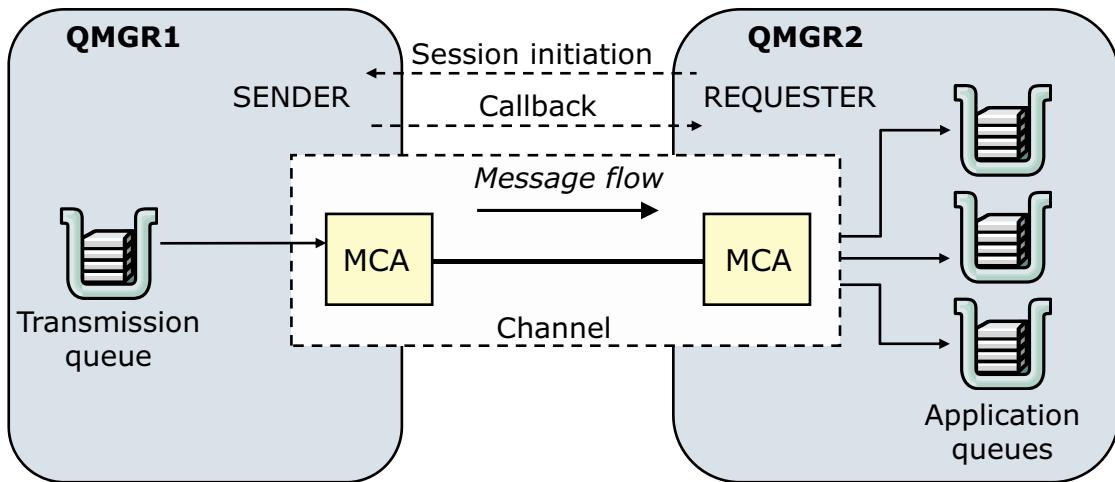
© Copyright IBM Corporation 2020

Figure 3-17. 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
 - Ends the call
 - Restarts the communication according to information in its channel definition
 - Sends messages from the transmission queue to the requester

Configuring distributed queuing

© Copyright IBM Corporation 2020

Figure 3-18. 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

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

Configuring distributed queuing

© Copyright IBM Corporation 2020

Figure 3-19. *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 hostname 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)
```

Configuring distributed queuing

© Copyright IBM Corporation 2020

Figure 3-20. 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 channel definition 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 hostname on the **CONNNAME** parameter. The port number is the port number of the queue manager listener, which must be running.

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

[Configuring distributed queuing](#)

© Copyright IBM Corporation 2020

Figure 3-21. 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)

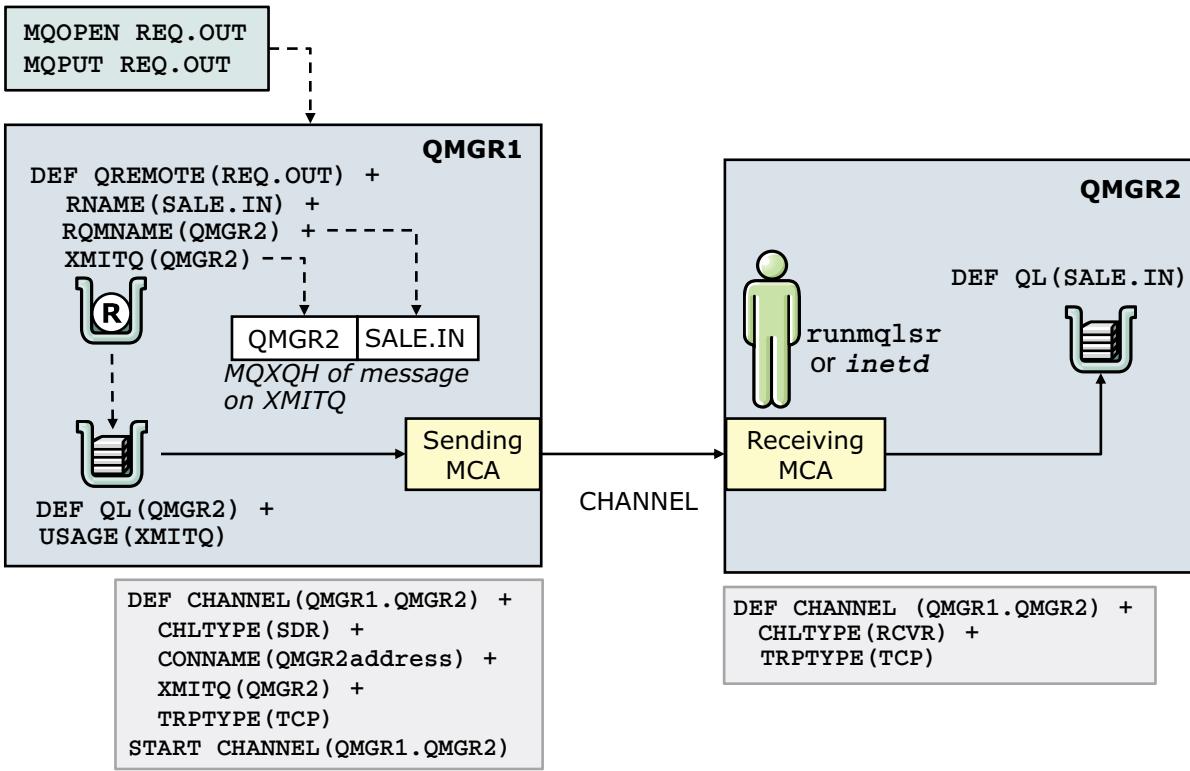
On the target queue manager, define a channel with a channel type of **RECEIVER**, and the same name as the sender channel. Also, specify the communication protocol on the channel (the **TRPTYPE** attribute).

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



Configuring distributed queuing

© Copyright IBM Corporation 2020

Figure 3-22. 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:

<code>MAXMSGL</code>	Maximum message length
----------------------	------------------------

<code>HBINT</code>	Heartbeat interval
--------------------	--------------------

<code>NPMSSPEED</code>	Nonpersistent message speed
------------------------	-----------------------------

Figure 3-23. Starting a message channel

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

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

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

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

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

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

If a message cannot be delivered, it is put on the local dead-letter queue. If it cannot be put there, the channel is stopped. However, if a fast nonpersistent message cannot be delivered and cannot be put on the dead-letter queue, it is discarded and the channel remains open.

One of the most common problems in IBM 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

[Configuring distributed queuing](#)

© Copyright IBM Corporation 2020

Figure 3-24. Displaying channel status

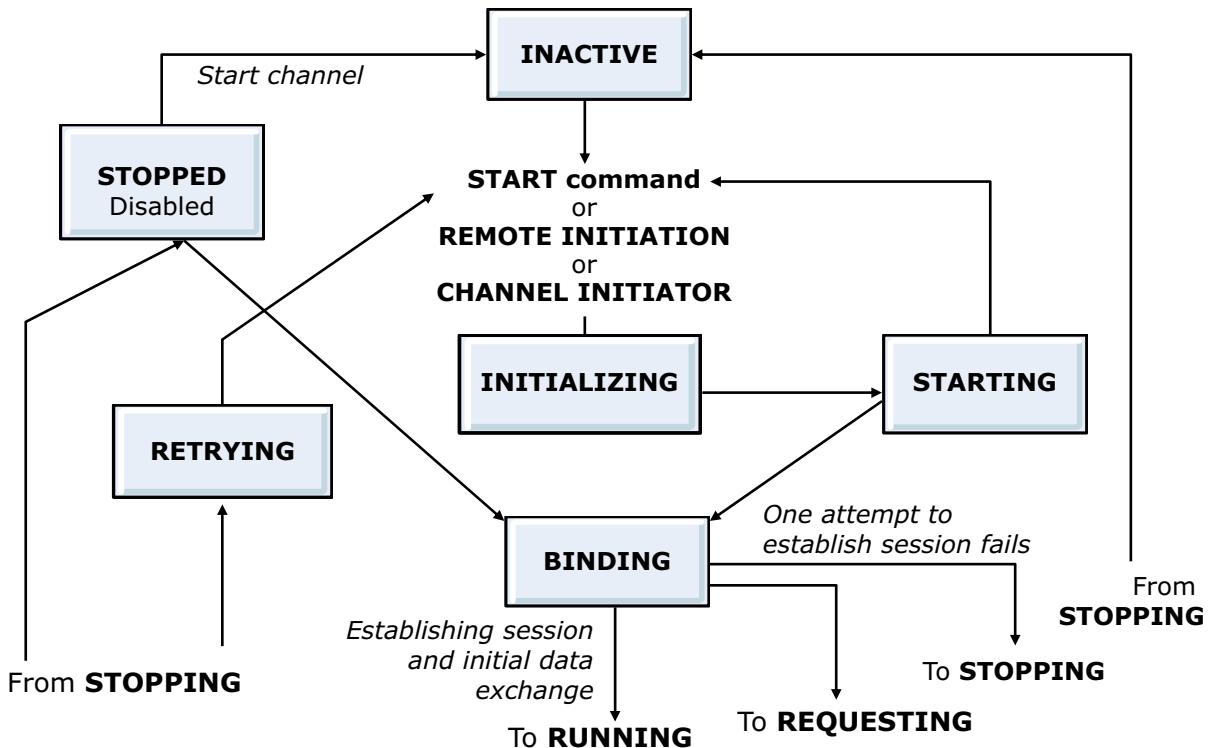
You can use the `DISPLAY CHSTATUS` command or MQ Explorer to display the status of one or more channels.

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.

Channel states (1 of 2)



Configuring distributed queuing

© Copyright IBM Corporation 2020

Figure 3-25. 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.

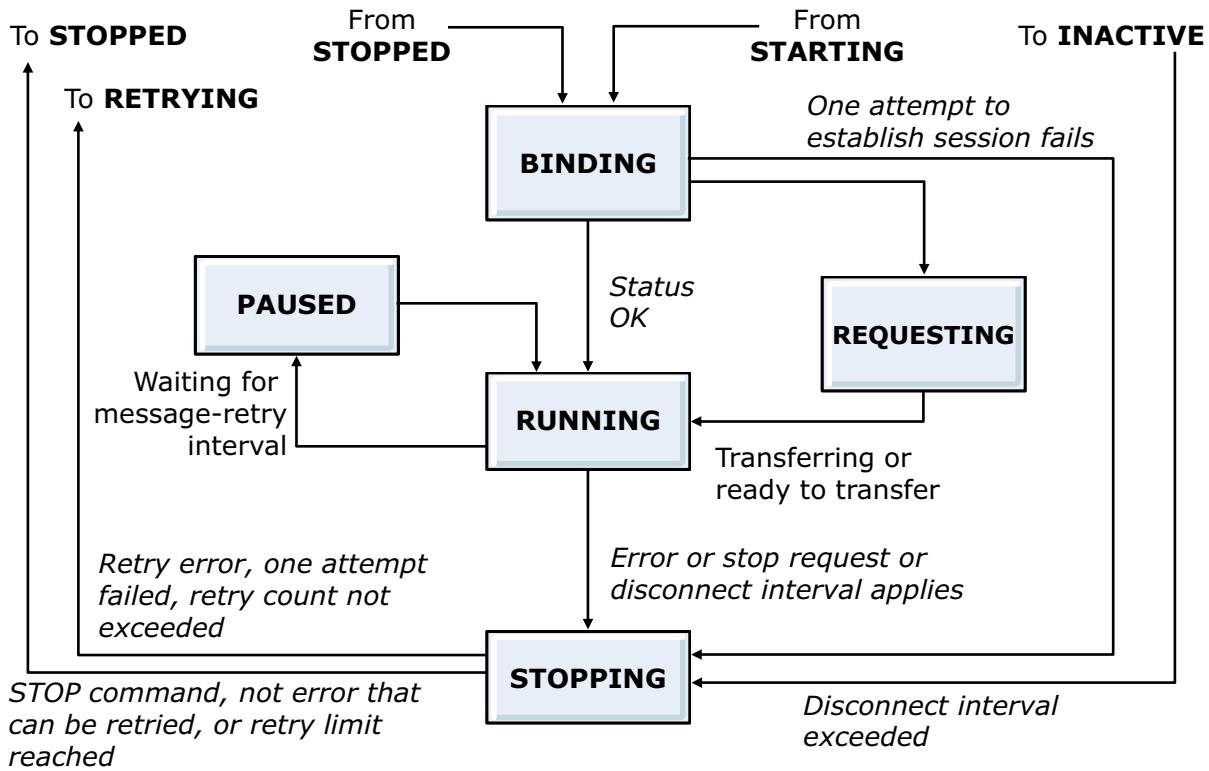
The RETRYING state indicates that the channel is waiting until it is time for the channel initiator to make the next attempt to start the channel. An error can cause a channel to go into the RETRYING state when it is possible that the problem might clear itself. Otherwise, it goes into the STOPPED state.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



[Configuring distributed queuing](#)

© Copyright IBM Corporation 2020

Figure 3-26. Channel states (2 of 2)

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

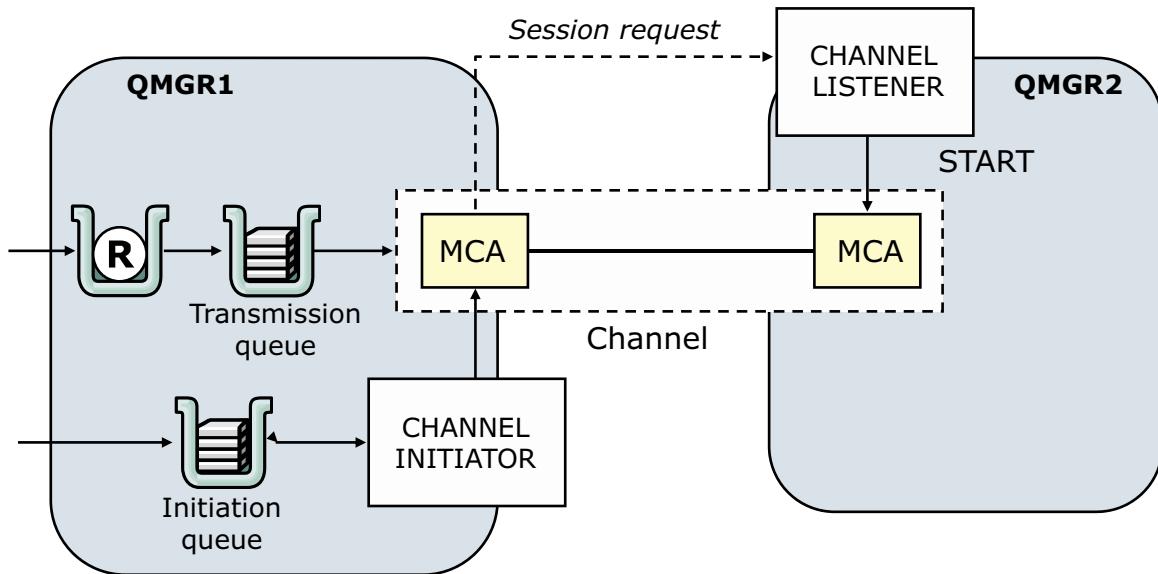
In the **REQUESTING** state, a requester is waiting for callback from a sender.

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 IBM 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

Configuring distributed queuing

© Copyright IBM Corporation 2020

Figure 3-27. 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 `runmq1sr`
 - 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 runmq1sr -t TCP -p 1414 -m QMGR`
- End IBM MQ listener: `endmq1sr`

Option 2

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

Configuring distributed queuing

© Copyright IBM Corporation 2020

Figure 3-28. 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 IBM 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 IBM 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 IBM MQ, use the `runmq1sr` command. One advantage of using `runmq1sr` 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 `endmq1sr` 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 IBM MQ operating systems, the listener can be started by using the MQSC command `START LISTENER`.

Listener object

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

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

Figure 3-29. 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 IBM 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.

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

Note: Exit programs are not supported on the IBM MQ Appliance

Figure 3-30. Channel-exit programs for message channels (1 of 2)

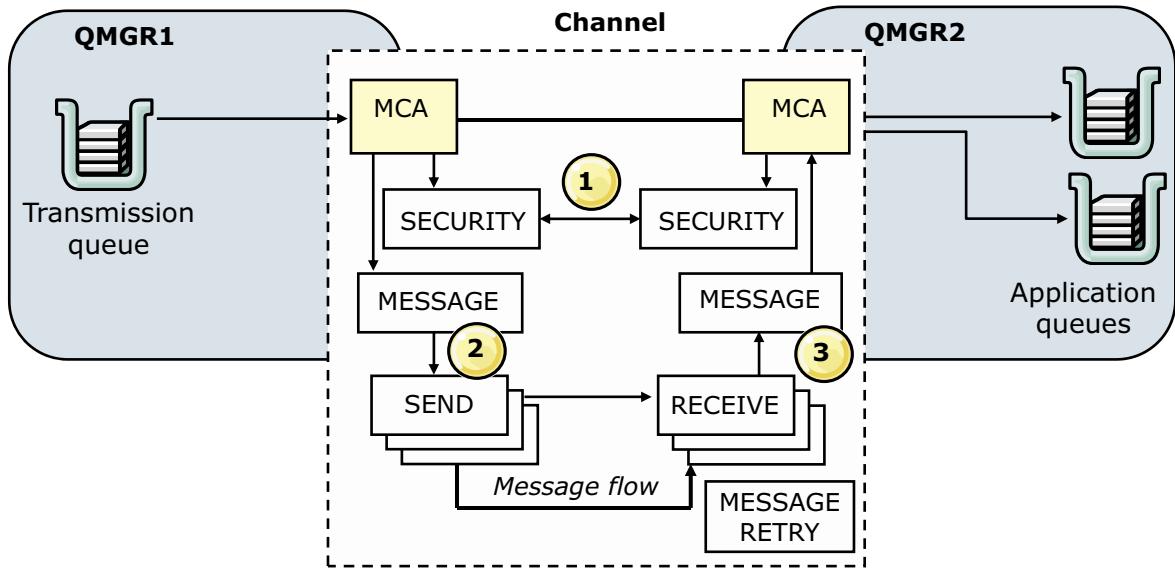
IBM MQ calls channel-exit programs at defined places in the processing carried out by the MCA.

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 IBM MQ capabilities such as SSL for encryption, or channel authorization and connection authorization for authentication.

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

Configuring distributed queuing

© Copyright IBM Corporation 2020

Figure 3-31. 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.

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

Figure 3-32. 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.

3.4. Message data conversion

Message data conversion

Configuring distributed queuing

© Copyright IBM Corporation 2020

Figure 3-33. Message data conversion

Data representation and conversion

- An IBM MQ message consists of two parts:
 - Control information in a message descriptor
 - Application data
- Either part might require data conversion when sent between queues on different queue managers
 - Character fields might need conversion
 - Numeric fields might need transformation
- Message channel agents (MCA) can convert the message descriptor and header data into the required character set and encoding
 - MCAs at either end (sending or receiving) can do the conversion

[Configuring distributed queuing](#)

© Copyright IBM Corporation 2020

Figure 3-34. Data representation and conversion

An IBM MQ message consists of two parts:

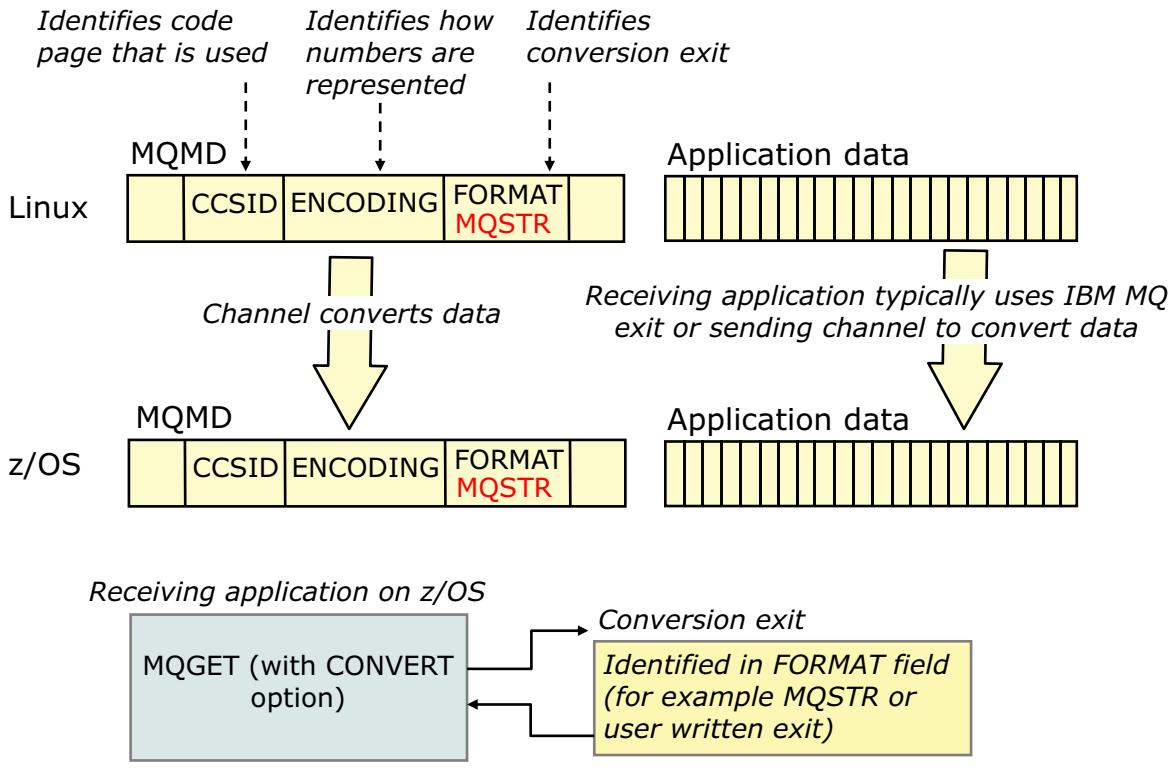
- Control information in a message descriptor
- Application data

When the queue managers or applications are on different operating systems and hardware, application data might need to be converted to the character set and the encoding that is required by the receiving application.

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

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

Data conversion example



Configuring distributed queuing

© Copyright IBM Corporation 2020

Figure 3-35. 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 IBM 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

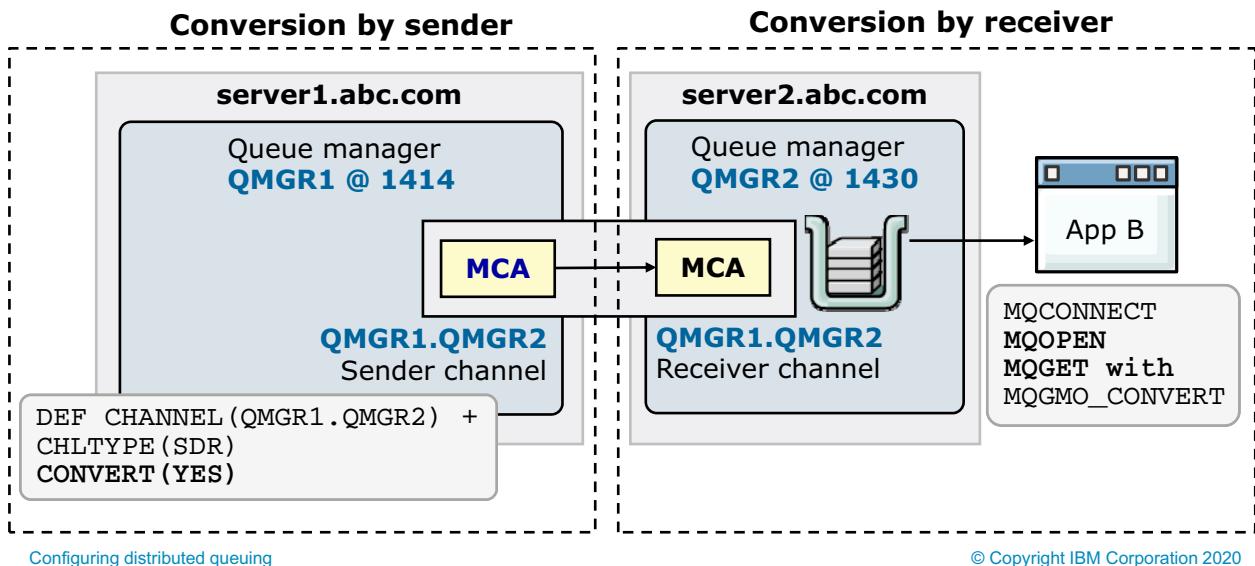


Figure 3-36. Application data conversion configuration

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 data conversion can be done

- Message data converted to requested character set
- Numeric fields converted to the encoding requested
- Some formats are built in
 - Data conversion done by 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
- See data conversion in IBM Knowledge Center for IBM MQ

Figure 3-37. What data conversion can be done

Data in the message that is described as character data by the built-in format or data-conversion exit is converted from the coded character set used by the message to that requested.

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

For more information, see

https://www.ibm.com/support/knowledgecenter/SSFKSJ_9.1.0/com.ibm.mq.ref.dev.doc/q104060_.htm

3.5. Message delivery

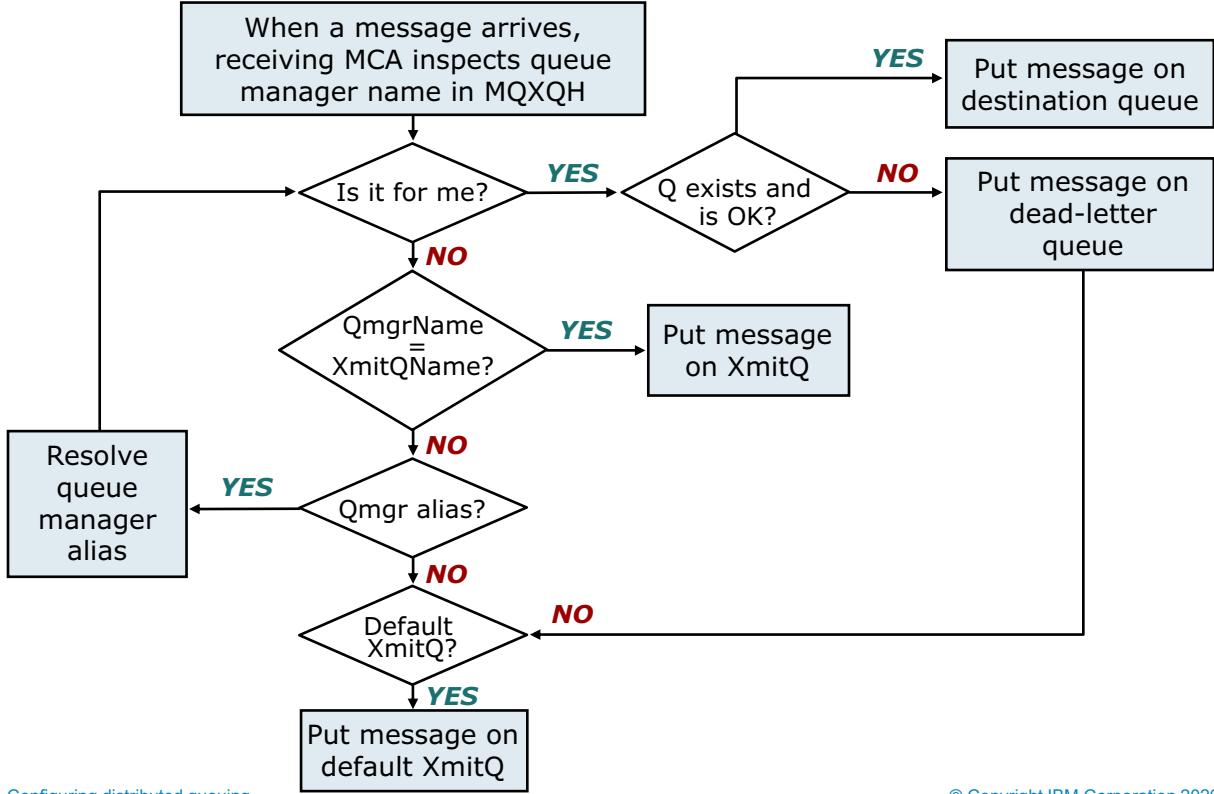
Message delivery

Configuring distributed queuing

© Copyright IBM Corporation 2020

Figure 3-38. Message delivery

When a message arrives at a queue manager



Configuring distributed queuing

© Copyright IBM Corporation 2020

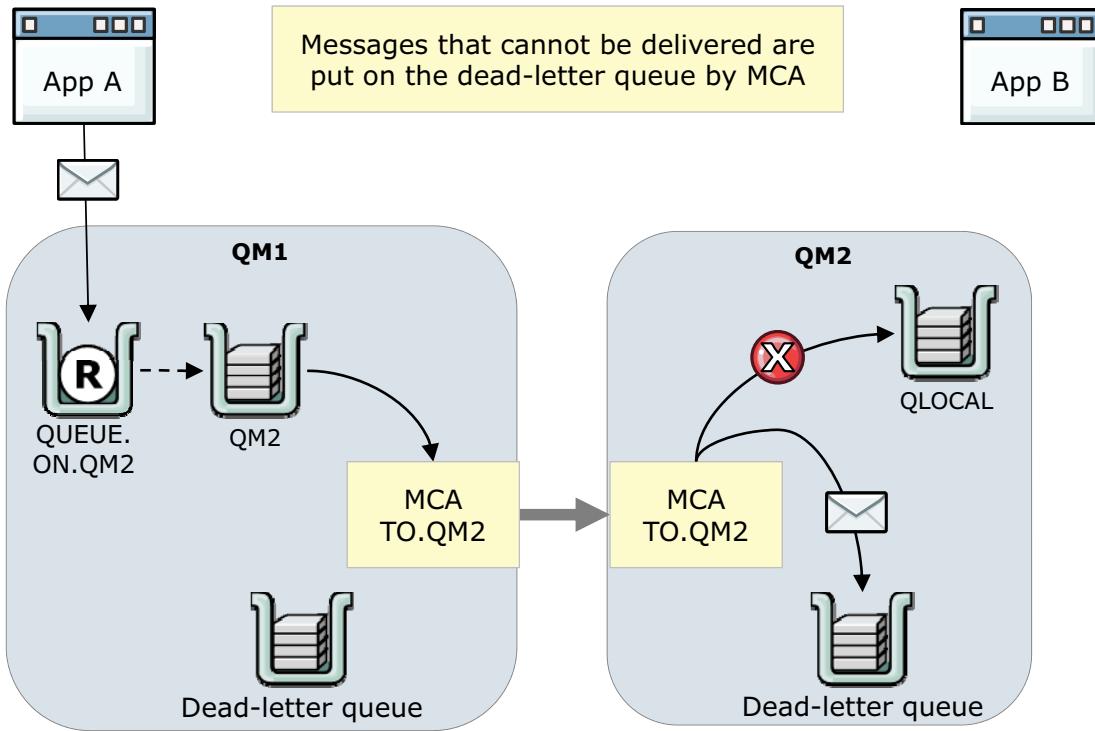
Figure 3-39. When a message arrives at a queue manager

During message processing, it is expected that the message can reach its target. Messages that cannot be delivered 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.

What happens when a message cannot be delivered?



Configuring distributed queuing

© Copyright IBM Corporation 2020

Figure 3-40. What happens when a message cannot be delivered?

https://www.ibm.com/support/knowledgecenter/SSFKSJ_9.1.0/com.ibm.mq.pro.doc/q002680_.htm

If a message cannot be delivered to its correct destination, maybe because the queue is full or the specified queue does not exist, what happens?

These messages are routed to the *dead-letter queue* (DLQ) or *undelivered-message queue*, at the receiving end of a message channel. Undelivered messages can be stored on the dead-letter queue for later retrieval.

Dead-letter queues are also used at the sending end of a channel, for data-conversion errors.

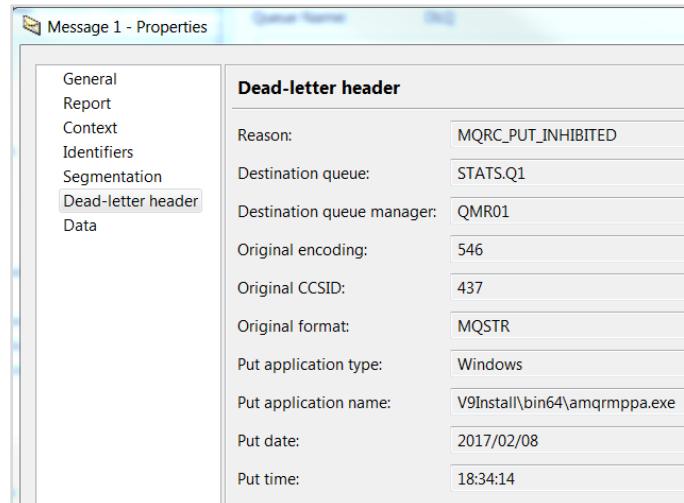
Queue managers, message channel agents (MCAs), and applications can all put messages on the dead-letter queue. All messages on the dead-letter queue must be prefixed with a *dead-letter header* structure, MQDLH. The *Reason* field of the MQDLH structure contains a reason code that identifies why the message is on the dead-letter queue.

If no dead-letter queue is defined for a queue manager, and the MCA is unable to put a message, the message is left on the transmission queue and the channel is stopped. For fast channels, non-persistent messages that cannot be delivered are discarded if no dead-letter queue exists on the target system.

You should always define a dead-letter queue for every queue manager in a network.

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



Configuring distributed queuing

© Copyright IBM Corporation 2020

Figure 3-41. 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.

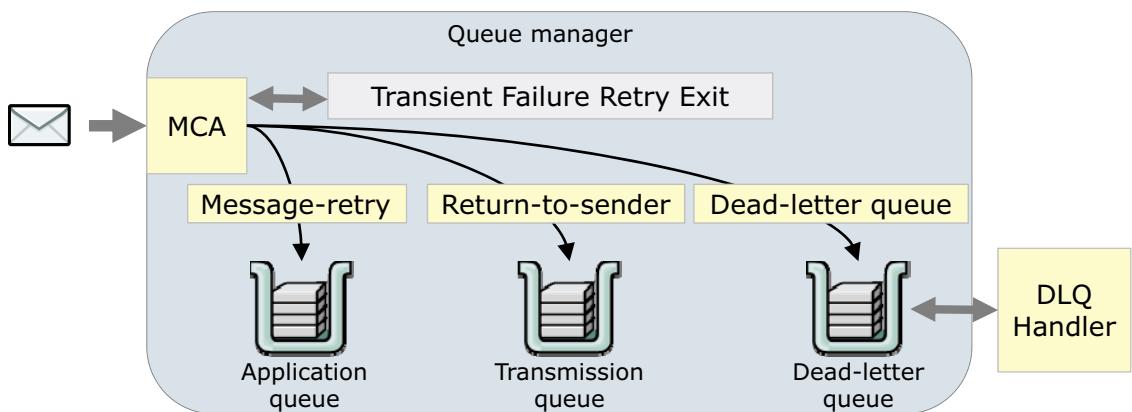
Fields in the MQDLH structure include:

- 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

For more information about the dead-letter header, see the IBM Knowledge Center.

Guidelines for handling undelivered messages

- Create a dead-letter queue on all queue managers
- Use message-retry on message channels for transient conditions
- Return the message to the originator
- If no retry options, put message on dead-letter queue
- Use IBM MQ Dead-letter Queue Handler to ensure that the dead-letter queue does not become full



Configuring distributed queuing

© Copyright IBM Corporation 2020

Figure 3-42. Guidelines for handling undelivered messages

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. The MCA can wait and try the operation again later. You can determine whether the MCA waits, for how long, and how many times it tries.
- 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 let an application dead-letter queue become full.
- Use the 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.
- If no reasonable retry option exists, forward the message to an application dead-letter queue.

For more information, see

https://www.ibm.com/support/knowledgecenter/en/SSFKSJ_9.1.0/com.ibm.mq.con.doc/q015720_.htm

Unit summary

- Diagram the components of a distributed topology
- Explain how point-to-point messaging works
- 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

Configuring distributed queuing

© Copyright IBM Corporation 2020

Figure 3-43. Unit summary

Review questions

1. True or False: You must define a transmission queue for every sending end of a message channel.
2. True or False: A common naming convention for a transmission queue is to use the same name as the target queue manager.
3. True or False: Applications can both *put* and *retrieve* messages from *local* and *remote* queues.



Configuring distributed queuing

© Copyright IBM Corporation 2020

Figure 3-44. Review questions

Write your answers down here:

- 1.
- 2.
- 3.

Review answers

1. True or False: You must define a transmission queue for every sending end of a message channel.
The answer is True.

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

3. True or False: Applications can both *put* and *retrieve* messages from *local* and *remote* queues.
The answer is False. Applications can put messages on local and remote queues, but they can only retrieve messages from local queues.



Configuring distributed queuing

© Copyright IBM Corporation 2020

Figure 3-45. Review answers

Write your answers down here:

- 1.
- 2.
- 3.

Exercise: Implementing distributed queuing

Configuring distributed queuing

© Copyright IBM Corporation 2020

Figure 3-46. Exercise: Implementing distributed queuing

Exercise introduction

- Set up a distributed topology
- Test point-to-point message queuing



Configuring distributed queuing

© Copyright IBM Corporation 2020

Figure 3-47. Exercise introduction

Unit 4. Managing clients and client connections

Estimated time

01:30

Overview

In this unit, you learn how to use different methods to connect IBM MQ clients to an IBM MQ server.

How you will check your progress

- Review questions
- Lab exercise

Unit objectives

- Describe the components of an MQI client connection
- Describe the various ways to connect a client to a queue manager
- Describe the client modes that MQSC supports
- Use troubleshooting tools and techniques to monitor and manage clients and connections

Topics

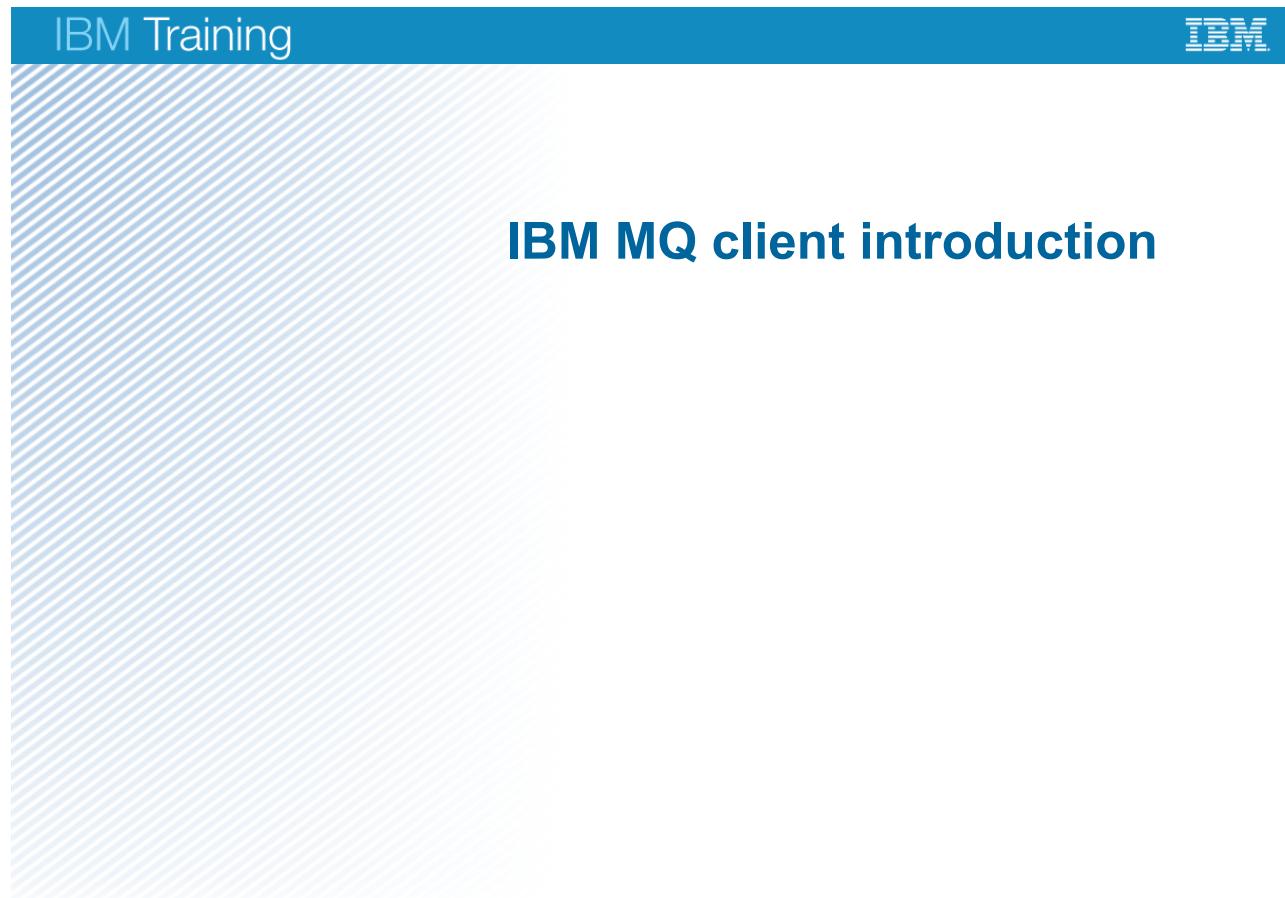
- IBM MQ client introduction
- Client configurations
- Client modes for MQSC
- Troubleshooting

Managing clients and client connections

© Copyright IBM Corporation 2020

Figure 4-2. Topics

4.1. IBM MQ client introduction



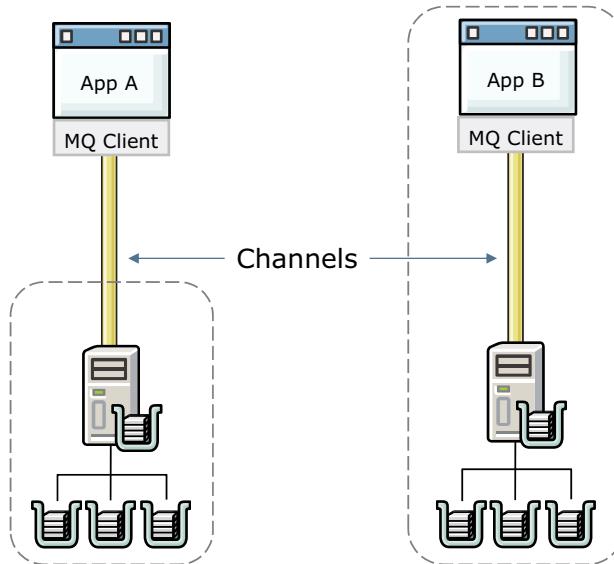
Managing clients and client connections

© Copyright IBM Corporation 2020

Figure 4-3. IBM MQ client introduction

IBM MQ client application options

- IBM MQ client applications can be on:
 - Separate system
 - Same system as queue manager



Managing clients and client connections

© Copyright IBM Corporation 2020

Figure 4-4. IBM MQ client application options

An IBM MQ MQI *client* is a component that allows an application on one system to issue MQI calls to a queue manager on another system. The output from the call is sent back to the client, which passes it back to the application. In this unit, you learn the configuration options.

An IBM MQ *server* is a queue manager that provides queuing services to one or more clients. All the IBM MQ objects, for example queues, exist only on the queue manager server (the IBM MQ server), and not on the client. An IBM MQ server can also support local IBM MQ applications.

See the client overview in the IBM Knowledge Center:

https://www.ibm.com/support/knowledgecenter/en/SSFKSJ_9.1.0/com.ibm.mq.pro.doc/q003460_.htm

See chapter 7 in the IBM Redbooks, *IBM MQ as a Service: A Practical Approach*:

<http://www.redbooks.ibm.com/abstracts/redp5209.html>

IBM MQ redistributable client

- Collection of runtime files
- Redistributable to a third party under redistributable license terms
- Single package for distribution
- Clients included are
 - Redistributable client
 - Relocatable client
 - Non-installed client
- Minimize files distributed with genmqpkg
 - Script provided with redistributable client
 - Determines needs of application
 - Generates the redistributable files needed
 - Allows package to be distributed to be as small as possible

[Managing clients and client connections](#)

© Copyright IBM Corporation 2020

Figure 4-5. IBM MQ redistributable client

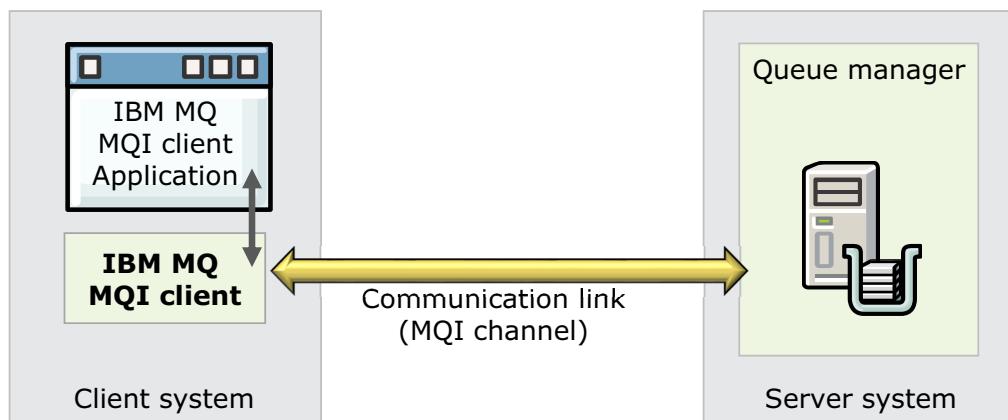
The IBM MQ redistributable client is a collection of runtime files that are provided in a .zip or .tar file that can be redistributed to third parties under redistributable license terms. This provides a simple way of distributing applications and the runtime files that they require in a single package.

The redistributable client that is supplied with IBM MQ is also a non-installed and relocatable image. Maintenance of a redistributable, non-installed image, is achieved through replacement; that is, you download newer versions of the runtime components when they are shipped. A *redistributable* client implies distributing the required run time with an application both inside and outside of your environment. A *relocatable* client implies putting the files somewhere else other than a fixed default location. For example, instead of installing into /opt/ installing into /usr/local. A *non-installed* client implies that you are not required to lay down client files, and that these files can be copied as required.

https://www.ibm.com/support/knowledgecenter/en/SSFKSJ_9.1.0/com.ibm.mq.ins.doc/q122882_.htm

IBM MQ MQI client

- Provides client-server support
- Multiple configuration options
- Many application possibilities
- Wide platform support



Managing clients and client connections

© Copyright IBM Corporation 2020

Figure 4-6. IBM MQ MQI client

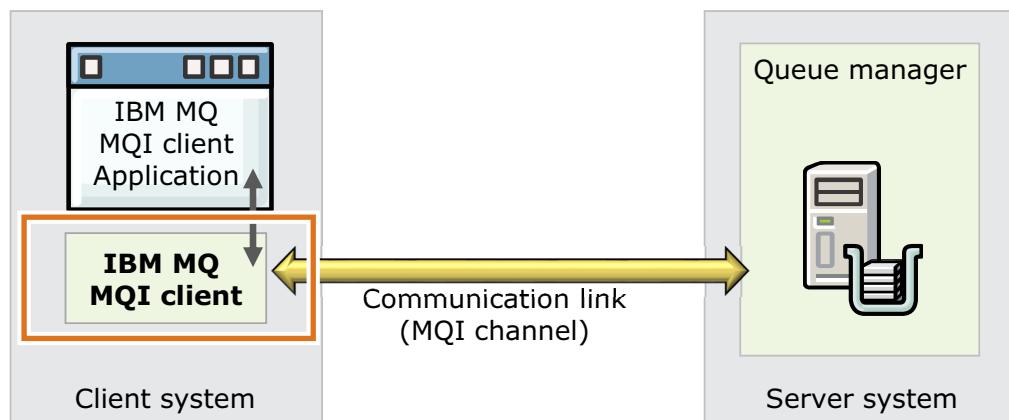
The IBM MQ client component provides for a server-client relationship allowing the IBM MQ applications to exist on a separate system from the IBM MQ queue manager. A client application can be connected to more than one queue manager server simultaneously. This unit shows the configuration options.

The IBM Knowledge Center has an overview at this URL:

https://www.ibm.com/support/knowledgecenter/en/SSFKSJ_9.1.0/com.ibm.mq.pro.doc/q003460_.htm

IBM MQ MQI client component

- Installed independently on any compatible system
- Connects application and queue manager



Managing clients and client connections

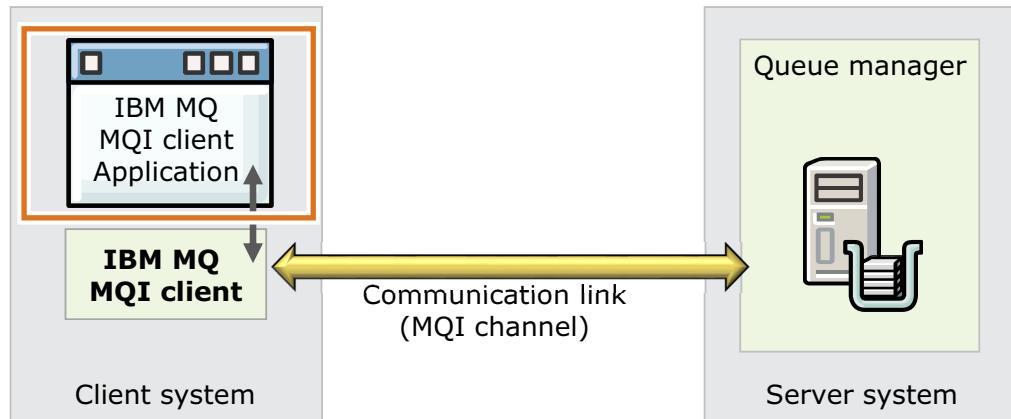
© Copyright IBM Corporation 2020

Figure 4-7. IBM MQ MQI client component

The IBM MQ MQI (messaging queue interface) client component is an installed component separate from the IBM MQ queue manager. It can be downloaded as part MQC91 of the Supportpac at: <https://www.ibm.com/support/pages/node/712701>. For local client usage on the IBM MQ server, the client can be installed as part of the IBM MQ queue manager installation.

IBM MQ application

- Languages: C, Visual Basic, COBOL, assembly, RPG, pTAL, and PL/I
- Uses message queue interface (MQI) calls
- Refer to IBM MQ Application Development course for details on creating IBM MQ applications



Managing clients and client connections

© Copyright IBM Corporation 2020

Figure 4-8. IBM MQ application

IBM MQ applications can be written using a variety of languages and on a variety of systems. All use the MQI to make calls to the queue manager. Nearly all MQI calls are available to MQI clients.

For more details about creating IBM MQ applications, take the *IBM MQ V9 Application Development* course and see:

www.ibm.com/support/knowledgecenter/SSFKSJ_9.1.0/com.ibm.mq.dev.doc/q022830_.htm.

MQI channel

- Links MQI client application to the IBM MQ server queue manager
- Bidirectional MQI channel
- How does the client communication with server?
 1. Application issues MQI calls
 2. Input parameters are sent to the queue manager
 3. Output parameters flow back to the application

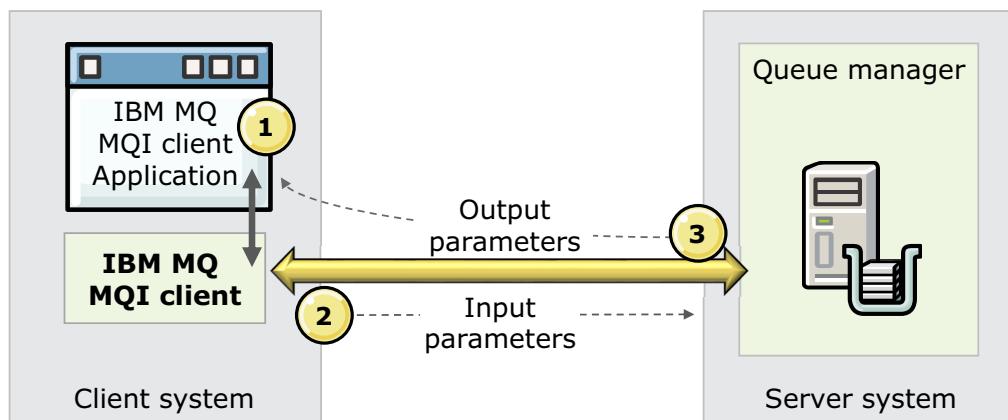


Figure 4-9. MQI channel

The MQI channel is the link between the IBM MQ client application and the IBM MQ queue manager. It is a single bidirectional link that allows the IBM MQ client to make MQI calls and send the input parameters to the queue manager and have the queue manager return the output parameters back to the client application. The input parameters of an MQI call flow in one direction on an MQI channel and the output parameters flow in the opposite direction.

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.

4.2. Client configurations

Client configurations

Managing clients and client connections

© Copyright IBM Corporation 2020

Figure 4-10. Client configurations

Defining 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 4-11. Defining connections between the server and client

To configure the communication links between IBM 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 4-12. Defining an MQI channel

Similar to a message channel, an MQI channel requires a definition at both ends of the channel. 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 IBM MQ listener must be running on the server system.

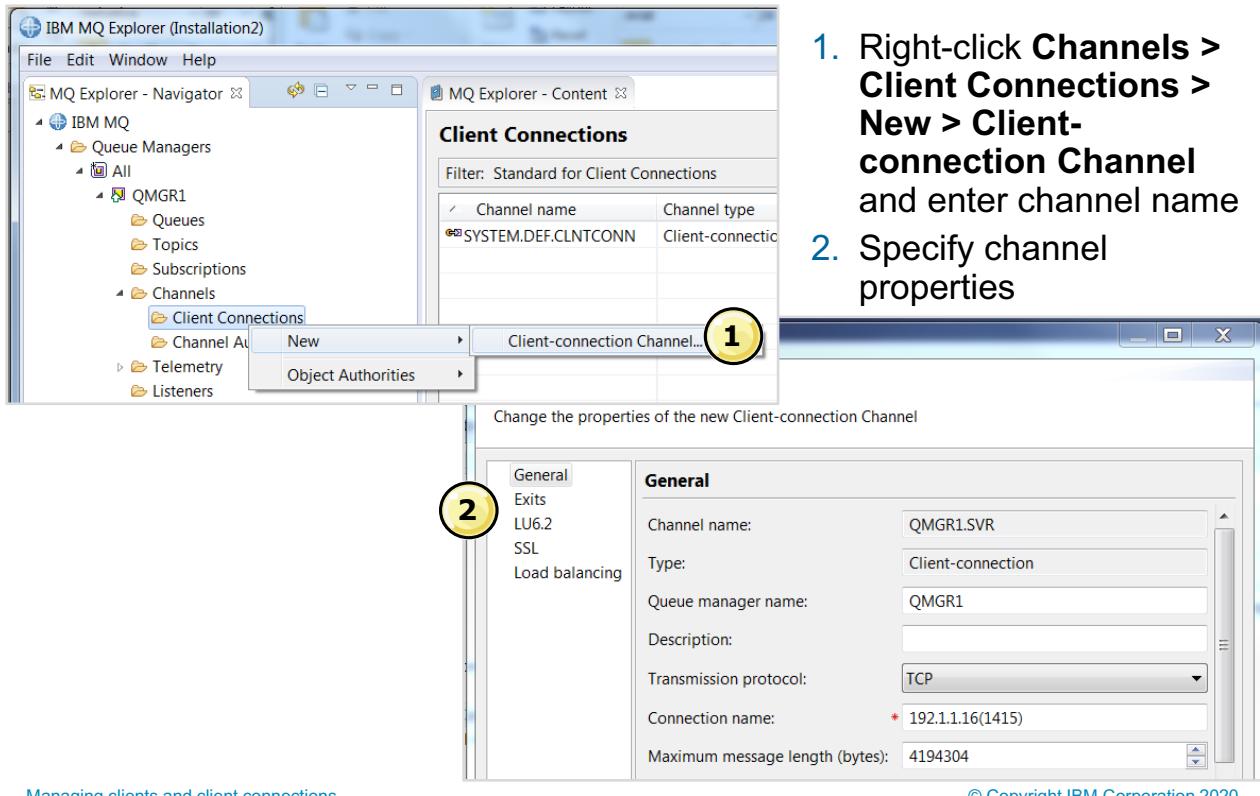
An MQI channel is bidirectional in terms of the flow of information. The input parameters of an MQI call flow in one direction and the output parameters in the reverse direction. It is not necessary to define two channels.

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 client-connection channels in IBM MQ Explorer



Managing clients and client connections

© Copyright IBM Corporation 2020

Figure 4-13. Defining client-connection channels in IBM MQ Explorer

You can define a client-connection channel on the IBM MQ server by using the **DEFINE CHANNEL** command or by using MQ Explorer.

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)

Managing clients and client connections

© Copyright IBM Corporation 2020

Figure 4-14. Client configuration methods

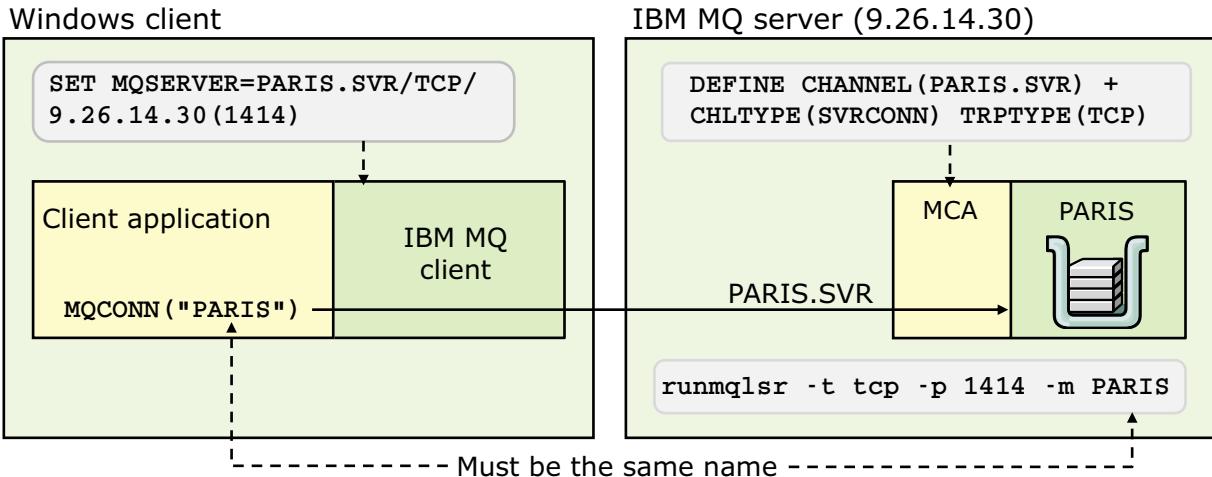
The figure lists the ways to configure clients connections:

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



- 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

Managing clients and client connections

© Copyright IBM Corporation 2020

Figure 4-15. 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, the MQSC `DEFINE CHANNEL` command defines a server connection (SVRCONN). The `runmqqlsr` command starts 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)
```

Managing clients and client connections

© Copyright IBM Corporation 2020

Figure 4-16. 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 IBM 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.

https://www.ibm.com/support/knowledgecenter/SSFKSJ_9.1.0/com.ibm.mq.con.doc/q016840_.htm

Method 3: Using a client channel definition table (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 (path or URL)
 - `MQCHLLIB` and `MQCHLTAB` environment variables
 - `mqclient.ini` file
 - Programmatically in the application

Figure 4-17. Method 3: Using a client channel definition table (CCDT)

Method 3 is a more complex method that uses the definition of a client connection and a server connection on the server system.

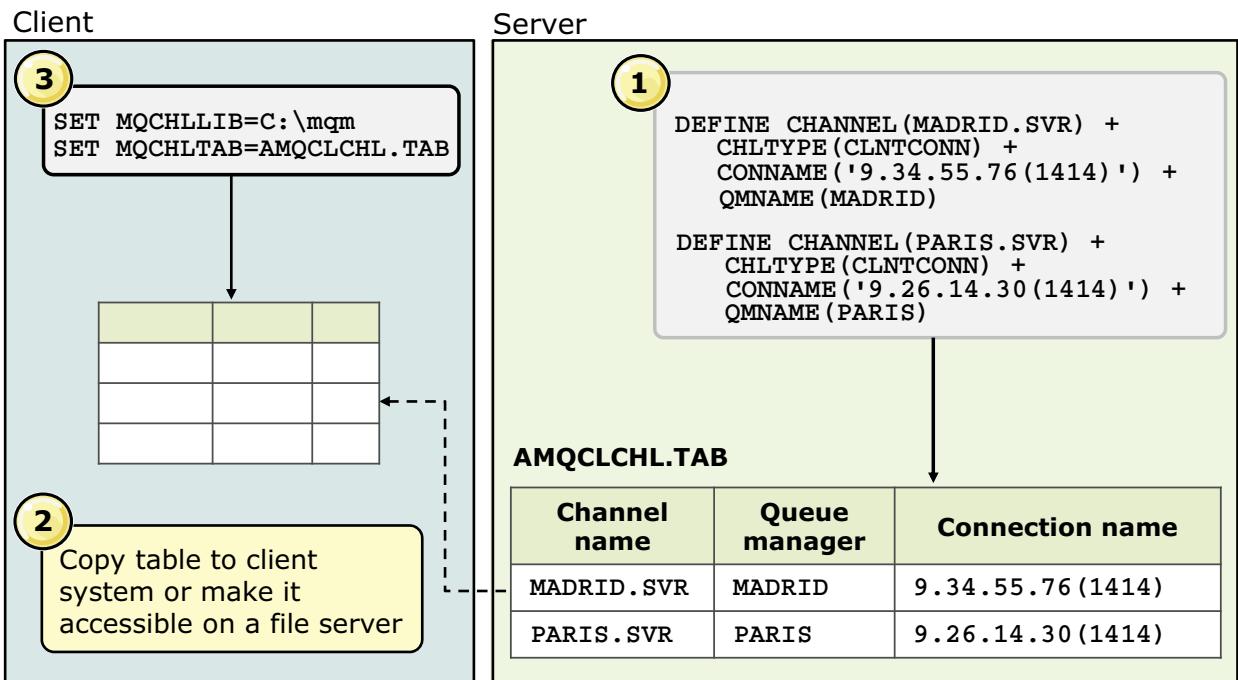
When you define a client connection on the server, it is stored in the client channel definition table (CCDT) in the file name AMQCLCHL.TAB. One table can contain client connection information for many queues managers, which gives the application a choice of client connections and queue managers.

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 IBM MQ client file.
- Locate a CCDT through a URL, either by programming, by using environment variables, or by using `mqclient.ini` file stanzas.

https://www.ibm.com/support/knowledgecenter/SSFKSJ_9.1.0/com.ibm.mq.con.doc/q016730_.htm

Using a CCDT example



Managing clients and client connections

© Copyright IBM Corporation 2020

Figure 4-18. 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.

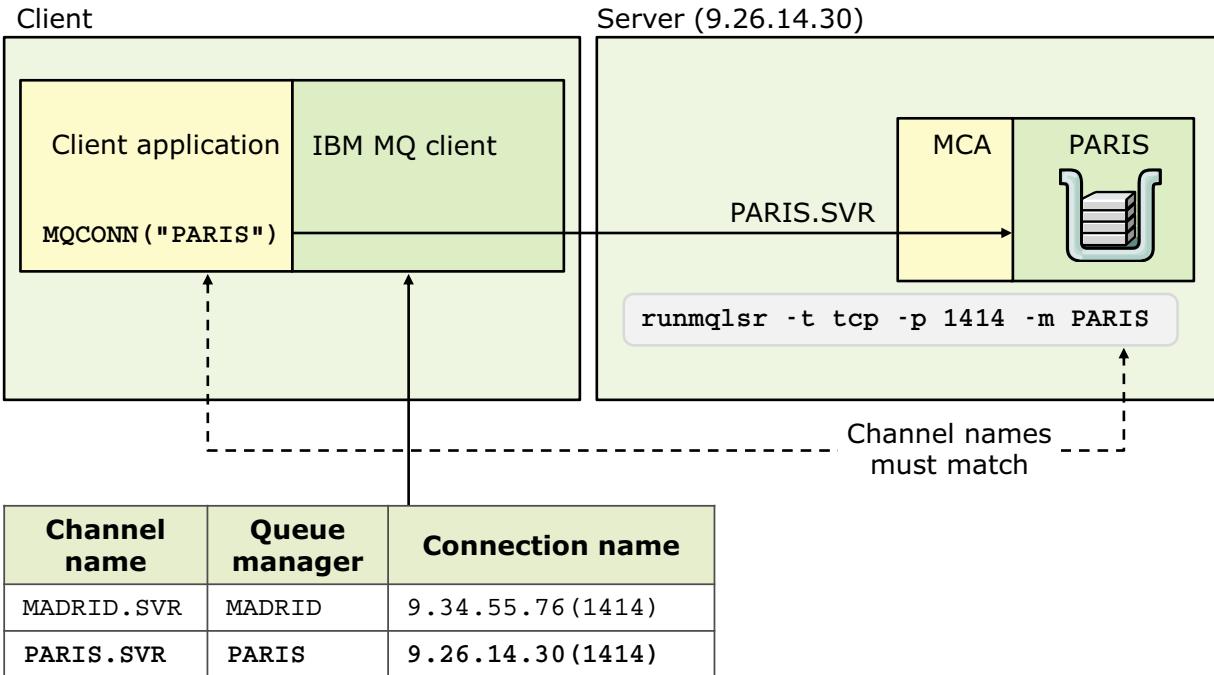
On Windows, the syntax is:

```
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
```

Accessing a CCDT



[Managing clients and client connections](#)

© Copyright IBM Corporation 2020

Figure 4-19. Accessing a CCDT

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 4-20. URL support for CCDT files

IBM MQ 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.

https://www.ibm.com/support/knowledgecenter/SSFKSJ_9.1.0/com.ibm.mq.con.doc/q016735_.htm

Precedence order for finding the CCDT

1. Application specifies connection details
 2. Application specifies CCDT URL location
 3. MQSERVER environment variable
 4. `mqclient.ini` file containing ServerConnectionParms
 5. MQCCDTURL environment variable
 6. MQCHLLIB and MQCHLTAB environment variables
 7. Attributes 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 4-21. Precedence order for finding the CCDT

This figure lists the order of precedence for a native client application to find a CCDT.

If the client application cannot find a CCDT, an error is reported in the error logs.

https://www.ibm.com/support/knowledgecenter/SSFKSJ_9.1.0/com.ibm.mq.con.doc/q016735_.htm

Weighted selection on client channels

- Client weight and affinity control distribution of messages
- Client weight in distribution
 - Zero values are used first, alphabetically
 - Nonzero values are weighted
- Affinity (subsequent connections by same client)
 - Preferred: Channels placed in same order of preference (if available)
 - None: Prior use not kept, weight distribution reapplied

Client channel definition table example

Group name	Channel name	Queue manager	Connection name	Client weight	Affinity
GRP1	LONDON.SVR	LONDON	9.45.31.124 (1414)	2	NONE
GRP1	MADRID.SVR	MADRID	9.34.55.76 (1414)	5	PREFERRED
GRP1	PARIS.SVR	PARIS	9.26.14.30 (1414)	3	NONE
GRP1	ROME.SVR	ROME	9.14.78.101 (1414)	0	PREFERRED

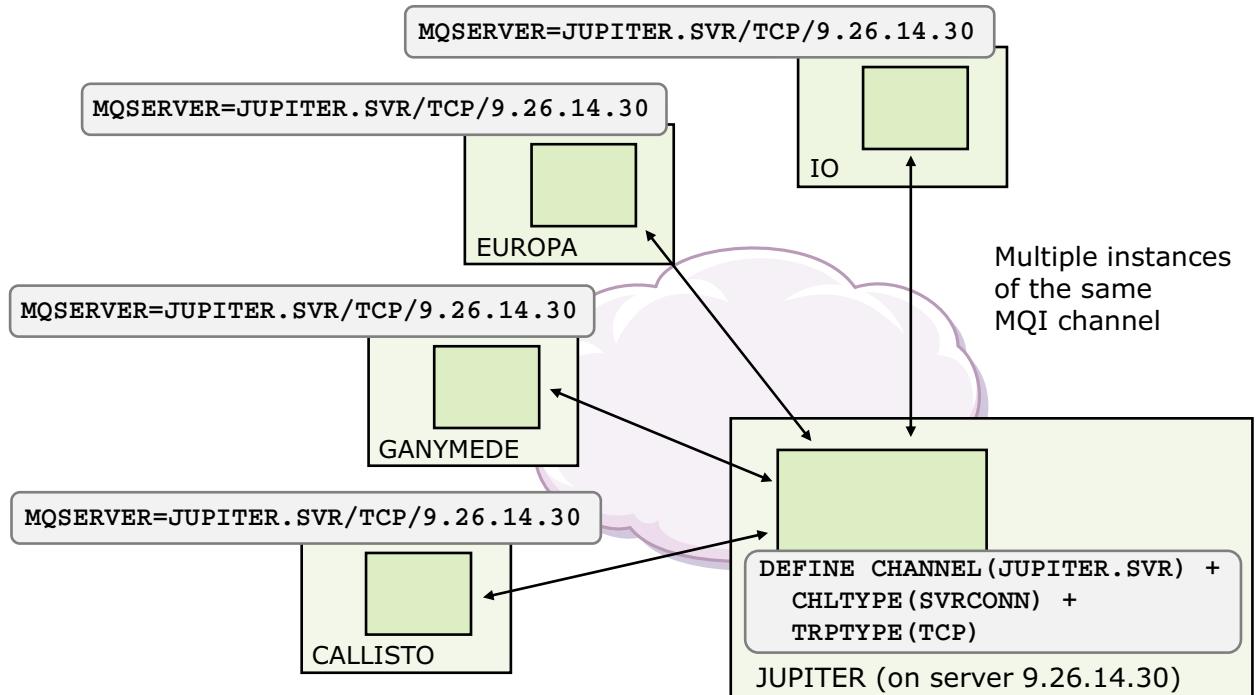
Figure 4-22. Weighted selection on client channels

IBM MQ has two attributes on the client-connection channel definition: client weighting and 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 used to determine percentages for distribution. In this example, Madrid is selected 50% of the time, and Paris and London are 30% and 20%.

The channel **AFFINITY** attribute is used so client applications that connect to the same queue manager multiple times uses the same client channels for each connection.

MQI channel instances



Managing clients and client connections

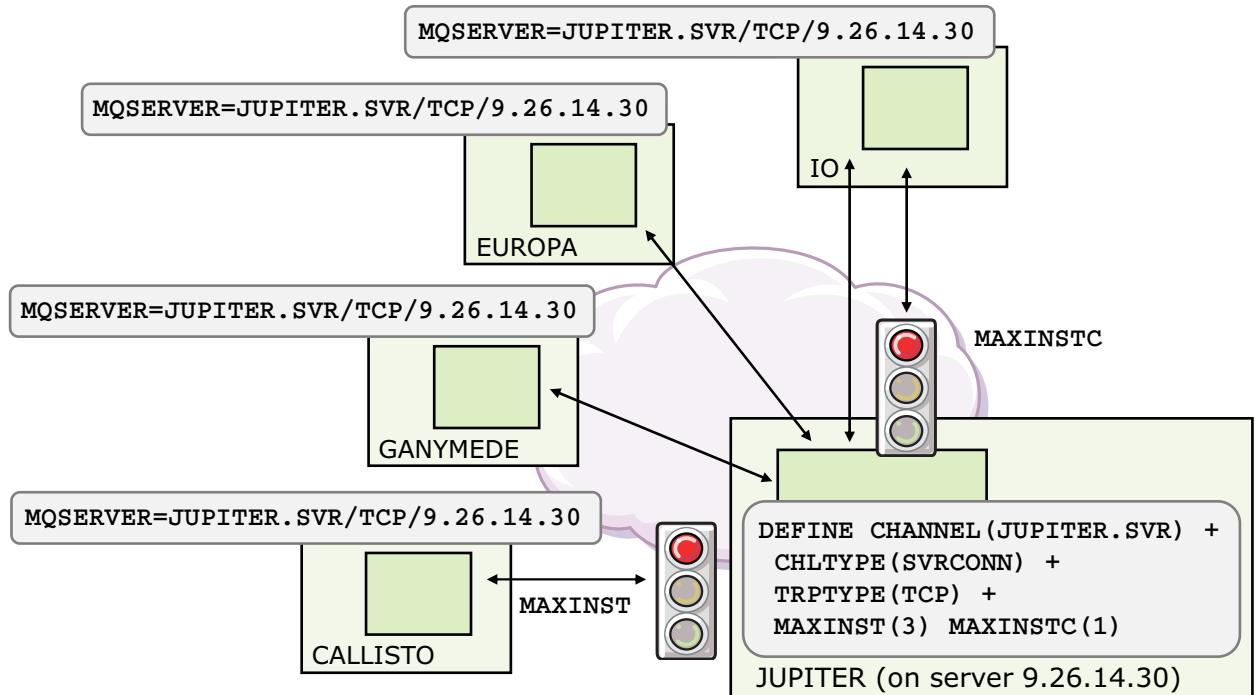
© Copyright IBM Corporation 2020

Figure 4-23. 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



Managing clients and client connections

© Copyright IBM Corporation 2020

Figure 4-24. 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 4-25. Auto-definition of channels

In IBM 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.

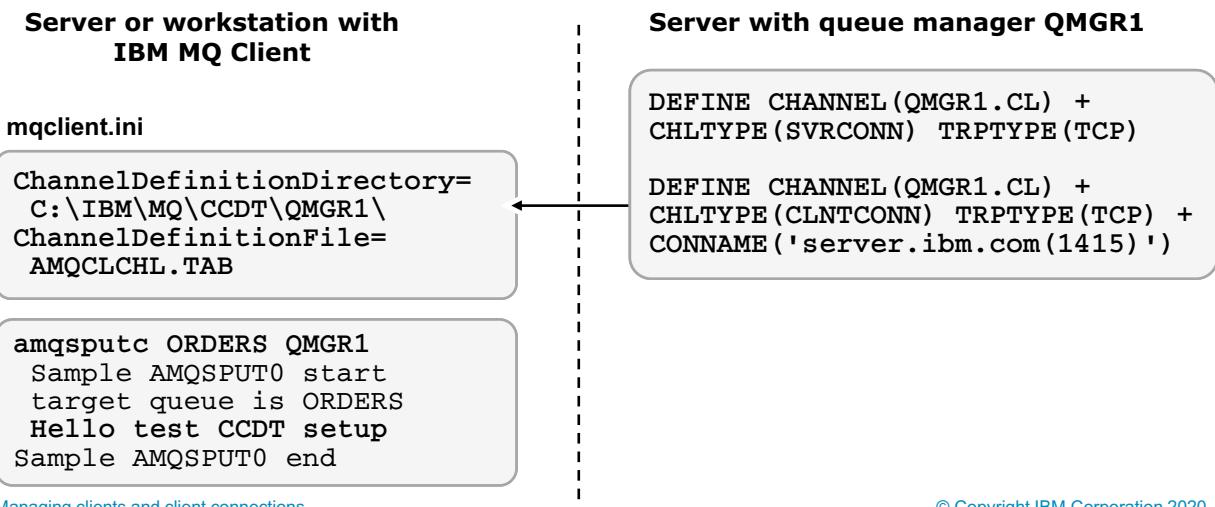
To enable the automatic definition of channels, the attribute **ChannelAutoDef** of the queue manager object must be set to MQCHAD_ENABLED. The corresponding parameter on the **ALTER QMGR** command is **CHAD(ENABLED)**.

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`



Managing clients and client connections

© Copyright IBM Corporation 2020

Figure 4-26. Testing client connectivity

You can test client connectivity by using the IBM MQ `amqsputc` sample program. The figure summarizes the steps for defining and testing an IBM MQ client connection to an IBM 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, re-enable channel authentication:

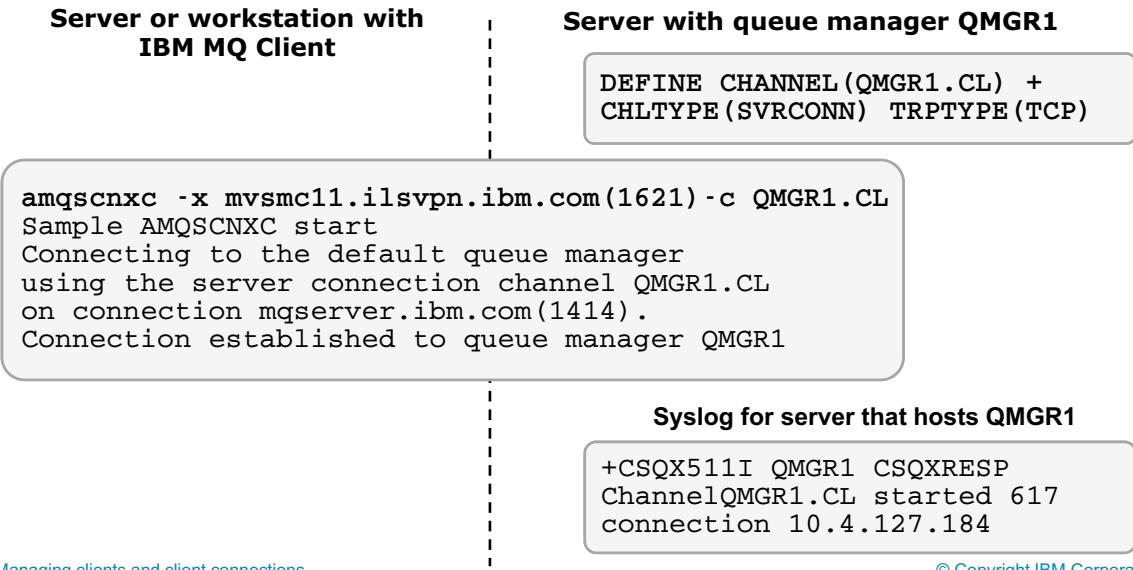
```
ALTER QMGR CHLAUTH (ENABLED)
```



Testing client-to-server connectivity

- Requirements:

- SVRCONN on the IBM MQ server
- IBM MQ client is installed on client computer
- IBM MQ sample client program to test connectivity: `amqscnxc`



Managing clients and client connections

© Copyright IBM Corporation 2020

Figure 4-27. Testing client-to-server connectivity

You can also use a sample program to simulate a connection by using an MQCONN call that is sent from the IBM 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.

4.3. Client modes for MQSC

Client modes for MQSC

Managing clients and client connections

© Copyright IBM Corporation 2020

Figure 4-28. Client modes for MQSC

MQSC client modes

- Client-local mode (-n) offers a limited set of MQSC commands to manage client channel definition files
- Client-connect mode (-c) connects to a remote queue manager by using a client channel and issues escaped PCF commands

Managing clients and client connections

© Copyright IBM Corporation 2020

Figure 4-29. MQSC client modes

As shown in the figure, IBM MQ supports two client modes.

Client-local mode sends MQSC commands to a queue manager through either a local or a client connection.

Client-connect mode sends MQSC commands against the client channel table without requiring a connection to a queue manager.

MQSC client-local mode (-n)

- Modifies the `runmqsc` command to not connect to a queue manager
 - All other command parameters must be omitted
 - Requires that the client libraries are installed
- MQSC commands that are entered in this mode are limited to managing the local channel definition file
- Limited to the following MQSC commands:
 - ALTER, DEFINE, DELETE, DISPLAY AUTHINFO (of the type CRLLDAP or OCSP only)
 - ALTER, DEFINE, DELETE, DISPLAY CHANNEL (of the type CLNTCONN only)

Figure 4-30. MQSC client-local mode (-n)

The client-local mode is started by adding the `-n` option to the `runmqsc` command. If this parameter is specified, all other command parameters must be omitted, otherwise an error message is generated.

Client-local mode requires the client libraries. If the client libraries are not installed on the client computer, an error message is generated.

MQSC commands that are entered in client-local mode are limited to managing the local channel definition file. This file is located through the `MQCHLLIB` and `MQCHLTAB` environment variables, or the default values if not defined.

MQSC client-connect mode (-c)

- Modifies the `runmqsc` command to connect to a queue manager by using a client connection
- Client channel definitions that are used to connect to the queue manager are located by using the following environment variables in this order of precedence: MQSERVER, MQCHLLIB, and MQCHLTAB
- Requires that the client is installed

Managing clients and client connections

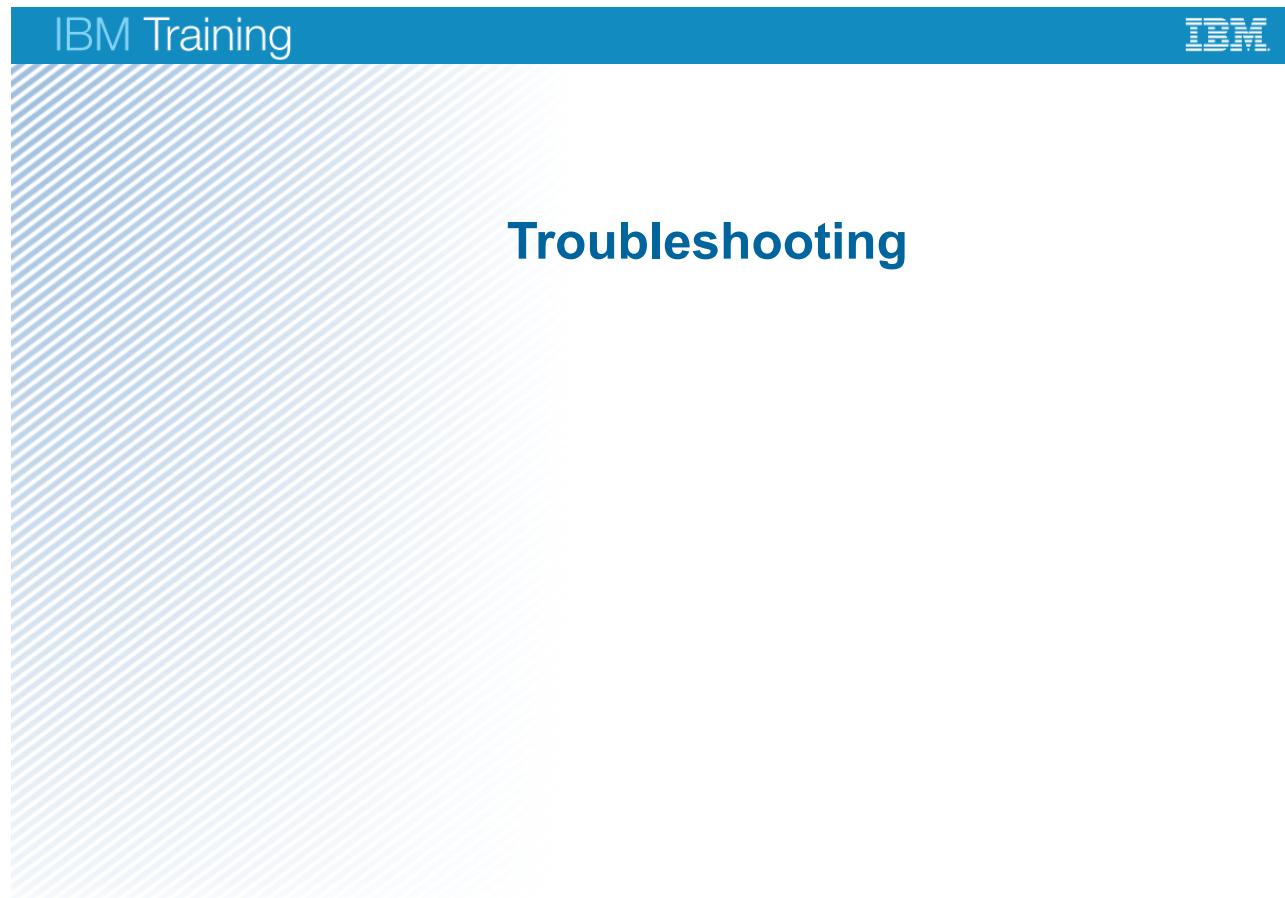
© Copyright IBM Corporation 2020

Figure 4-31. *MQSC client-connect mode (-c)*

The `-c` option modifies the `runmqsc` command to connect to a queue manager by using a client connection. The client channel definitions that are used to connect to the queue manager are located by using the following environment variables in this order of precedence: MQSERVER, MQCHLLIB, and MQCHLTAB.

This option requires the client libraries. If the client libraries are not installed on the client computer, an error message reports that client libraries are missing.

4.4. Troubleshooting tips



Managing clients and client connections

© Copyright IBM Corporation 2020

Figure 4-32. Troubleshooting tips

Troubleshooting IBM MQ channels

- Listener problems
- Channel configuration problems
- Network connectivity problems
- In-doubt channels

Managing clients and client connections

© Copyright IBM Corporation 2020

Figure 4-33. Troubleshooting IBM MQ channels

Most problems with IBM MQ channels can be identified as listener problems, channel configuration problems, network connectivity problems, or in-doubt channels. Tips and techniques for identifying and fixing each of these problems are described in this topic.

Listener problems

- If an IBM MQ object, check it is configured correctly and running
- Verify that the listener process is active and listening
- Check network status to ensure that the listener process is active on the correct port
- Verify that the queue manager is running

Managing clients and client connections

© Copyright IBM Corporation 2020

Figure 4-34. Listener problems

This figure lists the steps to take to identify listener problems.

First, make sure that the listener is running. For example, list the processes on your system and make sure that the `runmqlsr` process is present.

You can also get listener status by entering the following MQSC commands:

```
DISPLAY LISTENER(B.LISTENER) ALL  
DISPLAY LSSTATUS(B.LISTENER) ALL  
START LISTENER(B.LISTENER)
```

On Windows, you can run the `netstat -an` command to get a list of active ports and verify that the IBM MQ listener port is not already in use by another program.

On UNIX and Linux, you can use the `lsof` program to determine whether another application is using the port. For example, type: `lsof -i tcp:1414`

Channel configuration problems

- Confirm the channel definitions on the sending and receiving side match
- Ensure that channel names match exactly
- Ensure that the correct channel type is on both sides of the connection
- Verify the port number
 - Explicitly define the default port number of 1414
- Verify that the receiving channel is enabled

[Managing clients and client connections](#)

© Copyright IBM Corporation 2020

Figure 4-35. Channel configuration problems

Make sure that the channel definitions on the sending and receiving side match. Use the MQSC DISPLAY CHANNEL and DISPLAY CHSTATUS commands with the ALL option to display the definitions and the channel status on both sides of the channel.

Channel names must match exactly. Be sure to use single quotation marks around lowercase or mixed-case channel names in MQSC. For example, type `DEFINE CHANNEL('a.to.b')`. Or better yet, always use uppercase characters for channel names.

Make sure that you used the correct channel type on both sides of the channel. For example, SENDER/RECEIVER, SERVER/REQUESTER, and CLUSSDR/CLUSRCVR channels are compatible, but a SENDER/SENDER pair is not valid.

If the channel definition relies on the default port of 1414, try specifying the port number explicitly. The default of 1414 does not apply if you define a TCP service that is called "MQSeries" on a different port number.

Make sure that the receiving channel is not disabled. If you stopped a receiver channel with the `STOP CHANNEL(A.TO.B)` command instead of the `STOP CHANNEL(A.TO.B) STATUS(INACTIVE)` command, then that channel is disabled and the sending side cannot start it. In this case, you must run `START CHANNEL(A.TO.B)` against the receiver channel to re-enable it, and then enter `START CHANNEL(A.TO.B)` on the sending side to get the channel to run.

Network connectivity problems

- Use the TCP `ping` command to test network connectivity between the two systems
- Use Telnet to test the IBM MQ port on the remote system
Example: `telnet server1 1414`
- Use the MQSC PING CHANNEL command to test a channel by sending a special message to the remote queue manager and checking that the data is returned
- If the channel definition uses host names instead of IP addresses, make sure that they resolve to the expected address
- If a firewall exists between the systems, make sure that it is configured to allow IBM MQ traffic
- On server-connection channels, set **SHARECNV** to 1 or higher

Figure 4-36. Network connectivity problems

This figure lists some tips and techniques for finding common network connectivity problems.

You can use the TCP/IP `ping` command, the MQSC PING CHANNEL command, or Telnet to test network connectivity. For example, to use Telnet to test the IBM MQ port on the remote system, type: `telnet server1 1414`

If the channel uses host names instead of IP address, make sure that they resolve to the expected address. For example, use the `nslookup` or `dig` commands, where available, to ensure that the domain name service (DNS) servers are providing the right address information. Test with IP addresses in your channel definition if you are not sure whether the DNS is causing the problem.

If the network includes a firewall, make sure that it is configured to allow IBM MQ traffic. If necessary, you can use the LOCALADDR attribute on sending channels to ensure that the IBM MQ channel uses ports that the firewall allows.

On server-connection channels, set **SHARECNV** to 1 or higher to ensure that MQ sends heartbeats whenever the channel is otherwise idle. Receiving the heartbeat prevents firewalls from ending the connection due to inactivity.

In-doubt channels

- IBM MQ channels can show an “in doubt” status when the sending side does not know whether a previous batch of messages was received
- In most cases, starting the channel is sufficient to clear this status
- Might be necessary to check manually whether the receiving side received the last batch of messages

Managing clients and client connections

© Copyright IBM Corporation 2020

Figure 4-37. In-doubt channels

An in-doubt channel is a channel that is in doubt with a remote channel about which messages were sent and received. This figure lists some tips for identifying in-doubt channels.

In-doubt channel problems are typically resolved automatically. Even when communication is lost, and a channel is placed in doubt with a message batch at the sender with receipt status unknown, the situation is resolved when communication is reestablished.

You can, when necessary, resynchronize the channel manually. You can enter MQSC commands on the sending side to resolve in-doubt channels. For example:

```
STOP CHANNEL (A.TO.B)
RESOLVE CHANNEL (A.TO.B) ACTION(BACKOUT or COMMIT)
START CHANNEL (A.TO.B)
```

The **RESOLVE CHANNEL** command requests a channel to commit or back out in-doubt messages. This command is used when the other end of a link fails during the confirmation period, and for some reason it is not possible to reestablish the connection. In this situation, the sending end remains in doubt whether the messages were received. Any outstanding units of work must be resolved by being backed out or committed.

The **RESOLVE CHANNEL** command can be used only for sender (SDR), server (SVR), and cluster-sender (CLUSSDR) channels.

MQSC connection and channel status commands

- **DISPLAY CONN**

- Display connection information about the applications that are connected to the queue manager
- Use the results to identify applications with long-running units of work

- **DISPLAY CHSTATUS**

- Display the status of one or more channels

Figure 4-38. MQSC connection and channel status commands

You can use the **DISPLAY CONN** and **DISPLAY CHSTATUS** commands to get more information about a channel.

Channel monitoring

- To enable, set **MONCHL** attribute on the channel
 - Set to **LOW**, **MEDIUM**, or **HIGH** to control at the channel level
 - Set to **QMGR** and then set **MONCHL** value in the queue manager to control at queue manager level
- To display monitoring information for a channel, use IBM MQ Explorer or the MQSC command **DISPLAY CHSTATUS () MONITOR**

Managing clients and client connections

© Copyright IBM Corporation 2020

Figure 4-39. Channel monitoring

If you require real-time monitoring information for a channel, you can set the monitoring level either at the object level or at the queue manager level.

You can set channel monitoring to **LOW**, **MEDIUM**, or **HIGH**.

- **LOW** measures a small sample of the data, at regular intervals. Use this level for objects that process a high volume of messages.
- **MEDIUM** measures a sample of the data, at regular intervals. Use this level for most objects.
- **HIGH** measures all data, at regular intervals. Use this level for objects that process only a few messages per second, on which the most current information is important.

You can use the IBM MQ Explorer or the **DISPLAY CHSTATUS** command with the **MONITOR** option to show the channel monitoring information.

Channel monitoring example

- Sender channel QM1.TO.QM2 **MONCHL** attribute is set to the default value of **QMGR**
- Queue manager that owns the channel **MONCHL** attribute is set to **MEDIUM**

```
DISPLAY CHSTATUS (QM1.TO.QM2) MONITOR
  CHSTATUS (QM1.TO.QM2)
  XMITQ (Q1)
  CONNAME (127.0.0.1)
  CURRENT CHLTYP (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 4-40. Channel monitoring example

This figure shows an example of using the **DISPLAY CHSTATUS** command to display the monitoring information for a channel that is named QM1.TO.QM2. The example assumes that the monitoring level is set to **MEDIUM**.

Troubleshooting IBM MQ clients

- Ensure that the client application environment variables are set to the correct values
- Examine the client channel definition file and verify the settings
- Check the client application code to ensure that it is using the correct client-channel definition attributes
- Check the security attributes of the server-connection channel to which the client connects
- Check the error logs on the client and queue manager servers

Managing clients and client connections

© Copyright IBM Corporation 2020

Figure 4-41. Troubleshooting IBM MQ clients

This figure lists some tips for troubleshooting IBM MQ clients.

You can check the client channel definition table without connecting to the queue manager by using the `runmqsc -n` command to run in client-local mode.

To display information about the client chMQSC channel configuration, type:

```
DISPLAY CHANNEL(*) CHLTYPE(CLNTCONN) ALL
```

Be sure to check the security attributes of the server-connection channel.

If client application set its connection parameters programmatically, rather than relying on any external files or environment variables, check the client application code to ensure that it is using the correct client-channel definition attributes

Unit summary

- Describe the components of an MQI client connection
- Describe the various ways to connect a client to a queue manager
- Describe the client modes that MQSC supports
- Use troubleshooting tools and techniques to monitor and manage clients and connections

Review questions

1. What do channels with a client weight value of zero indicate?
 - A. Use first, in alphabetical order
 - B. Use last, in alphabetical order
 - C. Disable this channel

2. The client connection MQI channel must have how many channel definitions:
 - A. One: It is a bidirectional channel
 - B. Two: One inbound and one outbound channel
 - C. Two: Two bidirectional channels, one for data and one for commands
 - D. Three: One inbound, one outbound, and one channel for encryption



Managing clients and client connections

© Copyright IBM Corporation 2020

Figure 4-43. Review questions

Write your answers here:

1.

2.

Review answers

1. What do channels with a client weight value of zero indicate?
 - A. Use first, in alphabetical order
 - B. Use last, in alphabetical order
 - C. Disable this channel

The answer is A.

2. The client connection MQI channel must have how many channel definitions:
 - A. One: It is a bidirectional channel
 - B. Two: One inbound and one outbound channel
 - C. Two: Two bidirectional channels, one for data and one for commands
 - D. Three: One inbound, one outbound, and one channel for encryption

The answer is A.



Exercise: Connecting an IBM MQ client

Managing clients and client connections

© Copyright IBM Corporation 2020

Figure 4-45. Exercise: Connecting an IBM MQ client

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



Figure 4-46. Exercise introduction

Unit 5. Advanced IBM MQ client features

Estimated time

01:00

Overview

In this unit, you learn more about IBM MQ client advanced features.

How you will check your progress

- Review questions

Unit objectives

- Explain how to use the extended transactional client
- Manage client performance by sharing conversations
- Describe the performance impact of using read ahead
- Outline reasons for using asynchronous put

Advanced IBM MQ client features

© Copyright IBM Corporation 2020

Figure 5-1. Unit objectives

Topics

- Extended transactional client
- Sharing conversations
- Read ahead
- Asynchronous put

Advanced IBM MQ client features

© Copyright IBM Corporation 2020

Figure 5-2. Topics

Extended transactional client

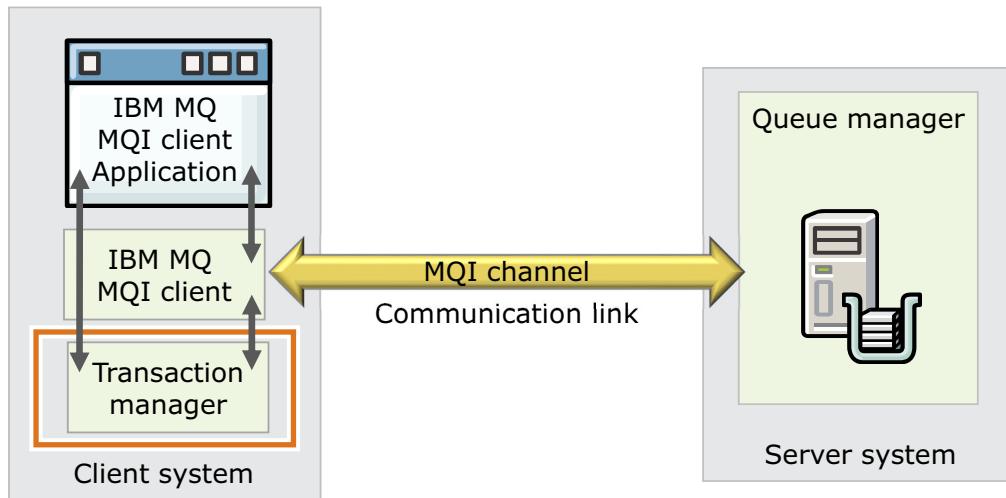
Advanced IBM MQ client features

© Copyright IBM Corporation 2020

Figure 5-3. Extended transactional client

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
 - External transaction manager runs on client system



Advanced IBM MQ client features

© Copyright IBM Corporation 2020

Figure 5-4. Extended transactional client

An MQ extended transactional client is an IBM MQ MQI client with some additional functions that can 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 transaction. The queue manager to which a client application is connected cannot manage the transaction. The queue manager can act only as a resource manager, not as a transaction manager. The client application can commit or back out the transaction by using only the API provided by the external transaction manager.

For more information about the extended transactional client, see:

www.ibm.com/support/knowledgecenter/en/SSFKSJ_9.1.0/com.ibm.mq.pro.doc/q003510_.htm

Units of work and sync points

- Unit of work (also known as transaction)
 - One or more instructions to one or more resources
 - All instructions must be successful to complete
 - Any failure requires all successful instructions to *backout*
 - To *commit* the transaction is to finalize all instructions
 - Two types: local and global
- Local units of work
 - Only resources that are updated are in the queue manager
 - Single-phase commit
- Global units of work
 - Other resource managers are also being updated
 - Two phase commit
- Large sets of instructions often have logical data change synchronization points (sync points)
- Each set of instructions between sync points is a unit of work

Figure 5-5. Units of work and sync points

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.

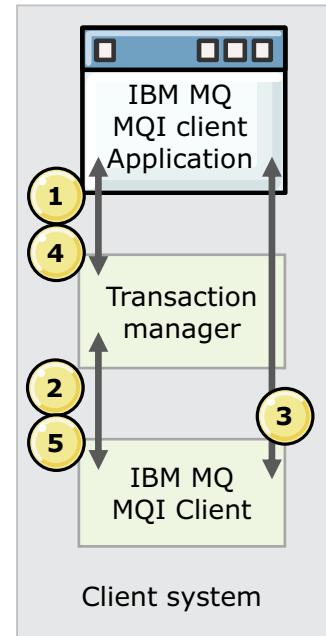
A global unit of work is one in which the resources of other resource managers and the resources of a queue manager are updated.

Sync points separate units of work.

https://www.ibm.com/support/knowledgecenter/SSFKSJ_9.1.0/com.ibm.mq.pro.doc/q023320_.htm

Control flow

- Control flow of a global unit of work coordinated by an external transaction manager:
 1. Application tells the transaction manager to start a transaction
 2. Transaction manager tells known resource managers about the current transaction
 3. Application issues calls to resource managers associated with the current transaction
 4. Application issues a commit or backout request to the transaction manager
 5. The transaction manager completes the transaction with each resource manager



[Advanced IBM MQ client features](#)

© Copyright IBM Corporation 2020

Figure 5-6. Control flow

Control flow of a global unit of work coordinated by an external transaction manager:

1. Application tells the transaction manager to start a transaction
2. Transaction manager tells known resource managers about the current transaction
3. Application issues calls to resource managers associated with the current transaction
4. Application issues a commit or backout request to the transaction manager
5. The transaction manager completes the transaction by issuing the appropriate calls to each resource manager, typically using two-phase commit protocols

In steps 2, 3 and 5, the MQI client is shown as the only resource manager. There can be additional resource managers to which the application makes calls.

https://www.ibm.com/support/knowledgecenter/SSFKSJ_9.1.0/com.ibm.mq.pro.doc/q023600_.htm

5.1. Sharing conversations

Sharing conversations

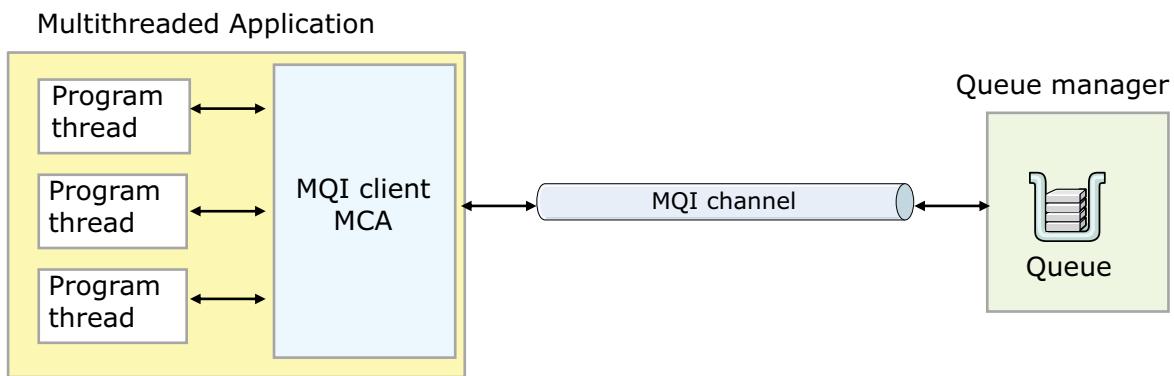
Advanced IBM MQ client features

© Copyright IBM Corporation 2020

Figure 5-7. Sharing conversations

Sharing conversations overview

- MQI channels
 - Bidirectional communications link between an MQI client application and a queue manager
 - Used to transfer MQI calls and responses between MQI client applications and queue managers
- A single MQI channel can support multiple threads



Advanced IBM MQ client features

© Copyright IBM Corporation 2020

Figure 5-8. Sharing conversations overview

The two-way channels and multiplexing support in IBM MQ allows an application to have multiple threads that share a client-channel connection.

In this context, an application means a single process. An example would be a Java or C program that runs multiple threads, with each thread that is processing a message. Separate processes, for example, two separate Java or C programs, cannot share a client channel.

Sharing conversation requirements

- Both ends of the client/server connection must be configured for sharing conversations
- Client channel **Sharing conversation** value must match
 - the value that is supplied on the client connection call or
 - the value in the client channel definition table (CCDT)
- Sharing conversations limit on the server side is not exceeded

Advanced IBM MQ client features

© Copyright IBM Corporation 2020

Figure 5-9. Sharing conversation requirements

Sharing conversations requires configuration at the client-connection and server-connection ends of the channel. This figure summarizes the requirements for sharing conversations.

The MQCD structure contains the parameters that control execution of a channel. It is passed to each channel exit that is called from an MCA.

SHARECNV channel property

- Can be configured on server-connection and client-connection channels
- Applications can set **SHARECNV** on the IBM MQ connect call or by setting the **sharingConversations** field of the **MQEnvironment** and **MQChannelDefinition** classes in Java
- If you do not need to share conversations:
 - Set **SHARECNV** to 1 to eliminate contention and use features such as bidirectional heartbeats and performance
 - If you have existing applications that do not run correctly when you set **SHARECNV** to 1 or greater, set **SHARECNV** to 0

[Advanced IBM MQ client features](#)

© Copyright IBM Corporation 2020

Figure 5-10. SHARECNV channel property

The IBM MQ sharing conversations option is configured on the client-connection channel. The client-connection can be configured dynamically by using the **MQSERVER** environment variable. It can also be configured by the application in the **MQCONN** call, by using the IBM MQ Java and JMS classes, or by using a pre-connect exit.

You use the **SHARECNV** channel property to specify the maximum number of conversations that can be shared over a particular TCP/IP client channel instance.

By default, the **SHARECNV** value for a new queue manager is set to 10 in any **SRVCONN** channel definitions. The **SYSTEM.DEF.SRVCONN** channel definition, which is used to supply default values when a new **SRVCONN** channel is defined, also has **SHARECNV** set to 10. You can change this default if needed.

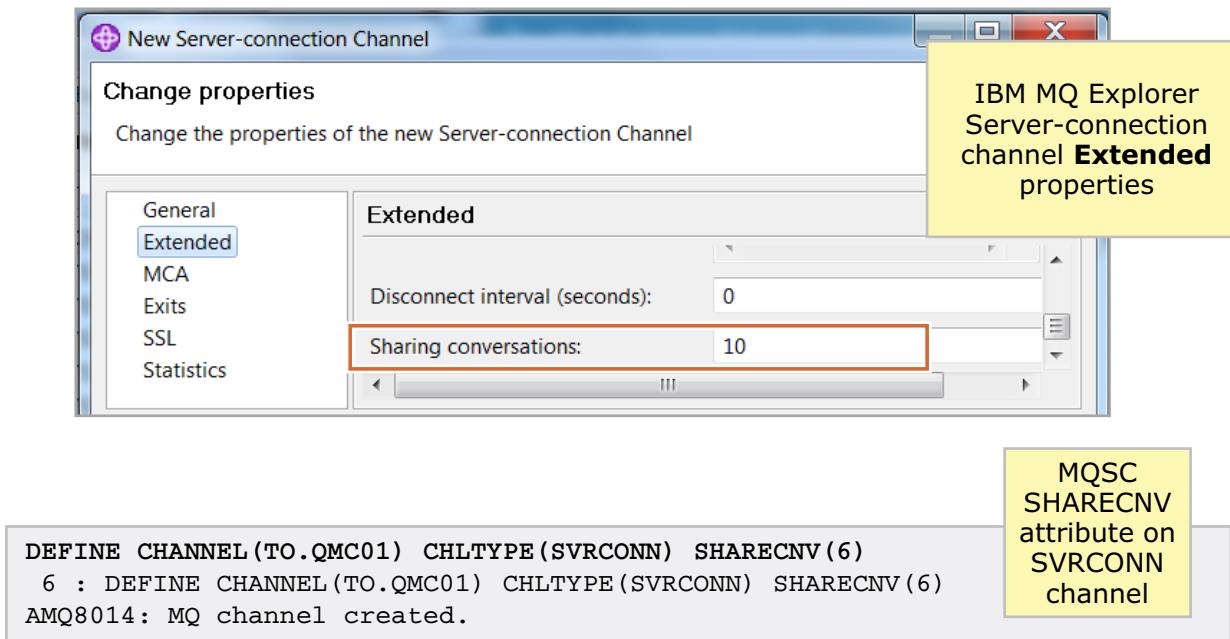
You set the **SHARECNV** property to a numeric value.

- **SHARECNV(0)**. This value specifies no sharing of conversations over a TCP/IP socket. Use a value of 0 if you have WebSphere MQ V6 client applications.
- **SHARECNV(1)**. This value allows one conversation on the channel instance. Client heartbeats and read ahead are available, and channel quiescing is more controllable. **SHARECNV(1)** is the suggested value for IBM MQ if you do not need to share conversations.

- SHARECNV(2) to SHARECNV(999999999). Each of these values specifies the number of shared conversations. If the client-connection SHARECNV value does not match the server-connection SHARECNV value, then the lowest value is used. The default value is SHARECNV(10), which specifies 10 threads to run up to 10 client conversations per channel instance. However, distributed servers might exhibit performance problems with SHARECNV channels that can be eased by setting SHARECNV(1) wherever possible.



Sharing conversations configuration: Server-connection



Advanced IBM MQ client features

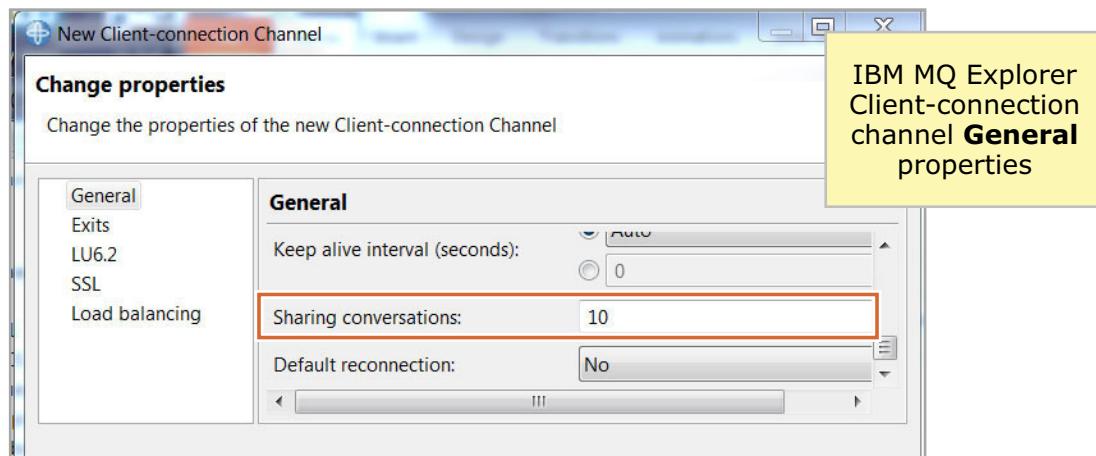
© Copyright IBM Corporation 2020

Figure 5-11. Sharing conversations configuration: Server-connection

You can use MQSC commands and IBM MQ Explorer to configure sharing conversations on the server-connection (SVRCNN) channel.

In IBM MQ Explorer, the **Sharing conversations** property is on the **Extended** properties page.

Sharing conversations configuration: Client-connection



```
DEFINE CHANNEL (TO.QMC01) CHLTYPE(CLNTCONN) SHARECNV(4) CONNAME(10.1.2.3)
 8 : DEFINE CHANNEL (TO.QMC01) CHLTYPE(CLNTCONN) SHARECNV(4)
CONNAME(10.1.2.3)
AMQ8014: MQ channel created.
```

MQSC
SHARECNV
attribute on
CLNTCONN
channel

Advanced IBM MQ client features

© Copyright IBM Corporation 2020

Figure 5-12. Sharing conversations configuration: Client-connection

You can use MQSC commands `DEFINE CHANNEL` and `ALTER CHANNEL` or IBM MQ Explorer to configure sharing conversations on the client-connection (SVRCONN) channel.

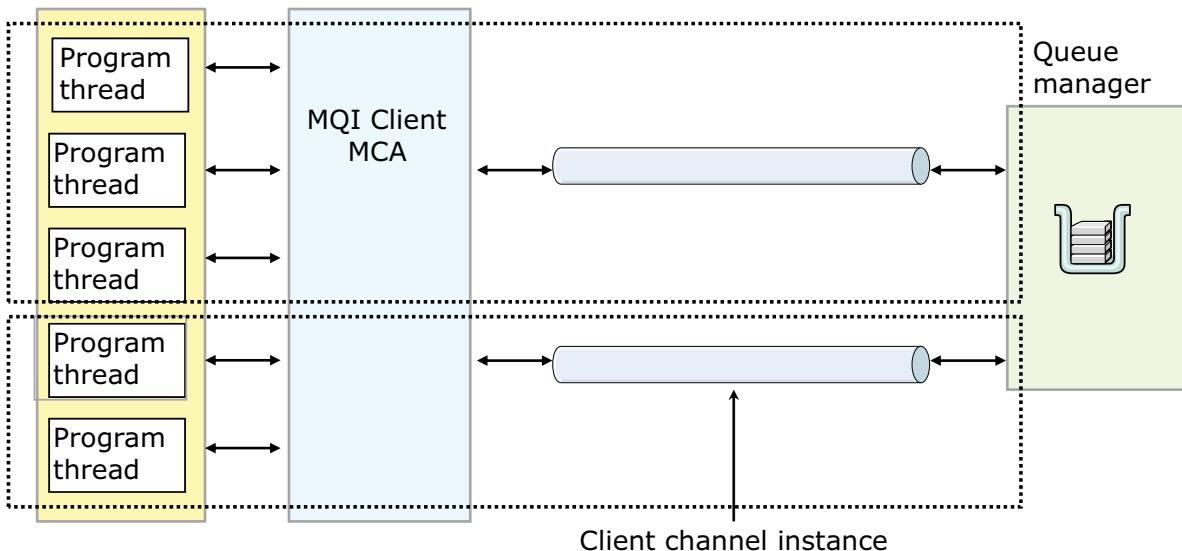
In IBM MQ Explorer, the **Sharing conversations** property is on the **General** properties page for client-connection channels.

Sharing conversations example

```
DEFINE CHANNEL (SALES)
  TYPE (CLNTCONN) SHARECNV (3)
```

```
DEFINE CHANNEL (SALES)
  TYPE (SVRCONN) SHARECNV (5)
```

Multithreaded Application



Advanced IBM MQ client features

© Copyright IBM Corporation 2020

Figure 5-13. Sharing conversations example

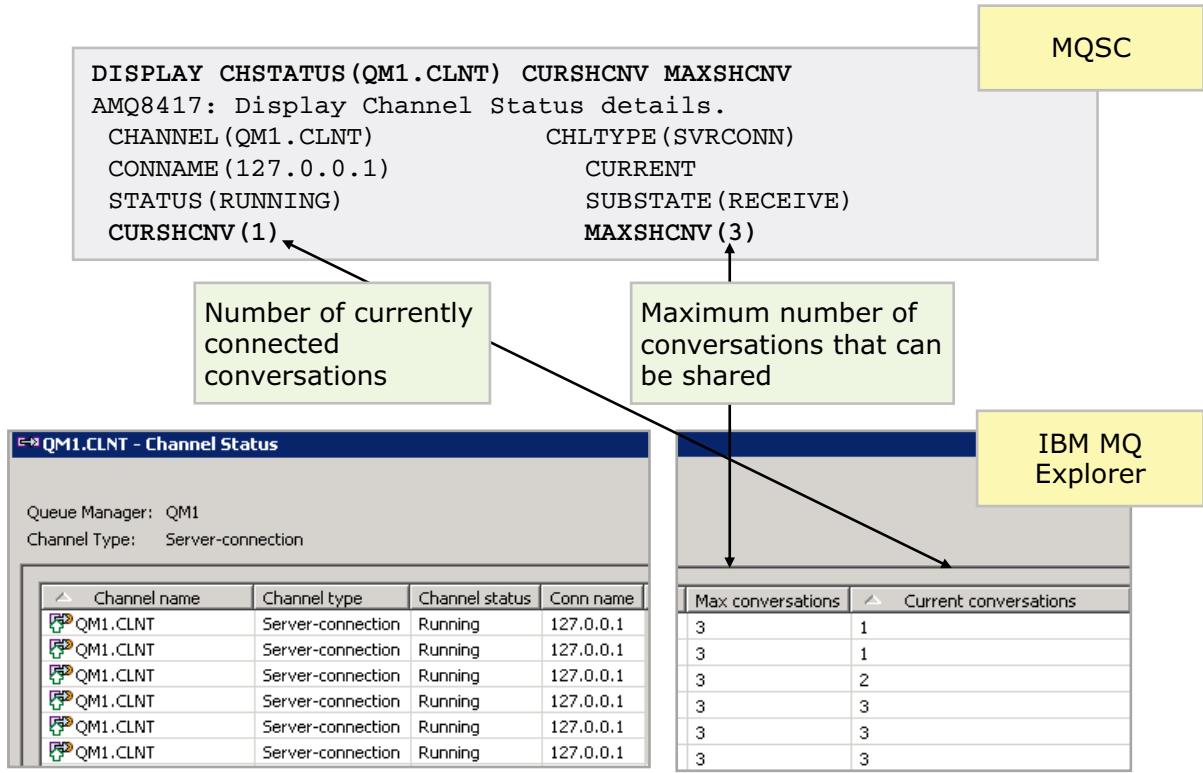
The example shows that the client-channel definition SHARECNV attribute is set to 3. The corresponding server-connection channel definition SHARECNV attribute is set to 5.

Because the channel SHARECNV values are different, the lowest value applies. So in this example, the maximum number of threads that can share a single client channel is 3. The figure shows three threads that share one client-channel instance, and another two threads in the same application that share a different client-channel instance.

You cannot specify the distribution of the number of threads evenly over client channels.



Sharing conversations channel status fields



Advanced IBM MQ client features

© Copyright IBM Corporation 2020

Figure 5-14. Sharing conversations channel status fields

Fields that are associated with sharing conversations are shown when you view channel status on server-connection type channels for a queue manager. The sharing conversations values can be viewed by using the `MQSC DISPLAY CHSTATUS` command or the IBM MQ Explorer **Channel Status** view.

- The **Current conversations** (`CURSHCNV`) property specifies the number of conversations that are currently connected to the queue manager on the channel instance.
- The **Max conversations** (`MAXSHCNV`) property specifies the maximum number of conversations that can share a channel instance. This value is the lowest value of the `SHARECNV` attribute on the associated client-connection channel and server-connection channel.

A value of 0 for `CURSHCNV` and `MAXSHCNV` indicates that sharing conversations is not enabled for this channel.

Sharing conversations and existing channel status fields

```
DISPLAY CHSTATUS (QM1.CLNT) ALL
AMQ8417: Display Channel Status details.
  CHANNEL (QM1.CLNT)          CHLTYPE (SVRCONN)
  BUFSRCVD (2)                BUFSSENT (2)
  BYTSRCVD (544)              BYTSSENT (536)
  CHSTADA (2017-05-22)        CHSTATI (06.31.42)
  COMPHDR (NONE,NONE)         COMPMMSG (NONE,NONE)
  COMPRATE (0,0)               COMPTIME (0,0)
  CONNNAME (127.0.0.)          CURRENT
  EXITTIME (0,0)               HBINIT (300)
  JOBNAME (0000A2800001498)    LOCLADDR ( )
  LSTMSGDA (2017-05-22)       LSTMSGTI (02.43.51)
  MCASTAT (RUNNING)           MCAUSER (MUSER_ADMIN)
  MONCHL (OFF)                MSGS (2)
  RAPPLTAG ( )                SSLCERTI ( )
  SSLKEYDA ( )                SSLKEYTI ( )
  SSLPEER ( )                 SSLKEYS (0)
  STATUS (RUNNING)             STOPREQ (NO)
  SUBSTATE (RECEIVE)           CURSHCNV (1)
  MAXSHCNV (3)                RVERSION (080000000)
  RPRODUCT (MQJB)
```

LSTMSGDA and LSTMSGTI are the date and time of the most recent MQI call that any thread that is sharing the channel instance made

If threads have a different user ID, then * is shown for MCA user ID

[Advanced IBM MQ client features](#)

© Copyright IBM Corporation 2020

Figure 5-15. Sharing conversations and existing channel status fields

Sharing conversations affects the display of server-connection (SVRCONN) type channels in a queue manager.

The **MCA user ID** (MCAUSER) is the effective user ID of the channel instance. If multiple program threads all have the same user ID, the value is shown. If any threads have a different user ID, an asterisk (*) is shown.

Last message date (LSTMSGDA) and **Last message time** (LSTMSGTI) indicate the date and time of the most recent MQI call that any thread that is sharing the channel instance makes.

Sharing conversations and channel exit considerations

- Consider channel exit behavior
- An exit sees the information flows for many independent client threads and might need to serialize access to the MQCD data type
- Review existing exits on client-connection channels and server-connection channels for their possible impact on their correct operation

Advanced IBM MQ client features

© Copyright IBM Corporation 2020

Figure 5-16. Sharing conversations and channel exit considerations

By using a pre-connect exit, MQI clients can be configured to retrieve client channel definitions from an external repository, such as LDAP and WebSphere Services Registry and Repository. The details of the exit library and function to call are specified in the `mqclient.ini` configuration file.

Sharing conversation performance

- Consider the possible impact on applications when multiple threads share a single client channel

Example:
An MQOPEN call in one thread might take longer than expected to run because many other threads are calling MQPUT and MQGET to process messages.
The tradeoff is that the queue manager is managing fewer client channels
- Set **SHARECNV** to 1 on the server connection channels to eliminate contention and improve performance
- Default **SHARECNV** value on a server-connection is 10.

Figure 5-17. Sharing conversation performance

The advantage of sharing conversations is that fewer active client-channel definitions are needed to support the environment and fewer resources are required, such as less memory and processor power.

You must consider the possible impact of sharing conversations on multithreaded applications. Some threads can be delayed when a multithreaded application issues an IBM MQ call because of the activity of other threads. If you know that threads in an application are each going to be making high use of a client channel, then do not share conversations. Instead, set the SHARECNV attribute so that each thread gets its own channel. If you know that each thread uses client channels relatively infrequently, then sharing conversations is appropriate.

5.2. Read ahead

Read ahead

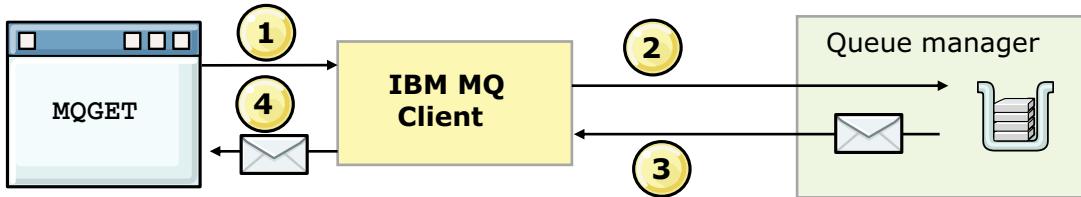
Advanced IBM MQ client features

© Copyright IBM Corporation 2020

Figure 5-18. Read ahead

Standard MQGET processing

1. Application sends an MQGET call
 2. IBM MQ client requests one message from the queue on the queue manager
 3. Queue manager sends one message to the IBM MQ client
 4. IBM MQ client passes the message to the application
5. Loop back to Step 1 and repeat until all of the messages are read



- Disadvantage:
 - Application client must wait for each MQGET operation to complete before it can process messages
 - Wait time can be long, depending on the network

[Advanced IBM MQ client features](#)

© Copyright IBM Corporation 2020

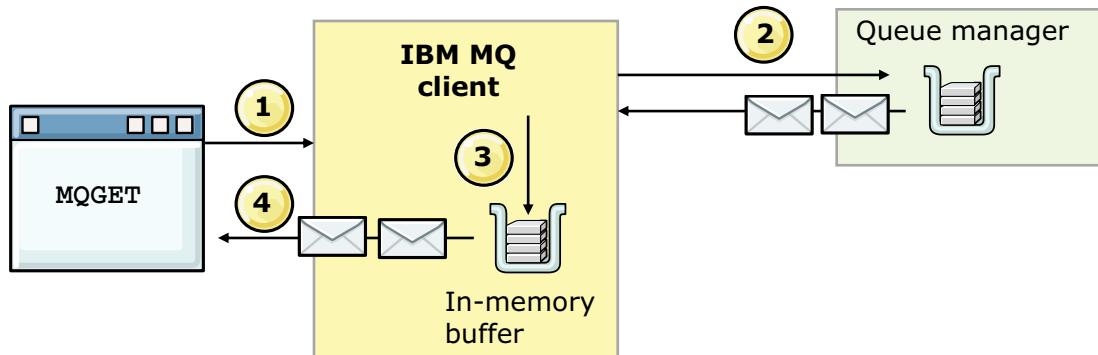
Figure 5-19. Standard MQGET processing

With standard MQGET processing, the IBM MQ client gets a message from the queue manager when the application sends an MQGET call and waits for a response from the queue manager before it sends another MQGET. The figure shows standard MQGET processing.

The disadvantage of standard MQGET processing is that the application client must wait for each MQGET operation to complete before it can process a message, which causes delays in message processing.

Read ahead overview

1. Application sends an MQGET call
2. Client reads messages from the queue on the queue manager
3. Client stores the messages in an in-memory buffer
4. Client passes the messages to the application from the in-memory buffer



Advanced IBM MQ client features

© Copyright IBM Corporation 2020

Figure 5-20. Read ahead overview

The figure shows the read ahead feature in IBM MQ.

When an application sends the first MQGET call, the IBM MQ client can independently get several messages from the queue manager and store them in an in-memory buffer. When the application issues subsequent MQGET calls, it does not have to wait for the IBM MQ client to get the message from the queue manager. Instead, because the messages are stored in a memory buffer, the IBM MQ client can immediately pass the next message to the application in response to an MQGET call.

The message buffer is an 'in memory' queue of messages. If the application ends or the server fails, these messages are lost.

Because this mechanism is designed to remove the network delay, it currently only benefits client applications.

The read ahead feature can also be used for browsing.

Read ahead advantages

- While the application is processing a message, the IBM MQ client is proactively:
 - Reading ahead
 - Getting messages from the queue on the queue manager
 - Storing the messages in an in-memory buffer
- When application sends an MQGET for the next message, the IBM MQ client can pass the message immediately from the in-memory buffer

[Advanced IBM MQ client features](#)

© Copyright IBM Corporation 2020

Figure 5-21. Read ahead advantages

Read ahead is useful for applications that get large numbers of non-persistent messages, outside of syncpoint where they are not changing the selection criteria regularly. For example, applications that get responses from a command server or a query, such as a list of airline flights.

A large proportion of the cost of an MQGET from a client is the turnaround time of the network connection. When IBM MQ uses read ahead (also referred to as *streaming*), the IBM MQ client makes a request for more than one message from the server. The server sends as many non-persistent messages that match the criteria as it can up to the limit set by the client. The largest speed benefit is seen where the application processes many similar non-persistent messages and where the network is slow.

If an application requests read ahead but the messages are not suitable, for example, they are all persistent, then only one message is sent to the client at any one time. Read ahead is effectively turned off until a sequence of non-persistent messages are on the queue again.

This performance improvement is available to both MQI and JMS applications.

Read ahead considerations

- Read ahead is supported for non-persistent messages only
- Messages that are streamed into memory on the client system are destructively removed from the queue manager
- If the client reads a persistent message, no further messages are read from the queue until the application reads the persistent message
- If the queue is closed, application MQCLOSE option determines what happens to any messages that are stored in the read ahead buffer
- Messages are lost if:
 - Client program ends
 - Client system fails before the application gets all the messages

[Advanced IBM MQ client features](#)

© Copyright IBM Corporation 2020

Figure 5-22. Read ahead considerations

Not all client application designs are suited to using read ahead, so read ahead is not enabled by default. For example, read is not supported for persistent messages. Do not use read ahead when the application processes persistent messages.

The administrator can enable read ahead at the queue or application level.

When an application calls MQOPEN with MQOO_READ_AHEAD, the IBM MQ client enables read ahead if certain conditions are met.

- Both the client and remote queue manager must be at WebSphere MQ Version 7 or higher.
- The client application must be compiled and linked against the threaded MQI client libraries.
- The client channel must be using TCP/IP protocol.

The two MQCLOSE options allow applications to configure what happens to any messages that are being stored in the read ahead buffer if the queue is closed. The application can specify to discard messages in the read ahead buffer or to get all of the messages before the queue is closed.

Default read ahead configuration (1 of 2)

1. Configure read ahead at the queue level
 - To enable read ahead, set the **Default read ahead (DEFREADA)** queue attribute to **YES**
 - To use read ahead only when a client requests it, set the **Default read ahead (DEFREADA)** queue attribute to **NO**
2. Configure read ahead at the application level option on the MQOPEN call
 - To use read ahead whenever possible, set to MQOO_READ_AHEAD
 - If the **Default read ahead (DEFREADA)** queue attribute determines read ahead behavior, set to MQOO_READ_AHEAD_AS_Q_DEF
3. Ensure that the channel has a nonzero value for the **Sharing conversations (SHARECNV)** setting on both the client and server channel definitions

[Advanced IBM MQ client features](#)

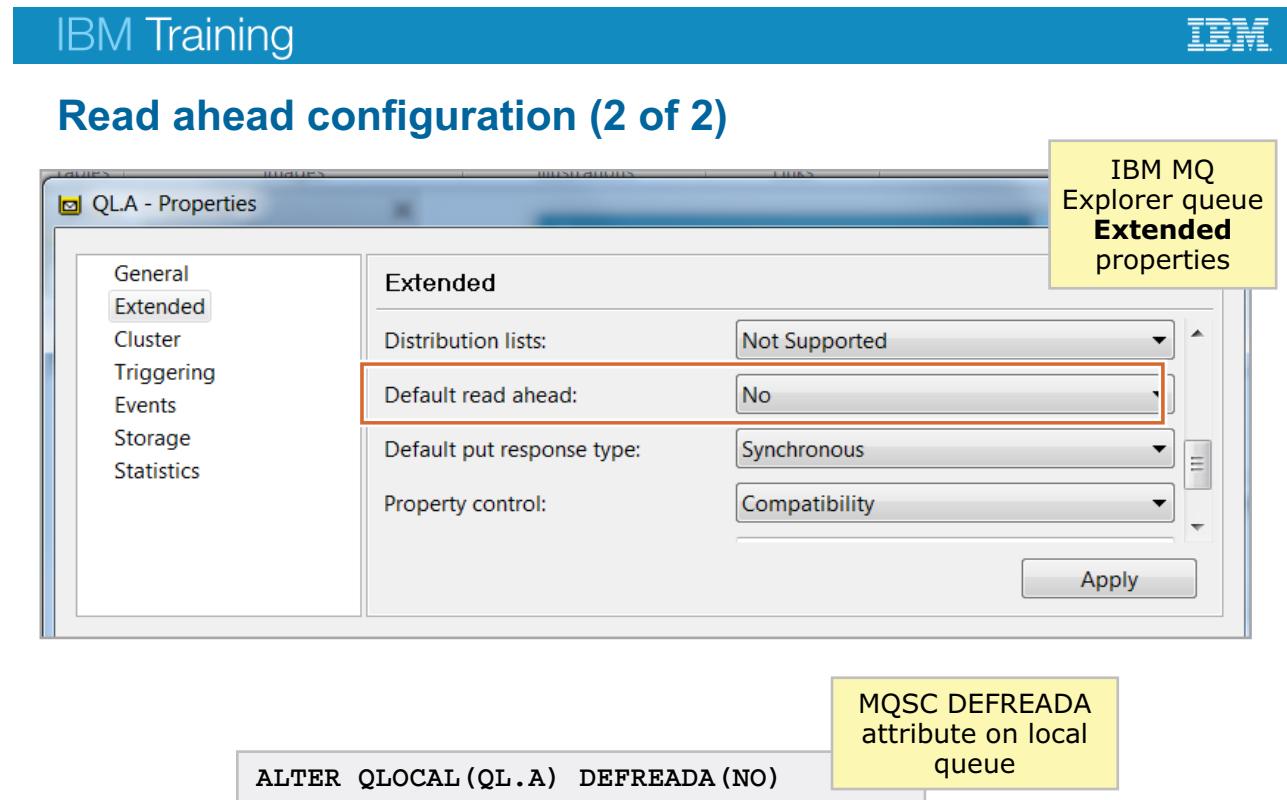
© Copyright IBM Corporation 2020

Figure 5-23. Default read ahead configuration (1 of 2)

Whether an application uses the read ahead feature is due to a combination of the settings that are used with the MQOPEN call from the application and the values that are set in the **Default read ahead (DEFREADA)** attribute on the queue.

The figure lists the steps for enabling read ahead.

1. To configure read ahead at the queue level, set the queue attribute **Default read ahead (DEFREADA)** to YES.
2. To use read ahead at the application level, implement one of the following options:
 - Use the MQOO_READ_AHEAD option on the MQOPEN function call to use read ahead wherever possible. It is not possible for the client application to use read ahead if the DEFREADA queue attribute is set to DISABLED.
 - Use the MQOO_READ_AHEAD_AS_Q_DEF option on the MQOPEN function call to use read ahead only when read ahead is enabled on a queue.
3. The channel must have a nonzero **Sharing conversations (SHARECNV)** values in both the client and server channel definitions.



Advanced IBM MQ client features

© Copyright IBM Corporation 2020

Figure 5-24. Read ahead configuration (2 of 2)

The **Default read ahead** (DEFREADA) parameter on a queue can be set by using the IBM MQ Explorer or the MQSC ALTER QLOCAL command. The valid values are NO, YES, and DISABLED.

- NO indicates that read ahead is not enabled. Setting DEFREADA to NO is equivalent to the behavior in IBM MQ versions before WebSphere MQ V7.0 in which messages are transported from the queue manager at the time they are requested.
- YES indicates that read ahead is enabled.
- DISABLED indicates that read ahead is disabled, even if an application that opens the queue specifies the MQOO_READ_AHEAD option.

Read ahead memory buffer size

- When the read ahead feature is in use:
 - Client receives messages ahead of an application that requests them, storing them in a buffer in memory on the client
 - Default behavior is that the client manages the size of the buffer
- Buffer size can be altered by adjusting the **MaximumSize** attribute of the **MessageBuffer** stanza on the client configuration file
 - Set to a value 1 - 999999
 - When **MaximumSize** = -1, the client determines the appropriate value
 - When **MaximumSize** = 0, read ahead is disabled for the client

Advanced IBM MQ client features

© Copyright IBM Corporation 2020

Figure 5-25. Read ahead memory buffer size

The read ahead memory buffer attributes are configurable.

Use the **MessageBuffer** stanza of the client configuration file to specify information about message buffers.

The **MaximumSize** attribute is an integer value that indicates the size of the read-ahead buffer, in the range of 1 KB through 999999 KB.

For more information about updating the client configuration file, see the IBM Knowledge Center for IBM MQ.

Read ahead buffer updates

- Default behavior is that the client keeps the memory buffer filled with messages so that the client can pass a message immediately to the application when it sends an **MQGET**
- **UpdatePercentage** and **MaximumSize** attributes in the client configuration determine when to get more messages

Example:

MaximumSize = 100 Kb and **UpdatePercentage** = 20

Threshold = $(100 - 20) = 80$ Kb

When **MQGET** calls remove 80 Kb from a queue, the client makes a new request automatically

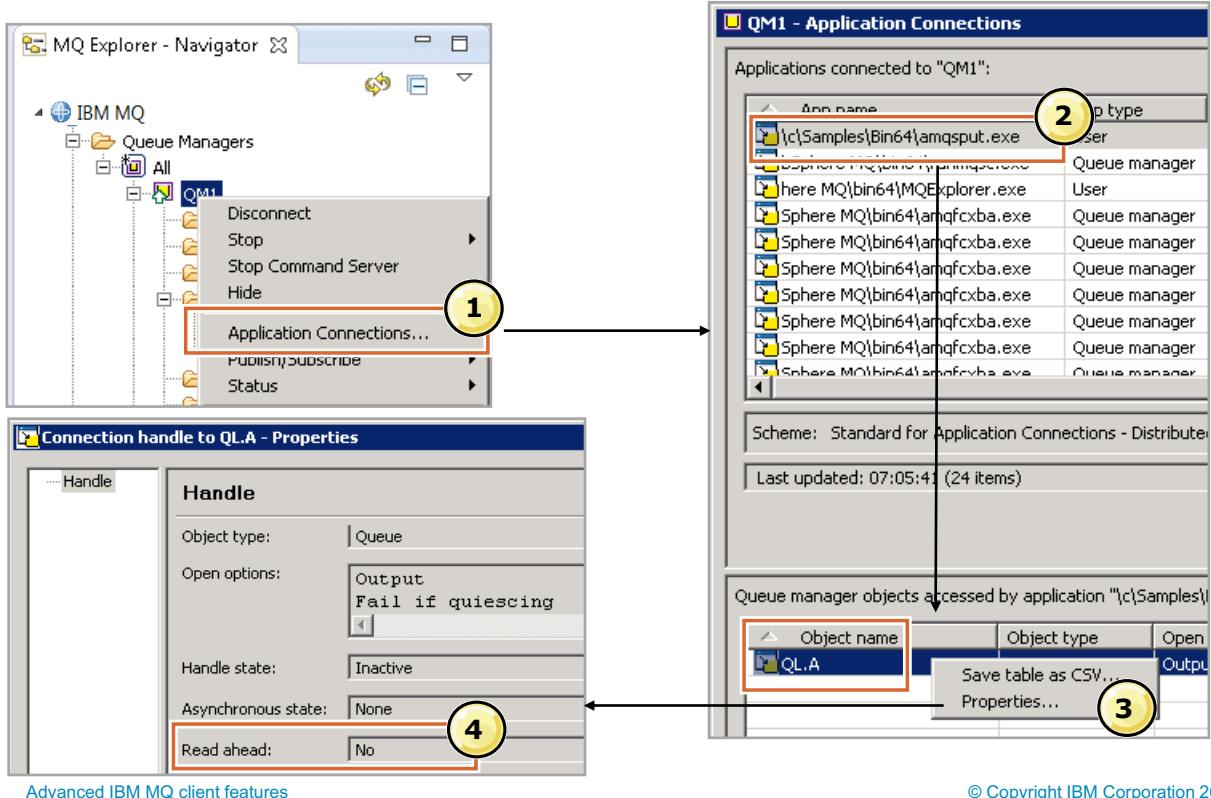
Figure 5-26. Read ahead buffer updates

The read ahead **UpdatePercentage** and **MaximumSize** attributes determine when client gets more messages.

The **UpdatePercentage** value, in the range of 1 - 100, is used in calculating the threshold value to determine when a client application makes a new request to the server. The special value of “-1” indicates that the client determines the appropriate value.



Using IBM MQ Explorer to get read ahead status



Advanced IBM MQ client features

© Copyright IBM Corporation 2020

Figure 5-27. Using IBM MQ Explorer to get read ahead status

The status of read ahead on connected applications can be viewed by using IBM MQ Explorer.

1. Right-click the queue manager in the **MQ Explorer - Navigator** view and then click **Application Connections**.
2. Click the application.
3. Right-click the application object name and then click **Properties**.
4. The **Read ahead** status field shows one of the following values:
 - **Yes** indicates that read ahead is enabled and is being used efficiently.
 - **No** indicates that read ahead is not enabled on any open queues.
 - **Inhibited** indicates that an application requested read ahead but it is inhibited because of incompatible options that are specified on the first call to MQGET.
 - **Backlog** indicates that read ahead is enabled but is not being used efficiently. **Backlog** can indicate that non-persistent messages were streamed into memory on the client system, but the client program is not requesting them. For example, the program might be using MQGET for specific correlation IDs.

Using MQSC to get read ahead status

```
DISPLAY CONN(*) TYPE(HANDLE) WHERE(READA EQ YES) ALL
CONN(301AEB4820008C02)
EXTCONN(414D5143514D43303120202020202020)
TYPE(CONN)
.
.
ASTATE(ACTIVE)
OBJNAME(QL.1)          OBJTYPE(QUEUE)
OPENOPTS(MQOO_INPUT_SHARED,MQOO_OUTPUT)
HSTATE(ACTIVE)          READA(YES)
```

Advanced IBM MQ client features

© Copyright IBM Corporation 2020

Figure 5-28. Using MQSC to get read ahead status

The figure shows an example of using the display connection (`DISPLAY CONN`) command in MQSC to show the read ahead status.

The **READA** field can show the following values:

- **YES** indicates that read ahead is enabled on an open queue and is being used efficiently.
- **NO** indicates that read ahead is not enabled on any open queues.
- **INHIBITED** indicates that an application requested read ahead but it is inhibited because of incompatible options that are specified on the first call to MQGET.
- **BACKLOG** indicates that read ahead is enabled but is not being used efficiently. BACKLOG can indicate that non-persistent messages were streamed into memory on the client system, but the client program is not requesting them. For example, the application might be using MQGET for specific correlation IDs.

5.3. Asynchronous put

Asynchronous put

Advanced IBM MQ client features

© Copyright IBM Corporation 2020

Figure 5-29. Asynchronous put

Asynchronous put introduction

- One of the strengths of IBM MQ is assured message delivery
- High qualities of service are not required for all messages in all applications
- With an asynchronous response from an **MQPUT** (or “fire and forget”) messages can be streamed down a network link without waiting for a return code
- Relevant in a client connection where a high quality of service is required
- In the right circumstances, significant performance improvement can be seen
- Enabled by a combination of API settings and queue and topic definitions

Advanced IBM MQ client features

© Copyright IBM Corporation 2020

Figure 5-30. Asynchronous put introduction

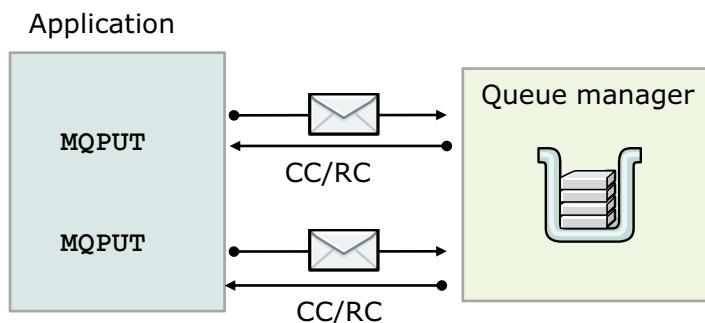
Asynchronous put (also known as *fire and forget*) recognizes that a large proportion of the cost of an MQPUT from a client is the turnaround of the network connection. When the application uses asynchronous put, it sends the message to the server but does not wait for a return code before it sends more MQI calls.

You can use asynchronous put to improve messaging performance in some situations. The largest performance improvement is seen in cases where the application sends many MQPUT calls and where the network is slow.

After the application completes the PUT sequence, it sends other calls, such as MQCMIT or MQDISC, which clears any MQPUT calls that are not complete.

MQPUT processing

- Normally, when an application puts a message on a queue, the application must wait for the queue manager to confirm that it processed the MQI request
- A completion code (CC) and reason code (RC) is returned after each message is written



Advanced IBM MQ client features

© Copyright IBM Corporation 2020

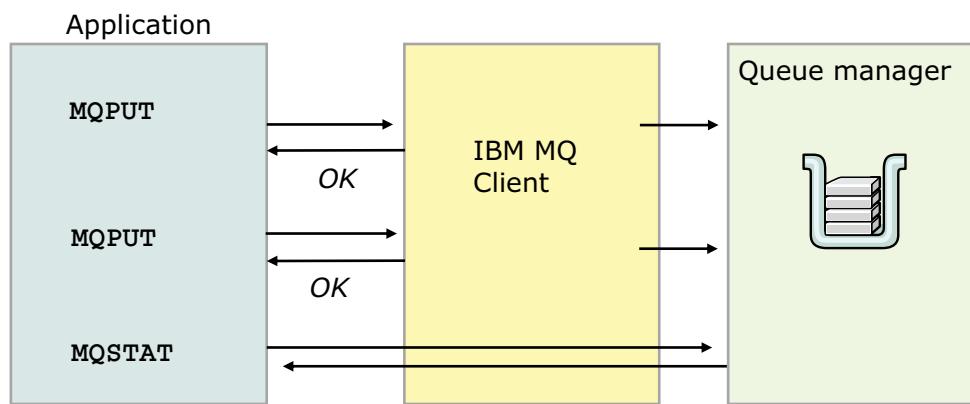
Figure 5-31. MQPUT processing

When a message is PUT by an application in bindings mode, the message is written directly to the queue in the queue manager.

Normally, when an application puts messages on a queue, by using MQPUT or MQPUT1, the application must wait for the queue manager to confirm that it processed the MQI request. The queue manager confirmation includes a completion code and a reason code.

MQPUT in client mode with asynchronous put

- Application puts message but the queue manager does not return the success or failure of each call
 - Client returns CC and RC with assumption that put was successful
 - An “OK” return from asynchronous put does mean that the message was delivered
- Application should use **MQSTAT** call to get the status information periodically



Advanced IBM MQ client features

© Copyright IBM Corporation 2020

Figure 5-32. MQPUT in client mode with asynchronous put

The figure shows an application with the asynchronous put function.

When the IBM MQ client receives the message from the application, it sends the message over the client channel. The client then returns an “OK” return code to the application without waiting for any reply from the queue manager.

When the application uses asynchronous put, the application should periodically send an MSTAT call to get the status information.

When asynchronous put applies

- Determined by combination of options in the MQPUT call and attributes of the queue
 - Application sets MQPMO_ASYNC_RESPONSE or MQPMO_RESPONSE_AS_Q_DEF in the MQPUT message options (MQPMO)
 - AND message is nonpersistent OR message is persistent and put within a unit of work
 - AND application client is connected
- If the conditions are not met, then a “normal” PUT occurs
- Queue attribute identifies default put response type

[Advanced IBM MQ client features](#)

© Copyright IBM Corporation 2020

Figure 5-33. When asynchronous put applies

To qualify for *fire and forget* put delivery, a message must meet a set of criteria. The process of determining whether it meets the criteria is carried out over two stages, which are information from the application.

In a nondistribution list scenario, an application put request is eligible for asynchronous delivery if one of the following conditions is true:

- The application MQPMO option field in the MQPUT call specifies MQPMO_ASYNC_RESPONSE.
- The application MQPMO options field in the MQPUT call specifies MQPMO_RESPONSE_AS_Q_DEF (or its synonym MQPMO_RESPONSE_AS_TOPIC_DEF) and the DEFRESP attribute was set to an asynchronous value at the time it was opened.

Finally, one or more of the following conditions must also be true:

- The message is a persistent message that is being put within a unit of work.
- The message MQMD persistence is set to MQPER_NOT_PERSISTENT.
- In the MQPUT call, the message MQMD **Persistence** attribute is set to MQPER_AS_Q_DEF (or its synonym MQPER_AS_TOPIC_DEF) and the object DEFPSIST attribute was set to non-persistent when the object was opened.

Sync point

- A successful commit for a unit of work means that all asynchronous puts were carried out successfully
- A subsequent **MQGET** (in the same unit of work) for a message that was PUT asynchronously, succeeds as normal

Advanced IBM MQ client features

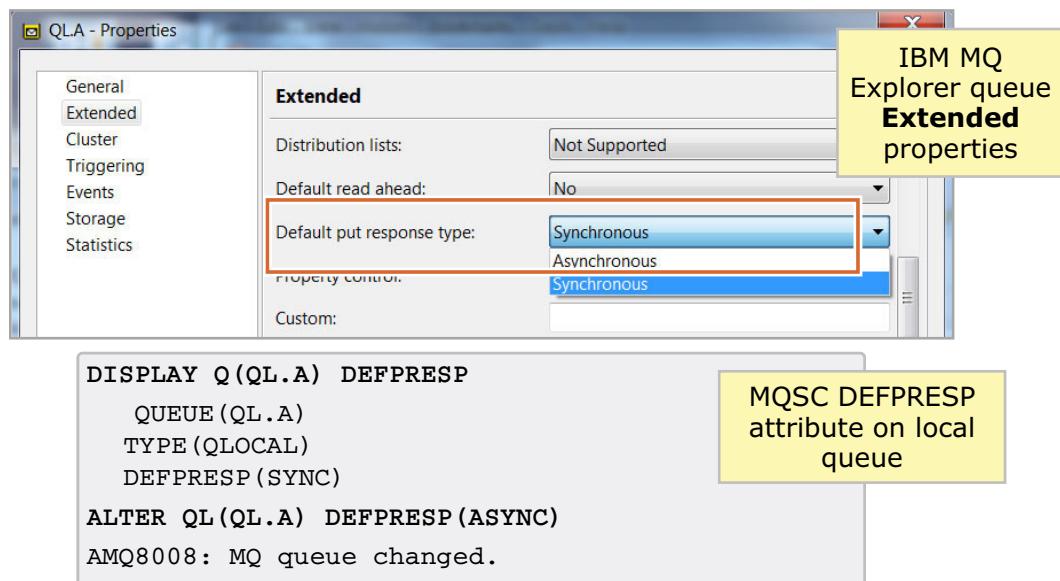
© Copyright IBM Corporation 2020

Figure 5-34. Sync point

If the application successfully commits a unit of work, then all messages put asynchronously were successfully written to the queue.

Setting the default put response type on a queue

- If the application uses the MQPMO_RESPONSE_AS_Q_DEF value in the MQPMO, then the **Default put response type (DEFPRESP)** attribute of the queue controls whether the application can use the asynchronous put capability



Advanced IBM MQ client features

© Copyright IBM Corporation 2020

Figure 5-35. Setting the default put response type on a queue

If the application uses the MQPMO_RESPONSE_AS_Q_DEF value in the put message options, then the **Default put response type (DEFPRESP)** property of the queue controls whether the application can use the asynchronous put capability.

The **Default put response type** property can be set by using IBM MQ Explorer or MQSC as shown in the figure.

- Synchronous** ensures that the PUT operations to the queue that specifies MQPMO_RESPONSE_AS_Q_DEF are sent as if MQPMO_SYNC_RESPONSE was specified instead. **Synchronous** is the default that is supplied with IBM MQ, but your installation might change it.
- Asynchronous** ensures that the PUT operations to the queue that specifies MQPMO_RESPONSE_AS_Q_DEF are sent as if MQPMO_ASYNC_RESPONSE was specified instead.

Unit summary

- Explain how to use the extended transactional client
- Manage client performance by sharing conversations
- Describe the performance impact of using read ahead
- Outline reasons for using asynchronous put

Advanced IBM MQ client features

© Copyright IBM Corporation 2020

Figure 5-36. Unit summary

Review questions



1. In an IBM MQ client application that uses the extended transactional client, in the case where one instruction fails, which component issues the backout command?
 - A. IBM MQ application
 - B. IBM MQ MQI Client
 - C. Transaction manager
 - D. MQI Channel
 - E. IBM MQ queue manager

2. True or False: A single MQI channel can support multiple threads.

Advanced IBM MQ client features

© Copyright IBM Corporation 2020

Figure 5-37. Review questions

Write your answers down here:

1.

2.

Review answers



1. In an IBM MQ client application that uses the extended transactional client, in the case where one instruction fails, which component issues the backout command?
 - A. [IBM MQ application](#)
 - B. IBM MQ MQI Client
 - C. Transaction manager
 - D. MQI Channel
 - E. IBM MQ queue manager

The answer is [A.](#)
2. [True](#) or False: A single MQI channel can support multiple threads.

The answer is [True.](#)

Unit 6. Working with queue manager clusters

Estimated time

01:30

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
- Lab exercise

Unit objectives

- Describe the components of a cluster
- Explain the purpose of full and partial repository queue managers
- Configure a basic cluster
- Outline cluster workload management features

Topics

- Introducing clusters
- Cluster workload management

Working with queue manager clusters

© Copyright IBM Corporation 2020

Figure 6-2. Topics

6.1. Introducing clusters

Introducing clusters

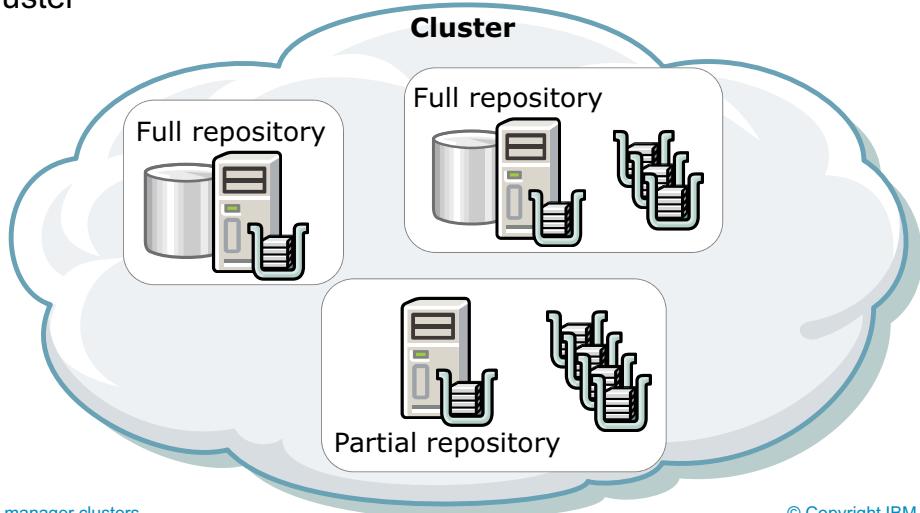
Working with queue manager clusters

© Copyright IBM Corporation 2020

Figure 6-3. Introducing clusters

Clusters

- Interconnected queue managers
 - Any queue manager can send a message to any other queue manager within the cluster
 - No explicit definitions required for channels, remote queues, or transmission queues
 - Queues on any queue manager are available to any other queue manager in the cluster



Working with queue manager clusters

© Copyright IBM Corporation 2020

Figure 6-4. Clusters

As introduced in Unit 3, "Configuring distributed queuing", clusters simplify how you connect your queue managers, and they make it easier to manage your cluster as you scale up.

You do not need to provide 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.

Benefits of clustering

- Reduced system administration
 - Fewer definitions (channels, remote queues, transmission queues for every destination in the cluster)
 - Set up or change network quickly and easily
 - Reduced risk of configuration errors
- Increased availability and workload balancing
 - Improved scalability
 - Message traffic is distributed across queues and queue managers
 - Increases resilience to system failure

Figure 6-5. Benefits of clustering

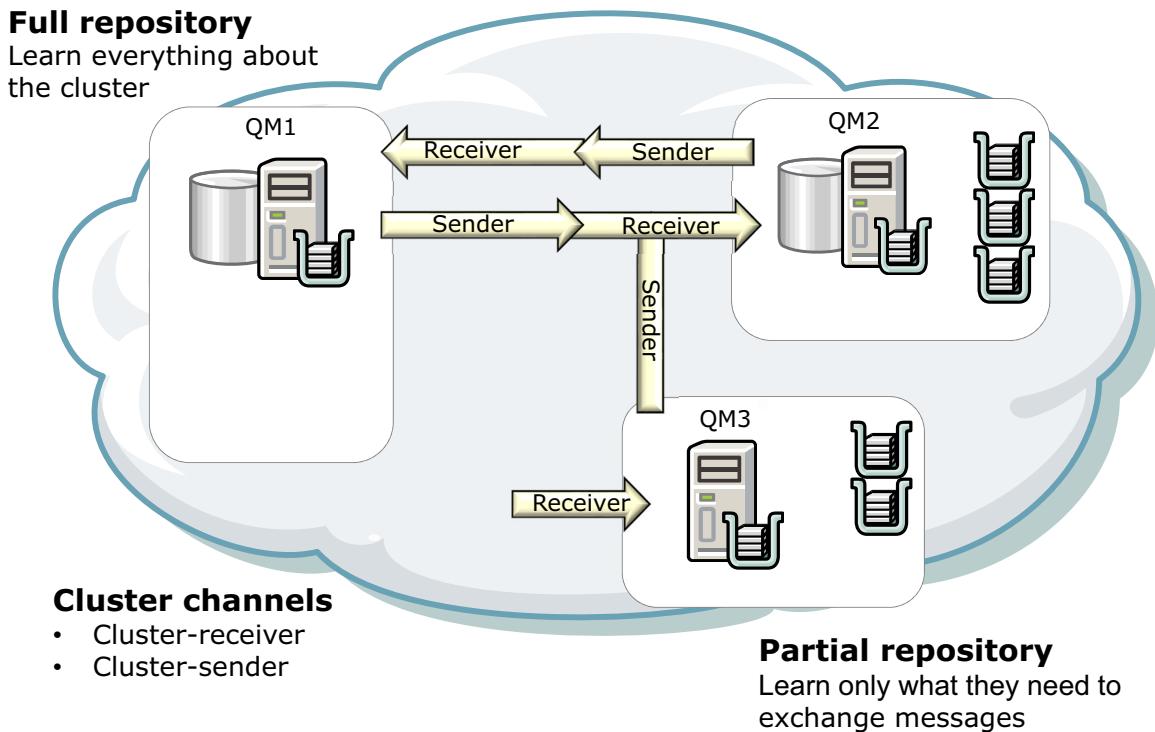
In addition to simplifying initial configuration, clusters also make it easier to manage your cluster as you scale horizontally. Whenever you create a receiver channel or define a queue, the corresponding sender channels and remote-queue definitions are created automatically on the other queue managers.

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.

You learn more about workload management in this unit and during the exercise. For more information, see Using clusters for workload management.

Overview of cluster components



Working with queue manager clusters

© Copyright IBM Corporation 2020

Figure 6-6. Overview of cluster components

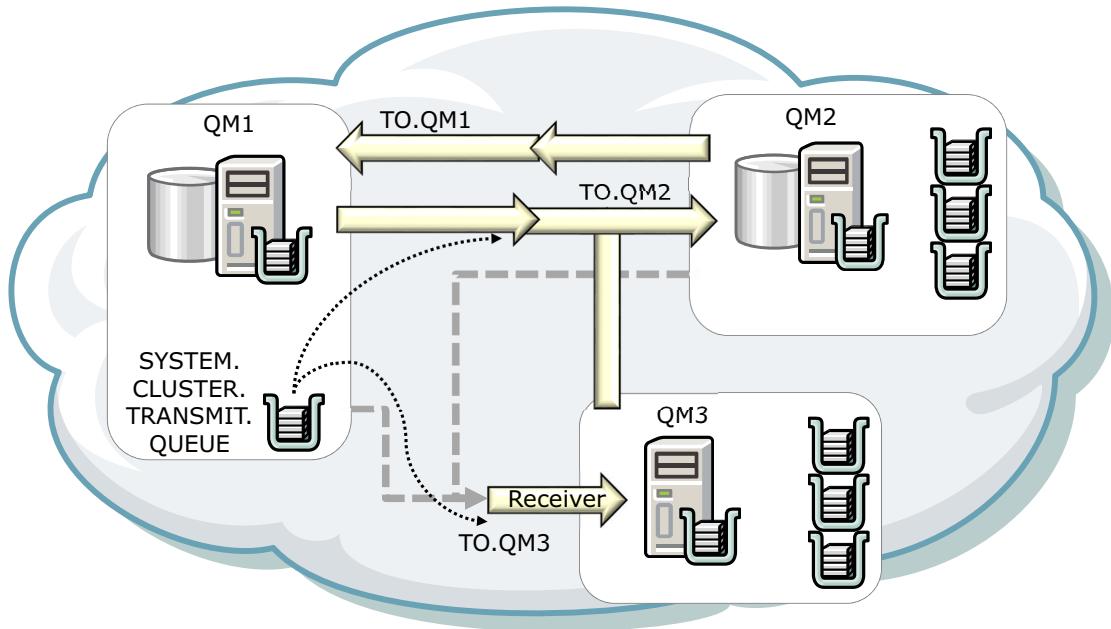
A cluster can contain two or more queue managers. In the diagram on the slide, the cluster has three queue managers. QM1 and QM2 host repositories of information about all the queue managers in the cluster. They are referred to as *full repository queue managers*. Full repository queue managers contain a complete set of information about every queue manager in the cluster.

QM3 has a partial repository. A partial repository queue manager holds records for local objects and remote objects that are used locally.

Each queue manager has a *cluster-receiver channel*. 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. To define the channel name, include the cluster name and the destination queue manager name.

Each queue manager also has a *cluster-sender channel* definition for the sending end of a channel on which it can send cluster information to one of the full repository queue managers. You 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. On this slide, QM1 and QM3 have cluster-sender channels that connect to QM2. QM2 has a cluster-sender channel that connects to QM1.

What makes a cluster work?



Working with queue manager clusters

© Copyright IBM Corporation 2020

Figure 6-7. What makes a cluster work?

Defining a cluster-sender channel introduces a queue manager to one of the full repository queue managers. The full repository queue manager updates the information in its full repository accordingly. Then, it automatically creates a cluster-sender channel back to the queue manager, and sends the queue manager information about the cluster. Thus a queue manager learns about a cluster and a cluster learns about a queue manager.

QM1 wants to send some messages to the queues at QM2. It knows which queues are available at QM2 because QM2 introduced itself to the queue by defining a cluster-sender channel to QM1. QM1 defined a cluster-sender channel to QM2. QM3 introduced itself to QM2. Because QM1 also holds a full repository, QM2 passed all the information about QM3 to QM1. Therefore, QM1 knows what queues are available at QM3 and what cluster-receiver channel QM3 defined. If QM1 wants to send some messages to queues at QM3, it automatically creates a cluster-sender channel connecting to the cluster-receiver channel at QM3.

All queue managers in the cluster have a cluster transmission queue, from which they can send messages to any other queue manager in the same cluster. QM1 uses the SYSTEM.CLUSTER.TRANSMIT.QUEUE to send its messages, as indicated by the dotted arrows.

The auto-definition of cluster-sender channels—defined automatically when needed—is crucial to the function and efficiency of clusters.

Considerations for a full repository

- Clusters should have two full repositories
 - Backup in case of failure
 - All other queue managers hold a partial repository
 - More than 2 is not recommended
- Hosting full repositories
 - Permanently connected platforms that do not have coinciding outages
 - Central position geographically
 - Consider dedicating systems as full repository host only
- Each full repository is connected to every other full repository in cluster
 - Manually define cluster-sender channels to full repositories and matching cluster-receiver channels
- Consider using the full repository in more than one cluster

[Working with queue manager clusters](#)

© Copyright IBM Corporation 2020

Figure 6-8. Considerations for a full repository

The cluster design should always have two queue managers with full repositories. Two full repositories are sufficient. If a failure of a full repository occurs, the cluster can still operate. Full repositories must be held on servers that are reliable and as highly available as possible and single points of failure must be avoided.

For administrative convenience, you can use the same queue manager to host the full repositories for several clusters. This benefit should be balanced against the expected workload for that queue manager.

You manually connect 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. When 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.

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. If a queue manager is temporarily out of service at the expiry date, this grace period allows it 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.

Steps to set up a basic cluster

1. Determine which queue managers to add to the cluster
 - Organization of the cluster
 - Naming conventions
2. Decide which queue managers are to hold full repositories
3. Alter the full repository queue manager definitions to add the repository information
4. Define the listeners
5. Define the cluster-receiver channels on each queue manager
6. Define the cluster-sender channels on each full repository queue manager
7. Define the cluster queues
8. Verify and test the cluster

Working with queue manager clusters

© Copyright IBM Corporation 2020

Figure 6-9. Steps to set up a basic cluster

The slide lists the basic steps for setting up a cluster. During the exercise, you use MQ Explorer to define a cluster.

Define cluster-receiver channels (CLUSRCVR)

- 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(CLUSSDR) TRPTYPE(TCP) +
CONNNAME(localhost(5002)) CLUSTER(CLUS1) +
DESCR('Cluster-receiver channel for QM1 on cluster CLUS1')
```

Figure 6-10. Define cluster-receiver channels (CLUSRCVR)

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 this object definition, a queue manager is advertised to the other queue managers in the cluster. Then, they 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 hostname, or an IP address. If the port number is not specified, the default TCP listener port for IBM MQ (1414) is used.

Define cluster-sender channel (CLUSSDR)

- 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')
```

Figure 6-11. Define cluster-sender channel (CLUSSDR)

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, such as 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.

Define local queues to the cluster

- Cluster queues are 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 6-12. Define local queues to the cluster

A cluster queue is a queue that is hosted by a cluster queue manager and made available to other queue managers in the cluster.

A cluster queue definition is advertised to other queue managers in the cluster. The other queue managers in the cluster can put messages to a cluster queue without needing a corresponding remote-queue definition.

When a queue is advertised, any queue manager in the cluster can put messages to it. To put a message, the queue manager must find out, from the full repositories, where the queue is hosted. Then, it adds some routing information to the message and puts the message on a cluster transmission queue.

=====

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 pre-existing 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.

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.

A full repository is in most ways exactly like any other queue manager, and it is therefore possible to host application queues on the full repository and connect applications directly to these queue managers. Should applications use queues on full repositories?

Although this configuration is possible, many customers prefer to keep these queue managers dedicated to maintaining the full repository cluster cache.

https://www.ibm.com/support/knowledgecenter/SSFKSJ_9.1.0/com.ibm.mq.pla.doc/q004760_.htm

System cluster queues

SYSTEM.CLUSTER.COMMAND.QUEUE

- Source of all work for the cluster repository process
- Messages arrive here from local configuration commands and from FR to PR and PR to FR
- If you delete messages from here, you might need a cluster refresh
- Steady state: empty

SYSTEM.CLUSTER.REPOSITORY.QUEUE

- Persistent store for accumulated cluster knowledge
- All cluster object changes persisted here
- Based on local configuration and remote knowledge
- If you delete messages from here, your only option is a cluster refresh
- Steady state: non-empty

SYSTEM.CLUSTER.TRANSMIT.*

- Used to transfer all messages within a cluster
- Shared between user messages and cluster control messages
- If you delete messages from here, you might need a cluster refresh, and you might lose your own messages
- Steady state: Ideally empty (might contain messages while channels are down)

SYSTEM.CLUSTER.HISTORY.QUEUE

- Intended for capturing diagnostics for IBM MQ Service teams
- Only used when REFRESH CLUSTER is issued
- Entire contents of local cluster cache written before processing REFRESH
- Messages expire after 180 days
- Steady state: ignore it

Figure 6-13. System cluster queues

This slide describes the special SYSTEM queues for clustering. These queues are created automatically when a queue manager is created.

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**

[Working with queue manager clusters](#)

© Copyright IBM Corporation 2020

Figure 6-14. 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. IBM 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.

Troubleshooting clusters

- A break in communication between queue managers
 - Check the partial-repository-to-full-repository and full-repository-to-partial-repository channels
- Configuration issues
 - Check that the host of the resource and that the queue manager correctly joined the cluster
- What to check
 - Cluster names
 - Are the queue managers known to the full repositories
 - Queue manager channels between full and partial repositories
 - Queue manager error logs and messages about expiring objects
 - System queues
- Sequence of diagnostic steps
 - Check the queue manager where the cluster definitions live
 - Check all the full repositories
 - Check the queue manager where you're sending the messages from

[Working with queue manager clusters](#)

© Copyright IBM Corporation 2020

Figure 6-15. Troubleshooting clusters

If cluster queues are not visible where you think they should be, or if messages sit on a transmission queue for no reason, check for the following issues:

- A communication break between queue managers: Check the partial-repository-to-full-repository and full-repository-to-partial-repository channels
- A configuration issue: Check the host of the resource and that the queue manager correctly joined the cluster.

You can also check:

- Cluster names
- Queue manager cluster knowledge and whether the queue managers known to the full repositories
- Queue manager channels between full and partial repositories
- Queue manager error logs and whether they contain messages about expiring objects
- System queues

Start by checking the queue manager where the cluster definitions live. Then, check all the full repositories. Finally, check the queue manager where you're sending the messages from.

Cluster troubleshooting

- Check the channels
 - All cluster channels are paired
 - Channels are running: `DISPLAY CHSTATUS (*)`
 - Every queue manager in the cluster has a manually defined cluster-receiver channel
 - Every full repository has a cluster-sender channel to every other full repository and the channel is running
- Check the queue managers
 - All queue managers in the cluster are aware of all the full repositories: `DISPLAY CLUSQMGR(*) QMTYPE`
 - Intended full repository queue managers are defined as full repositories and are in the correct cluster: `DISPLAY QMGR REPOS REPOSNL`
- Check the queues
 - Messages are not building up on transmission queues
 - Messages are not building up on system queues

Figure 6-16. Cluster troubleshooting

This slide summarizes cluster troubleshooting tips that help you to detect and resolve problems when you use queue manager clusters.

6.2. Cluster workload management

Cluster workload management

Working with queue manager clusters

© Copyright IBM Corporation 2020

Figure 6-17. Cluster workload management

Cluster workload management options

- Organize your cluster with queue managers as clones of each other
 - Applications do not need to explicitly name the queue manager when sending messages
 - Workload management algorithm determines target queue manager
- Advantages
 - Increased availability of your queues and applications
 - Faster throughput of messages
 - More even distribution of workload in your network

Figure 6-18. 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. Any one of the queue managers that hosts an instance of a particular queue can handle messages that are destined for that queue. So applications do not need to explicitly name the queue manager when sending messages. A workload management algorithm determines which queue manager should handle the message.

This technique can increase the capacity available to process messages. It can also enable failover 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.

Channel workload balancing

- Applies when multiple cluster queues with the same name
- Applied in one of 3 ways:
 - **Bind on open:** When the putting application opens the queue
 - **Bind on group:** When a message group is started
 - **Bind not fixed:** When a message is put to the queue
- Results
 - The source queue manager builds a list of all potential targets based on the queue name
 - Eliminates impossible options
 - Prioritizes remaining options
 - If multiple options are equal, workload balancing applies
- Balancing is based on:
 - The channel, not the target queue
 - Channel traffic to all queues
 - Weightings that might be applied to the channel

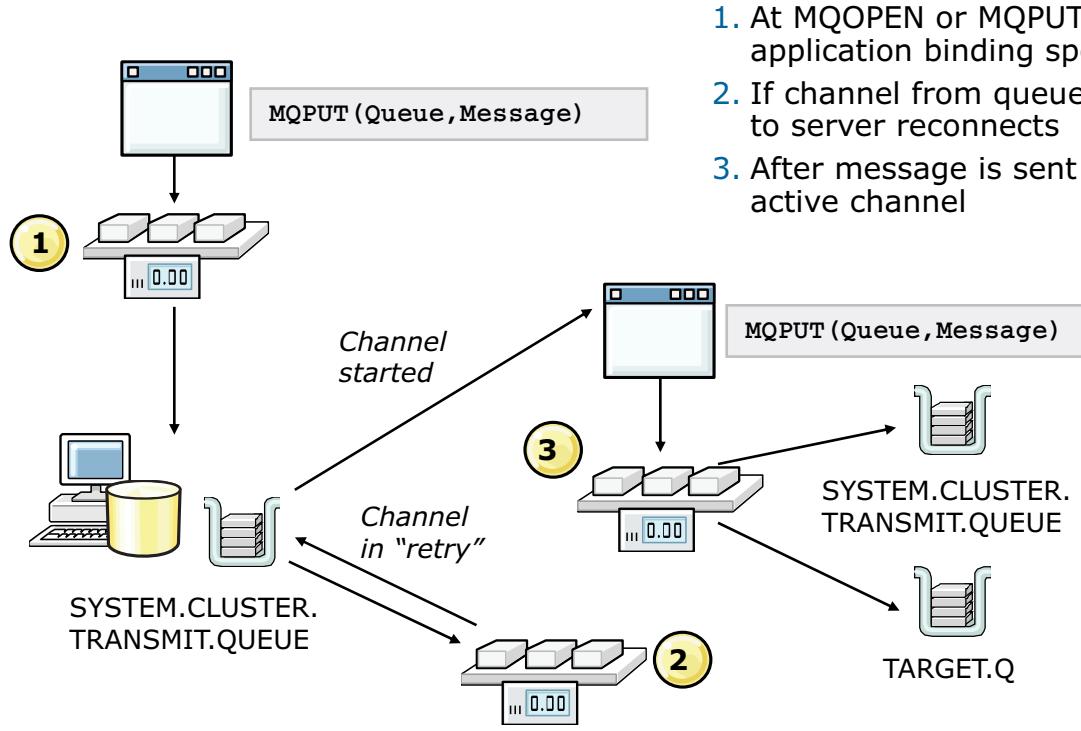
Working with queue manager clusters

© Copyright IBM Corporation 2020

Figure 6-19. Channel workload balancing

With multiple choices, which are based on availability and channel priorities, a built-in workload management algorithm determines the remote queue manager.

When workload balancing can occur



Working with queue manager clusters

© Copyright IBM Corporation 2020

Figure 6-20. When workload balancing can occur

In a typical queue manager cluster, a message is put to a queue manager in the cluster. In some cases, the message is destined for a back-remote queue manager that is also in the cluster. In this configuration, workload balancing can occur in three points.

1. Workload balancing can occur at either MQOPEN or MQPUT, depending on the bind options that are associated with the message affinity.
 - If “bind on open” is specified, then all messages go to the same destination. After the destination is chosen at MQOPEN time, no more workload balancing occurs on the message.
 - If “bind not fixed” is specified, the messages can be sent to any of the available destinations. The decision where to send the message is made at MQPUT time. However, the destination can change while the message is in transit.

In either case, the message is put on the local cluster queue instance if one is available on the application’s queue manager. If a local cluster queue instance is not available, the message is put on the SYSTEM.CLUSTER.TRANSMIT.QUEUE.

2. Workload balancing can occur when a channel from a queue manager to a back-end server is attempting to connect again. In this case, any “bind not fixed” messages that are waiting to be sent go through workload balancing again to see whether a different destination is available.

3. After a message is transported over an active, started channel, the channel calls MQPUT to put the message to the target queue and the workload algorithm is called again.

Cluster workload management algorithm: Summary

1. Queue manager builds a list of possible destinations
2. If queue is specified, eliminate the following queues:
 - Queues with PUT(DISABLED)
 - Remote instances of queues that do not share a cluster with the local queue manager
 - Remote CLUSRCVR channels that are not in the same cluster as the queue
3. If queue manager name is specified, eliminate the following queues:
 - Remote instances of queues that do not share a cluster with the local queue manager
 - Remote CLUSRCVR channels that are not in the same cluster as the queue or topic
4. If available, use the local queue instance unless overridden by use-queue
5. Evaluate channel rank
6. Evaluate channel status
7. Evaluate cluster-receiver channel priority
8. Evaluate channel priority
9. Evaluate queue priority
10. Evaluate most recently used channels
11. Evaluate least recently used with channel weight

Working with queue manager clusters

© Copyright IBM Corporation 2020

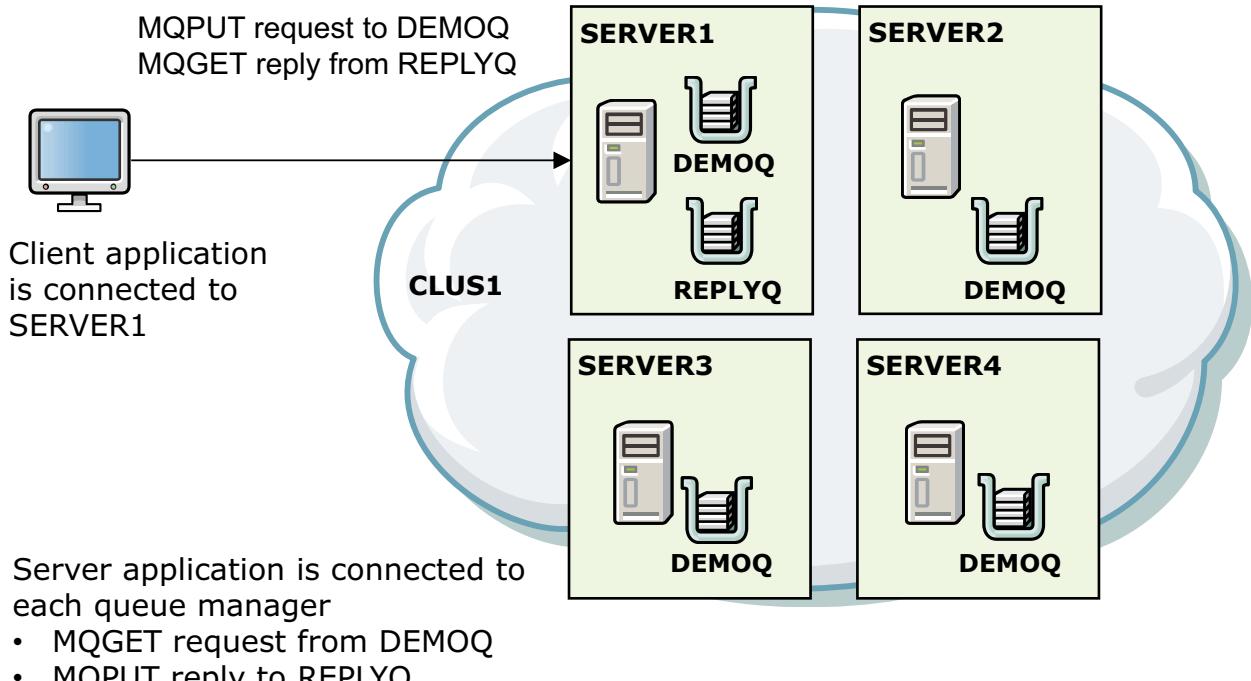
Figure 6-21. Cluster workload management algorithm: Summary

Every time a message is put on a queue, the cluster workload management algorithm runs once.

This figure summarizes the workload management algorithm.

The distribution of user messages is not always exact. The administration and maintenance of the cluster causes messages to flow down channels, which can result in an apparent uneven distribution of user messages. For this reason, do not rely on the exact distribution of messages during workload balancing.

Queue manager cluster scenario



[Working with queue manager clusters](#)

© Copyright IBM Corporation 2020

Figure 6-22. Queue manager cluster scenario

In this scenario, the cluster contains 4 queue managers on different servers. The client application is connected to SERVER1. It puts request messages to DEMOQ, which is a cluster queue on all four queue managers.

When a message arrives on a DEMOQ, a server application reads it and then puts a reply message to the reply-to queue REPLYQ on SERVER1.

Use-queue basics

- Allows remote queues to be chosen when a local queue exists
- On local queue: Applies when an application or a channel that is not a cluster channel puts the message

```
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

- On queue manager when **CLWLUSEQ** queue attribute is set to **QMGR**: Specifies whether a local instance of a queue is given preference as a destination over other instances of the queue in a cluster

```
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

Figure 6-23. Use-queue basics

The use-queue attribute, CLWLUSEQ, specifies the behavior of an MQPUT operation when the cluster contains a local instance and at least one remote instance of a cluster queue. 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.

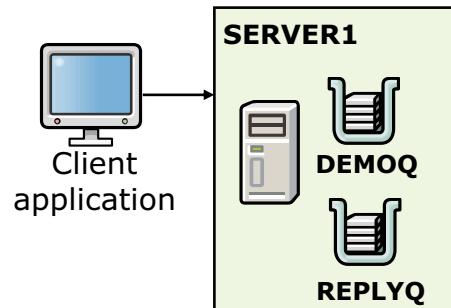
- If you specify QMGR for the attribute value on the queue, the CLWLUSEQ parameter of the queue manager definition determines the behavior. This parameter is valid only for local queues.
- If you specify ANY, the queue manager treats the local queue as another instance of the cluster queue for the purposes 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.

Use-queue example

- Start with the use-queue defaults on SERVER1
 - Queue manager CLWLUSEQ (LOCAL)
 - Queue CLWLUSEQ (QMGR)
- Consider the following definition changes:

```
ALTER QMGR CLWLUSEQ (ANY)
ALTER QL (DEMOQ) CLWLUSEQ (LOCAL)
ALTER QL (DEMOQ) CLWLUSEQ (ANY)
```



With default values, all messages from the client application are delivered to SERVER1

Figure 6-24. Use-queue example

The examples for each workload feature that is described in the rest of this unit starts with all cluster workload attributes reset to the initial values, with one exception.

By default, the queue CLWLUSEQ value is set to QMGR and the queue manager CLWLUSEQ value is set to LOCAL so that all request messages are delivered to the local DEMOQ on SERVER1.

- Changing the queue manager value to ANY means that request messages are delivered to any of the four queue managers because the queue value is set to QMGR.
- Changing the queue value to LOCAL means that the queue manager value is ignored, so all request messages are delivered to SERVER1.
- Changing the queue value to ANY means that the queue manager value is ignored, and all request messages are delivered to any of the four queue managers.

Rank basics

- Channels and queues with the highest rank are chosen preferentially over channels and queues with lower ranks
 - Range is 0 – 9
 - Default is 0
- Channel rank is checked before queue rank

Examples:

```
DEFINE CHL(CLUS1.SERVER1) CHLTYPE(CLUSRCVR) CLUSTER(CLUS1) ...
CLWLRank( )

DEFINE QL(DEMOQ) CLUSTER(CLUS1) CLWLRank( )
```

Figure 6-25. Rank basics

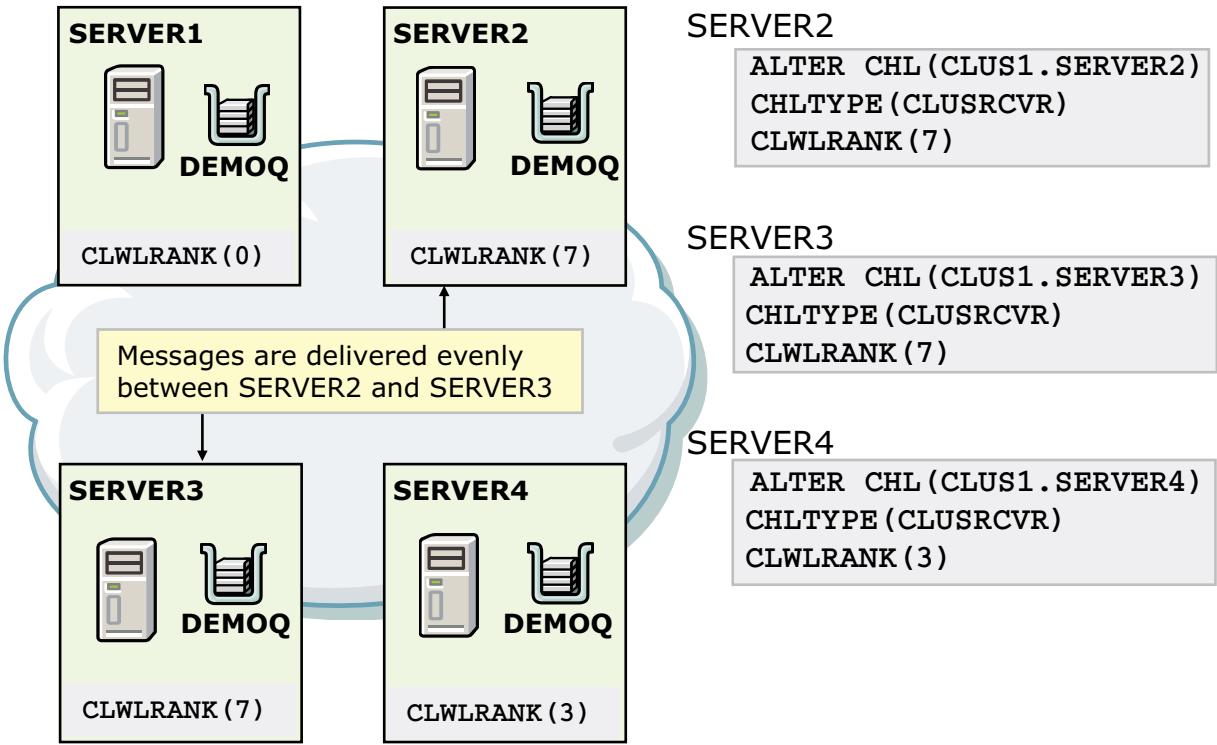
This cluster workload rank attribute (CLWLRank) specifies the rank of the queue for the purposes of cluster workload distribution. Rank can be specified on channels and queues.

The cluster workload rank parameter is valid only for local, remote, and alias queues. The value must be in the range of 0-9, where 0 is the lowest rank and 9 is the highest.

Use the channel cluster workload rank attribute on cluster-receiver channel definitions only.



Channel rank example



Working with queue manager clusters

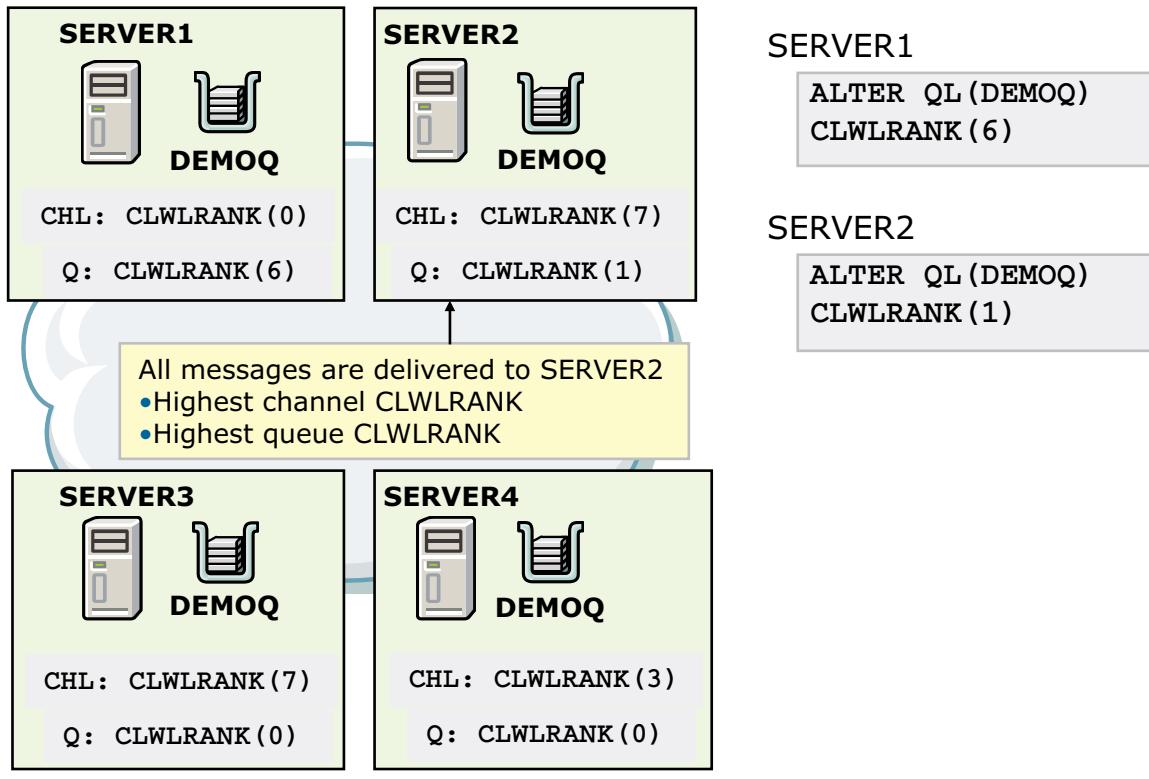
© Copyright IBM Corporation 2020

Figure 6-26. Channel rank example

In this example, the cluster workload channel ranks for SERVER2 and SERVER3 are set higher than SERVER4. SERVER1 has the lowest rank because the default channel rank is zero. After the ranks for channels on SERVER2, SERVER3, and SERVER4 are altered, the messages are distributed equally between the most highly ranked destinations (SERVER2 and SERVER3).

This example assumes that the SERVER1 queue manager cluster workload use-queue (CLWLUSEQ) value is set to "Any". Messages are delivered to all four queue managers, despite the existence of a local instance of DEMOQ on SERVER1.

Channel and queue rank example



Working with queue manager clusters

© Copyright IBM Corporation 2020

Figure 6-27. Channel and queue rank example

After the ranks for queues on SERVER1 and SERVER2 are changed in this example, all the messages are delivered to SERVER2 because the cluster workload management algorithm checks channel ranks before it checks queue rank.

The channel rank check leaves SERVER2 and SERVER3 as valid destinations. Because the queue rank for DEMOQ on SERVER2 is higher than the rank on SERVER3, the messages are delivered to SERVER2. Channel rank is more powerful than queue rank, so the most highly ranked queue (on SERVER1) is not chosen.



Note

In the figure, the cluster workload queue rank is shown below the cluster workload channel rank. For example, on SERVER1 CLWLRANK for the channel is 0, and for the queue it is 6.

The destinations with the highest rank are chosen regardless of the channel status to that destination. Ranking can lead to messages that build up on the SYSTEM.CLUSTER.TRANSMIT.QUEUE.

Channel and queue priority basics

- Channels and queues with the highest priorities are chosen preferentially over channels and queues with lower priorities
 - Range is 0 – 9
 - Default is 0
- Channel priority is checked before queue priority
- Rank is checked before channel status, and priority is checked afterward

Examples:

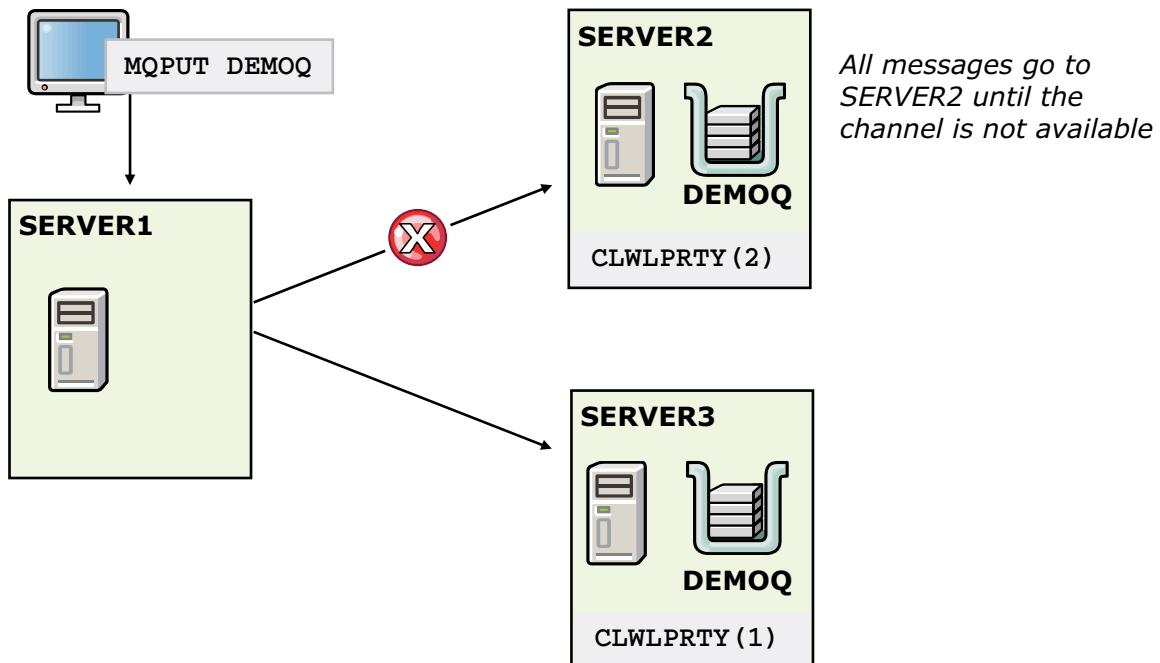
```
DEFINE CHL(CLUS1.SERVER1) CHLTYPE(CLUSRCVR) CLUSTER(CLUS1) ...
      CLWLPRTY( )

DEFINE QL(DEMOQ) CLUSTER(CLUS1) CLWLPRTY( )
```

Figure 6-28. Channel and queue priority basics

The cluster workload priority attribute (CLWLPRTY) can be specified on queues and channels. The channels and queues with the highest priorities are chosen over channels and queues with lower priorities.

Using CLWLPRTY to identify primary and backup servers



Working with queue manager clusters

© Copyright IBM Corporation 2020

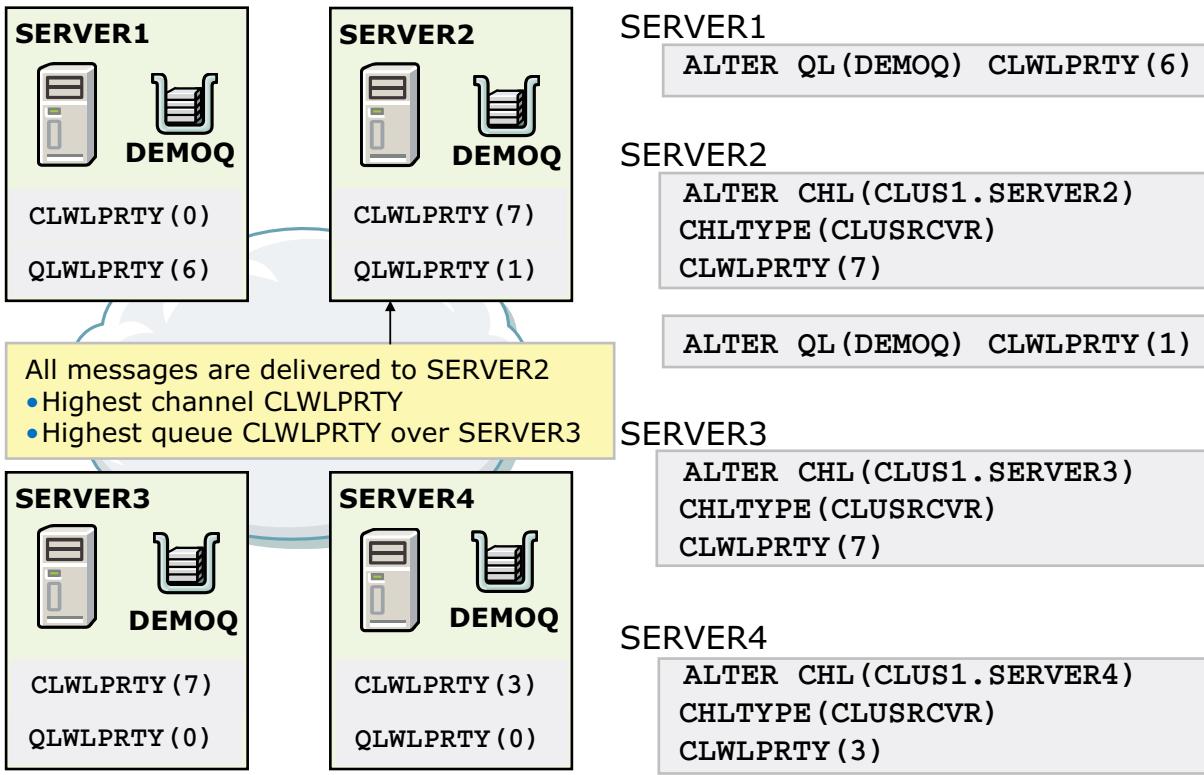
Figure 6-29. Using CLWLPRTY to identify primary and backup servers

Cluster workload priority can be used to identify primary and backup servers. Messages are delivered to primary (high priority) servers until they become unavailable, at which point messages are then workload-balanced to the backup (low priority) servers.

While both channels from SERVER1 are running, all messages put to DEMOQ by the application connected to SERVER1 are delivered to SERVER2 because SERVER2 has a higher channel priority.

When the channel from SERVER1 to SERVER2 stops (as shown with the red "X"), messages that are then put to DEMOQ by the application that is connected to SERVER1 are delivered to SERVER3.

Priority example (1 of 2)



Working with queue manager clusters

© Copyright IBM Corporation 2020

Figure 6-30. Priority example (1 of 2)

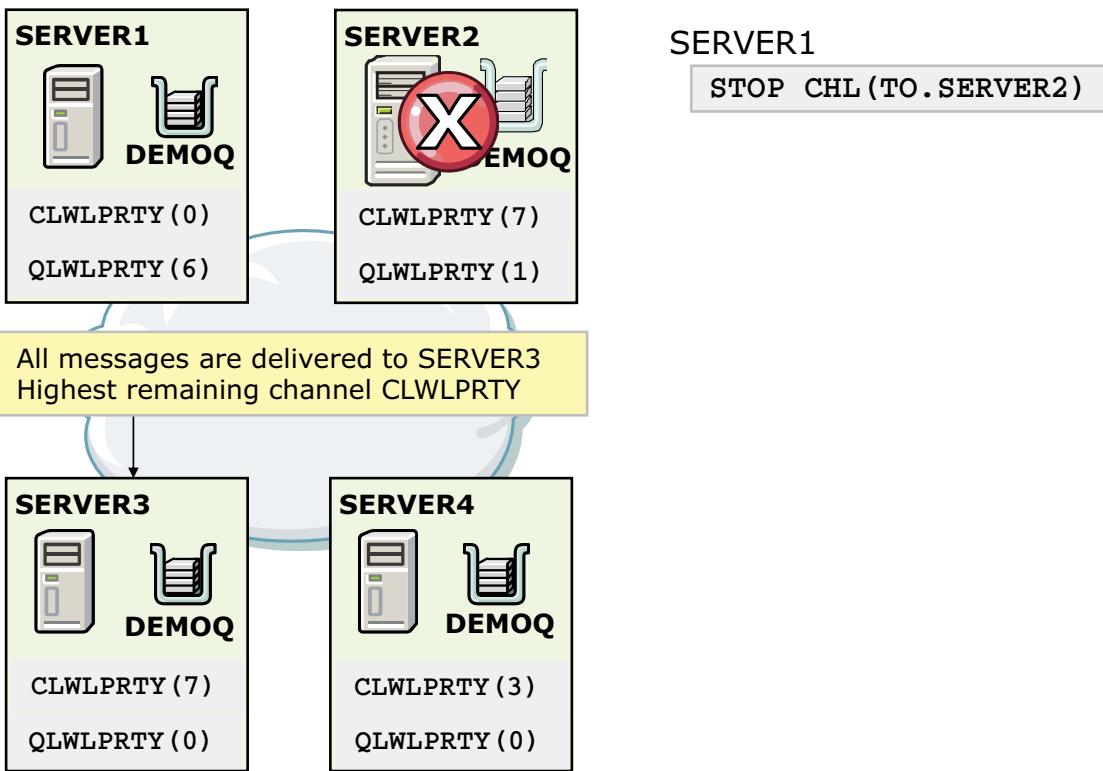
The channel priority check leaves SERVER2 and SERVER3 as valid destinations. Because the queue priority for DEMOQ on SERVER2 is higher than the queue priority on SERVER3, the messages are delivered to SERVER2. This scenario assumes that the channel status to all destinations is equally preferential (in this case, either running or inactive).



Note

Because channel priority is more powerful than queue priority, the queue with the highest priority (on SERVER1) is not chosen.

Priority example (2 of 2)



Working with queue manager clusters

© Copyright IBM Corporation 2020

Figure 6-31. Priority example (2 of 2)

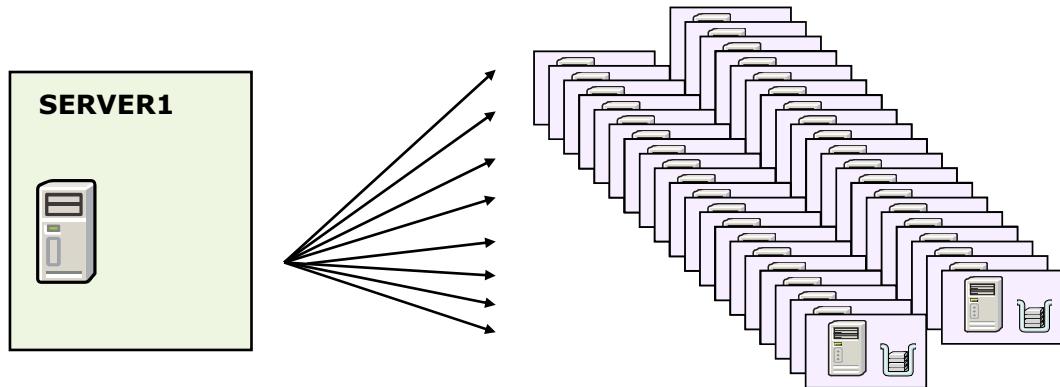
Workload priority is checked after channel status in the cluster workload management algorithm. Stopping the channel from SERVER1 to SERVER2 leaves the following channel status for each destination:

- SERVER1: Inactive
- SERVER2: Stopped
- SERVER3: Running
- SERVER4: Running

The cluster workload management algorithm removes SERVER2 from the list of valid destinations, so the priority check is completed only on SERVER1, SERVER3, and SERVER4. Out of the remaining three queue managers, the one with the highest priority is SERVER3, so all messages are delivered to SERVER3.

Most recently used channels

- **Problem:** How can you limit the number of active outbound channels in large clusters?
 - Hundreds of queue managers in the cluster means hundreds of outbound channels exist
 - Active channel limits can limit cluster size



- **Solution:** Use the cluster workload most recently used channels feature

Working with queue manager clusters

© Copyright IBM Corporation 2020

Figure 6-32. Most recently used channels

The most recently used channels attribute can be used to limit the number of active outbound channels in large clusters. The cluster workload attribute is CLWLMLRUC.

Most recently used channels basics

- If more than one destination is valid, limit the round robin to the “n” most recently used channel destinations
- MQSC: **ALTER QMGR CLWLMRUC()**
 - Range: 1 – 999999999
 - Default: 999999999
- Example:
From the default CLWLMRUC(999999999), reduce the MRU value to restrict the number of destinations in the round robin
ALTER QMGR CLWLMRUC(2)

[Working with queue manager clusters](#)

© Copyright IBM Corporation 2020

Figure 6-33. Most recently used channels basics

When the most recently used (MRU) value is lower than the number of valid destinations available to the round robin, all but the MRU channels are removed from the round robin.

If the CLWLMRUC attribute is set to 2 on a queue manager, messages are delivered to only two of the four destinations.

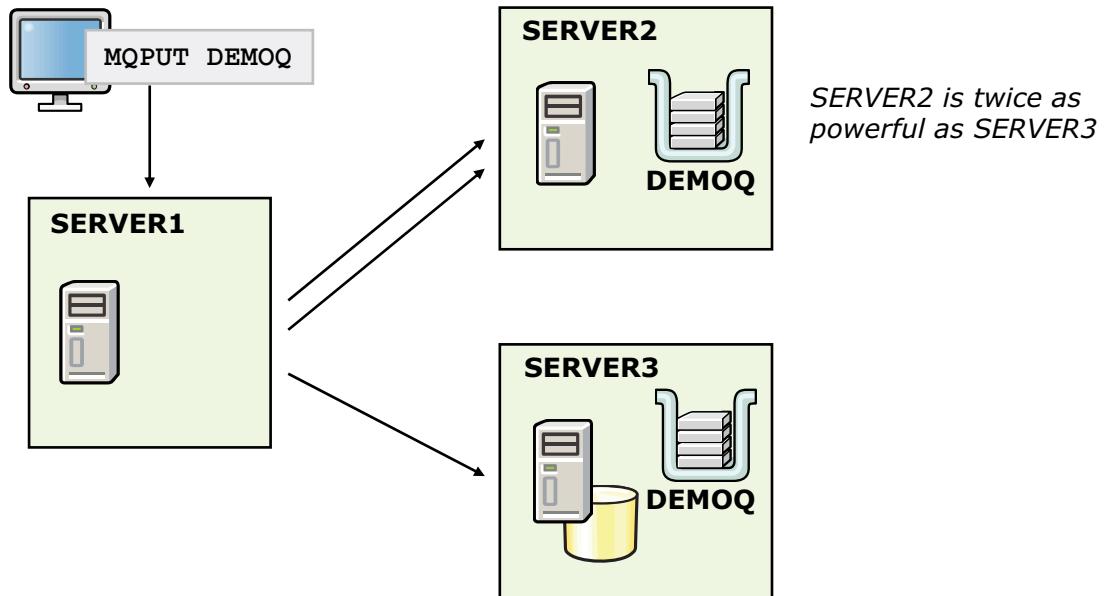


Note

The MRU channels cannot be reliably identified externally, although this information is available to cluster workload exits.

When IBM MQ uses MRU, the number of active channels a server can run do not limit the cluster size.

Using processing power to control the workload



How can you send SERVER2 twice as many messages?
Use **CLWLWGHT** attribute

Working with queue manager clusters

© Copyright IBM Corporation 2020

Figure 6-34. Using processing power to control the workload

This figure shows that one of the servers in the cluster has greater processing power, and the user wants to send a greater workload to the more powerful server.

You can use the cluster workload weight attribute to send more workload to a specific server and queue manager in the cluster.

Channel weight basics

- If more than one destination is valid, the round robin algorithm sends messages in numbers proportional to their channel weights
- Attribute **CLWLWGHT**
 - Range: 1 – 99
 - Default: 50

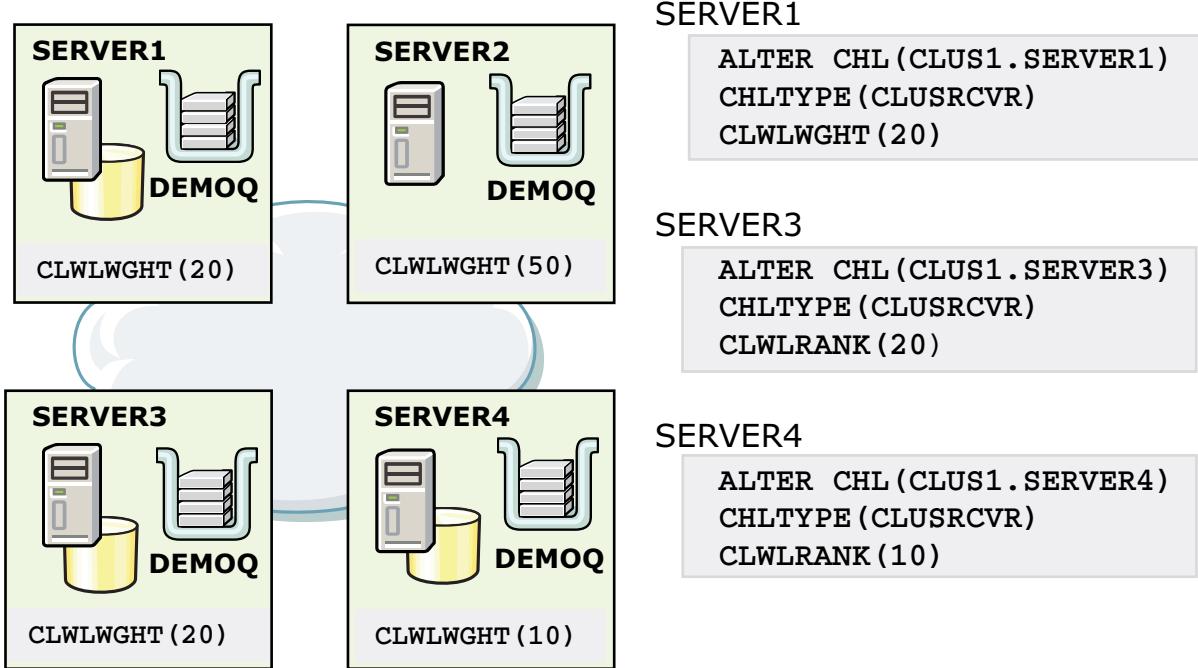
Example:

```
DEFINE CHL(CLUS1.SERVER1) CHLTYPE(CLUSRCVR) CLUSTER(CLUS1) ...
      CLWLWGHT( )
```

Figure 6-35. Channel weight basics

The cluster workload channel weight (CLWLWGHT) attribute causes the round robin algorithm to send messages in numbers proportional to their channel weights.

Channel weight example



Working with queue manager clusters

© Copyright IBM Corporation 2020

Figure 6-36. Channel weight example

By using cluster workload weight, the cluster workload management algorithm can favor more powerful servers.

In this example, the approximate percentages of messages that are distributed to each queue manager are as follows:

- SERVER1: 20%
- SERVER2: 50%
- SERVER3: 20%
- SERVER4: 10%

Unit summary

- Describe the components of a cluster
- Explain the purpose of full and partial repository queue managers
- Configure a basic cluster
- Outline cluster workload management features

Working with queue manager clusters

© Copyright IBM Corporation 2020

Figure 6-37. Unit summary

Review questions

1. True or False: A cluster should have two full repositories.
2. True or False: The SYSTEM.CLUSTER.COMMAND.QUEUE holds inbound and outbound administrative messages.
3. True or False: Remote queue definitions are not required when using clusters.
4. 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.



Working with queue manager clusters

© Copyright IBM Corporation 2020

Figure 6-38. Review questions

Write your answers here:

- 1.
- 2.
- 3.

Review answers

1. True or False: A cluster should have two full repositories.
The answer is True.
2. 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.
3. True or False: Remote queue definitions are not required
when using clusters.
The answer is True.
4. 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.



Working with queue manager clusters

© Copyright IBM Corporation 2020

Figure 6-39. Review answers

Exercise: Implementing a basic cluster

Working with queue manager clusters

© Copyright IBM Corporation 2020

Figure 6-40. Exercise: Implementing a basic cluster

Exercise introduction

- Create a simple queue manager cluster
- Test the cluster environment



Working with queue manager clusters

© Copyright IBM Corporation 2020

Figure 6-41. Exercise introduction

Unit 7. Publish/subscribe messaging

Estimated time

02:00

Overview

In this unit, you learn about the publish/subscribe support in IBM MQ. The unit describes how to use IBM MQ commands and IBM MQ Explorer to define and manage publications and subscriptions.

How you will check your progress

- Review questions
- Lab exercise

Unit objectives

- Describe publish/subscribe messaging
- Explain distributed publish/subscribe topologies
- Manage publish/subscribe topics, subscriptions, and topologies
- Compare publish/subscribe topologies

Publish/subscribe messaging

© Copyright IBM Corporation 2020

Figure 7-1. Unit objectives

Topics

- Introducing publish/subscribe
- Topic trees and topic strings
- Topic objects
- Managing subscriptions
- Managing publications
- Publish/subscribe topologies
- Comparing publish/subscribe topologies
- Troubleshooting

Publish/subscribe messaging

© Copyright IBM Corporation 2020

Figure 7-2. Topics

7.1. Introducing publish/subscribe

Introducing publish/subscribe

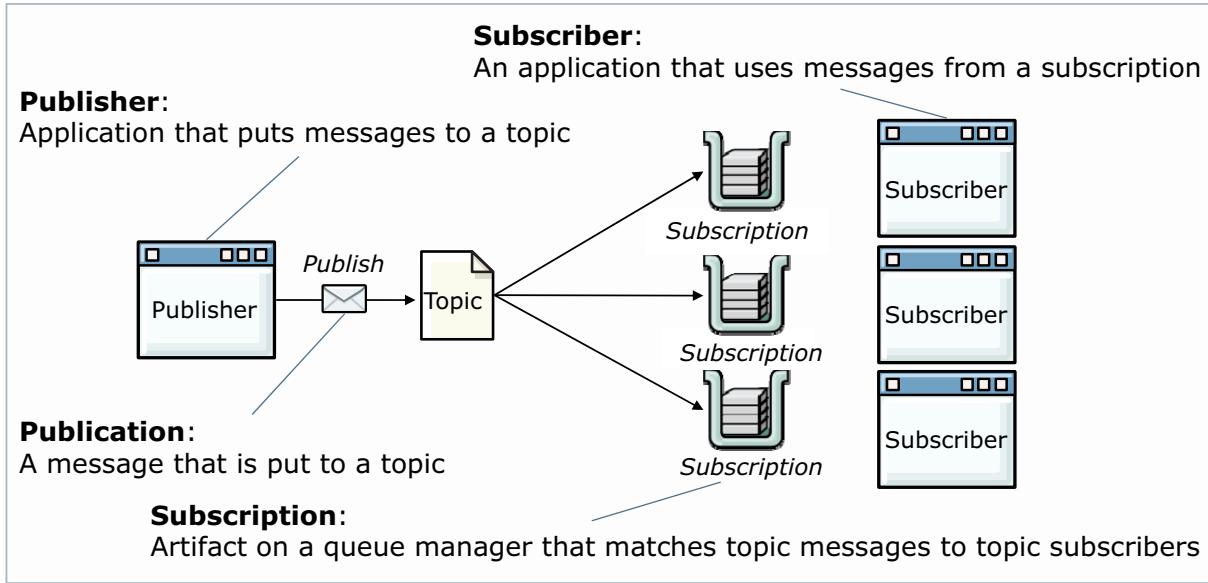
Publish/subscribe messaging

© Copyright IBM Corporation 2020

Figure 7-3. Introducing publish/subscribe

What is publish/subscribe?

- Message producers ***publish*** messages to a topic
- Message consumers receive messages through a ***subscription***
- ***Publishers*** are decoupled from ***subscribers***



Publish/subscribe messaging

© Copyright IBM Corporation 2020

Figure 7-4. What is publish/subscribe?

With publish/subscribe messaging, Instead of working directly with a queue, a message producer publishes messages to a topic. Message consumers receive messages by using *subscriptions*.

Each subscription is associated with a queue. Subscriptions are also associated with topics, and can match topic messages to interested consumers.

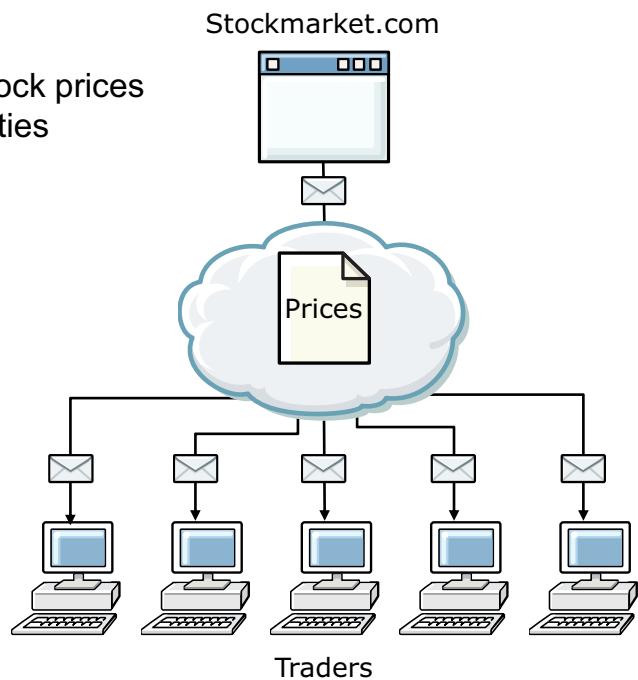
IBM MQ works out which subscriptions are interested in a topic, and delivers copies of the message to each subscription queue. There might be many, one, or no interested applications.

Publishers do not need to know the identity or location of the subscriber applications that receive the messages. Similarly, the subscribing applications do not need to know the identity or location of the publishing application.

Example: Stock market

- **Publisher:** Stockmarket.com
 - Publishes a continuous flow of stock prices
 - Prices delivered to interested parties

- **Subscribers:** Traders
 - Subscriptions to stock prices
 - Regularly receive updates



[Publish/subscribe messaging](#)

© Copyright IBM Corporation 2020

Figure 7-5. Example: Stock market

The most common example of a publish/subscribe system is one that provides stock-market information.

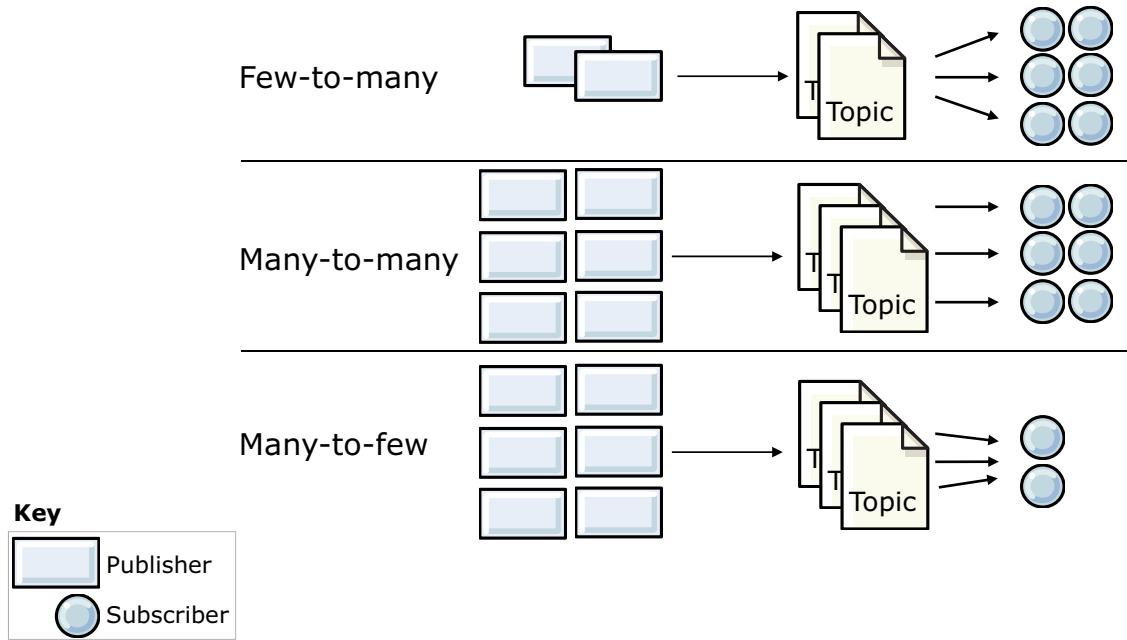
In this example, Stockmarket.com publishes a continuous flow of recent stock prices. Traders subscribe to these stock prices so that they can conduct trades.

Traders register their interest in (subscribe to) particular stocks and receive updates as prices change. Publishers are unaware of who receives their messages or when trader subscriptions are added or removed.

The terms "push" and "pull" describe the flow of information between applications. In this example, traders receive new information as soon as a stock price changes, so the information is *pushed* directly to them. This pushing of information from provider to consumer is one of the major differentiators between publish/subscribe and more conventional systems. The stock market example might also be designed so that updated stock prices flow to the traders only when they are requested, or *pulled* from a central repository (server) of all stock prices. In such a system, the traders must continually request a refresh of their stock prices. IBM MQ supports both modes of operation.

Publish/subscribe patterns

- Patterns can be FEW to MANY, MANY to MANY, MANY to FEW



Publish/subscribe messaging

© Copyright IBM Corporation 2020

Figure 7-6. Publish/subscribe patterns

Publishers are unaware of subscribers and do not know whether any applications subscribe to the topics that they publish to.

Publishing and subscribing are dynamic processes. New subscribers and new publishers can be added to the system without disruption.

As shown on the slide, various combinations of publish/subscribe are possible:

- One or more multiple publishing applications can provide information about each topic
- Zero or more subscribing applications can receive and process the information

For example, a many-to-few pattern can apply to a topic such as orders. Many orders might be published, but only one or few suppliers subscribe. A few-to-many pattern might apply to news topics, or a many-to-many pattern might apply to pricing topics.

7.2. Topic trees and topic strings

Topic trees and topic strings

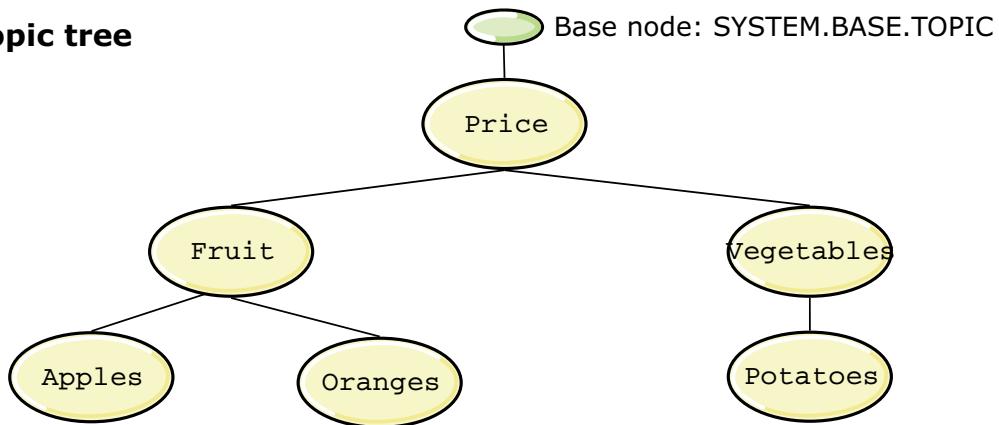
Publish/subscribe messaging

© Copyright IBM Corporation 2020

Figure 7-7. Topic trees and topic strings

It's all about the topic tree!

Topic tree



Topic strings

```

/Price/Fruit/Apples
/Price/Fruit/Oranges
/Price/Vegetables/Potatoes
/Price/Fruit/#
```

Publish/subscribe messaging

© Copyright IBM Corporation 2020

Figure 7-8. It's all about the topic tree!

The topic tree is the central control point for all publish/subscribe. Every queue manager has only one topic tree. By default, the topic tree has an empty base node called: SYSTEM.BASE.TOPIC

The base node, or root, is a blank topic. IBM MQ dynamically builds the topic tree by adding nodes as publishers publish and subscribers subscribe.

The topic tree is built by topic strings. A topic string is how humans read what is happening with publish/subscribe. A topic string takes the format:

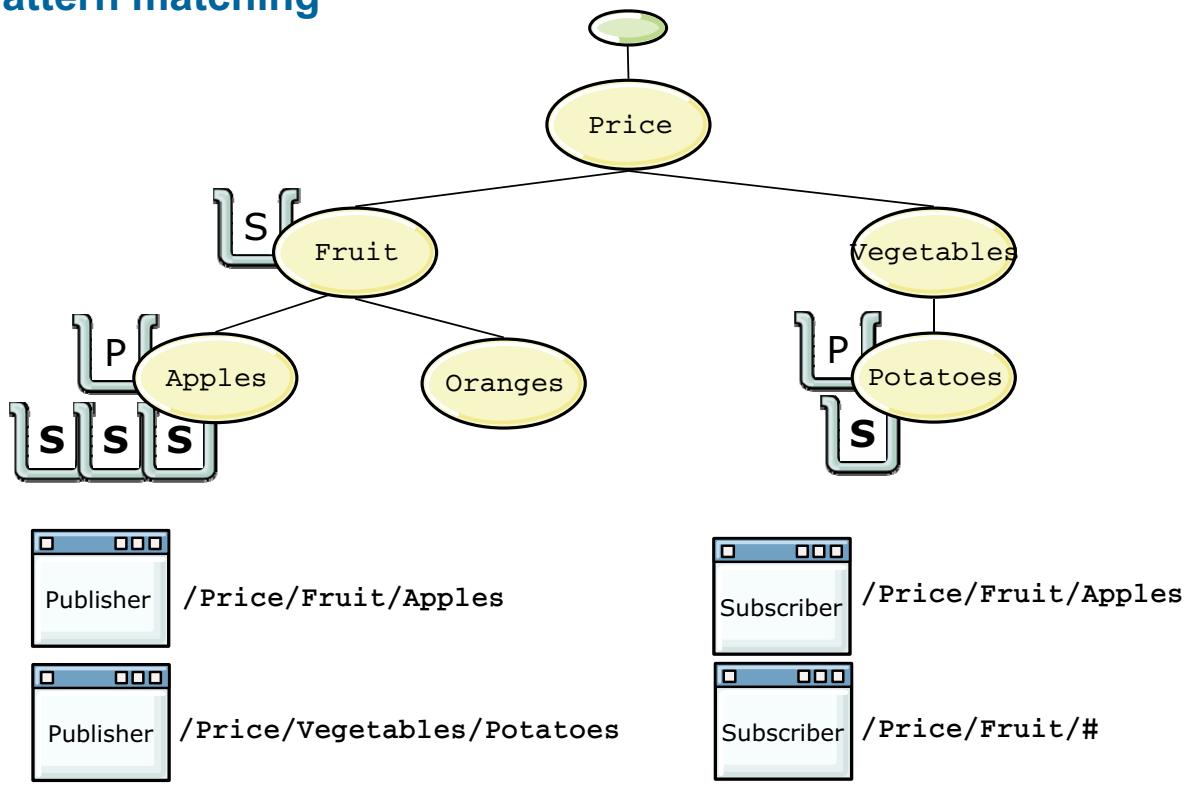
/Price/Fruit/Apples

While IBM MQ does not understand the string, it uses the forward slash (/) character to determine how many parts are in the topic tree. For example, the **/Price/Fruit/Apples** topic string has three forward slashes, which means the tree has three parts. MQ dynamically builds these parts into a topic tree, creating the nodes to match this pattern. If more topic strings with the same pattern are published or subscribed to, the tree is extended to include those nodes. So when a subscriber requests a subscription to **/Price/Fruit/Oranges**, the **Oranges** node is dynamically added to the topic tree.

Each topic that you define is an element, or topic, in the topic tree. The topic tree can be empty to start or it can contain topics that were previously defined by using IBM MQ commands. You can define a new topic by using the create topic commands or by specifying the topic for the first time in a publication or subscription.



Pattern matching



Publish/subscribe messaging

© Copyright IBM Corporation 2020

Figure 7-9. Pattern matching

The tree is dynamically built as needed. At run time, it allows publishers and subscribers to be matched up. When a subscription comes along, it maps a topic string onto a queue. IBM MQ finds the correct node in the tree, or creates the node if it does not already exist, and attaches the subscription to it. As more subscriptions come along, MQ attaches them to the correct place in the tree.

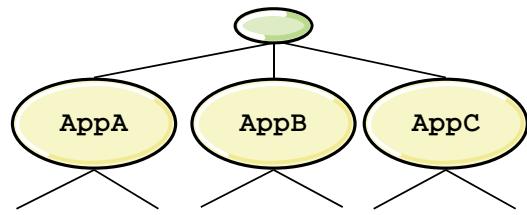
When a publisher has a message to publish, it needs to open a topic string and say what it will publish on. The message can then be attached to the right point in the tree.

When both the publisher and subscriber are attached to the correct node, the queue manager can efficiently deliver published messages to the subscriptions. It is the queue manager's job (or queue manager network when multiple queue managers are connected) to ensure that all subscribing applications with matching subscriptions receive the published message.

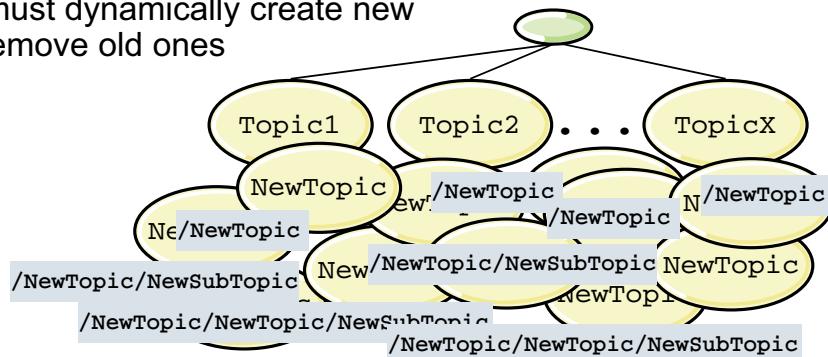
Wildcards in topic strings can be used for subscriptions to match against multiple topic strings that start with the same pattern. For example, use the hashtag (#) in the topic string /Price/Fruit/# to show interest in any topic that uses the /Price/Fruit pattern. Wildcards identify all matching nodes in the tree. If you publish on a lower node, a copy of the message is delivered to the wildcard subscriber. For example, if a publisher publishes to /Price/Fruit/Apples or /Price/Fruit/Bananas, the subscriber receives those messages.

Designing the topic tree: Make it extendible

- Start with a **high-level delimiter**
 - Clearly identify which application uses that part of the tree
 - Allows tree to be extended without interference between applications



- Avoid rapidly changing set of topic strings
 - Constantly extending the tree by publishing on unique topic strings
 - Queue manager must dynamically create new topic nodes and remove old ones



Publish/subscribe messaging

© Copyright IBM Corporation 2020

Figure 7-10. Designing the topic tree: Make it extendible

What's a good design for a topic tree?

Topic trees do not have a standard design. The tree design depends on the specific data. A well-structured topic hierarchy should be intuitive to the user, but also consider the later design considerations.

Make the tree **extendible**. A topic tree is used for the whole queue manager, so all applications that use publish/subscribe on that queue manager, share topic tree. If you have different applications that use different sets of topic strings, you don't want them to interfere with each other. For example, if you have applications that use the same underlying topic strings, publications might be delivered to the wrong application. You need to be able to separate them out. Even if you start off with a single application, you need to clearly identify what the topic strings are used for.

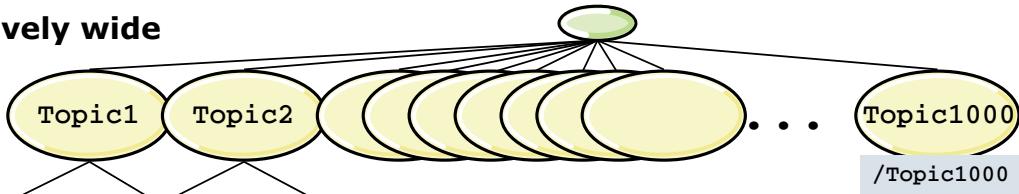
Consider using a high-level delimiter that clearly identifies what application it is used for. If you join multiple queue managers together, parts of the topic tree become global across all of those queue managers, at which point the topic tree must make sense across the IBM MQ network.

Due to the dynamic creation of nodes in the topic trees, you can, unknowingly continually **extend** the tree. Every time that you publish, you might be publishing on a new, unique topic string, which means the queue manager must create new topic nodes. The queue manager also automatically detects and cleans up old or unused nodes, similar to garbage collection. This monitoring of nodes prevents overusing system resources. All this dynamic activity means work for the queue manager.

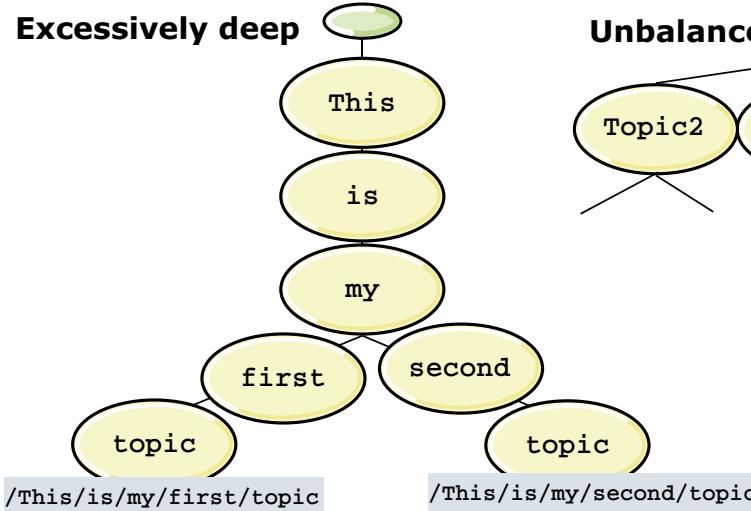
If you're rapidly publishing to new topic strings, consider the impact on the queue manager. To avoid performance problems, make sure that you consider these factors as you design your system.

Designing the topic tree: Shape

Excessively wide

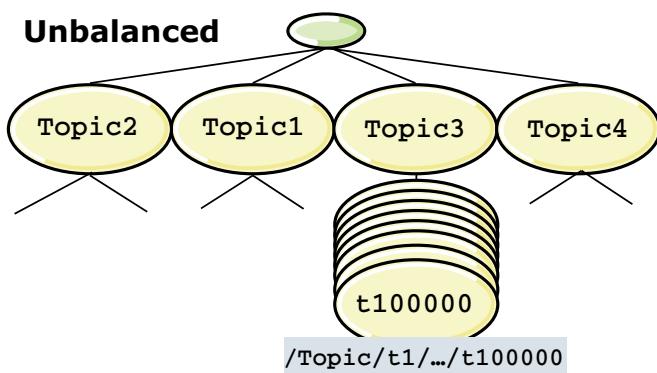


Excessively deep



Publish/subscribe messaging

Unbalanced



© Copyright IBM Corporation 2020

Figure 7-11. Designing the topic tree: Shape

The shape of the tree can also impact performance for dynamic topic trees.

Avoid making the tree excessively wide or deep. If you have extra layers in the tree that are not used for subscriptions or for wildcard matching, remove them. Otherwise, every time IBM MQ learns of a new topic string, MQ must generate the extra nodes for each forward slash in that string.

If your tree becomes unbalanced, with a single point in the tree that has thousands of nodes that hang from it, you might put undue stress on that point of the tree. For example, if you have a node such as USER ID or ACCOUNT REFERENCE, you might potentially end up with tens or hundreds of thousands of nodes hanging from that single point in the tree. It is better to separate those types of nodes into smaller subtrees.

Why worry about the size and shape of the topic tree?

- Simplified administration
 - Fewer topic nodes and topic objects mean fewer issues to manage
- IBM MQ can efficiently scale hundreds of thousands of topics, but some design considerations that can impact users
 - Broad topic trees put stress on certain parent topic nodes
 - Deep topic trees add unnecessary topics and levels
 - Many topics put a strain on system memory resources
 - Wildcard subscriptions at the beginning of a topic string (such as /Price/#) must be checked for almost all topic nodes that are created
- Topic tree management
 - Old or unused topic nodes must be removed at the same pace as new topic strings are published to ensure enough resources
 - Extra work for the queue manager due to design issues can impact performance

Figure 7-12. Why worry about the size and shape of the topic tree?

Why worry about the size and shape of the topic tree? These factors can simplify administration. The fewer topic objects and topics there are, the less there is to manage. The more topics, the more memory that is used by the queue manager.

IBM MQ can efficiently scale to tens or hundreds of thousands of topics but certain topic designs can cause problems and affect the user.

Consider how IBM MQ processes a publication:

- When an application opens a topic string, the topic string is looked up in a hash table.
 - More topic strings mean bigger hash tables and more memory consumption by the queue manager.
- If the topic string is not found, the topic string is stripped down to its '/' delimited parts and every new topic that is referenced in the topic string is dynamically created.
 - Each new node is linked to its parent.
- For every topic that is created, that branch of the topic tree is walked backwards to discover:
 - The node's configuration
 - The set of wildcard subscriptions that might match it

The queue manager scans topics periodically to discover whether they are still required. If a topic has no attached publisher or subscription, the topic is considered a *leaf* topic, or unnecessary. A leaf topic is also a topic that was not used for a specific time (minimum 30 minutes by default). The TREELIFE queue manager attribute specifies the time. Leaf topics are deleted and unlinked from their parent. This scanning is necessary to keep memory use under control.

Performance is affected when the queue manager's processing work for these tasks increased by:

- Broad topic trees
- Deep hierarchy in the topic tree
- Many topics that are not used

7.3. Topic objects

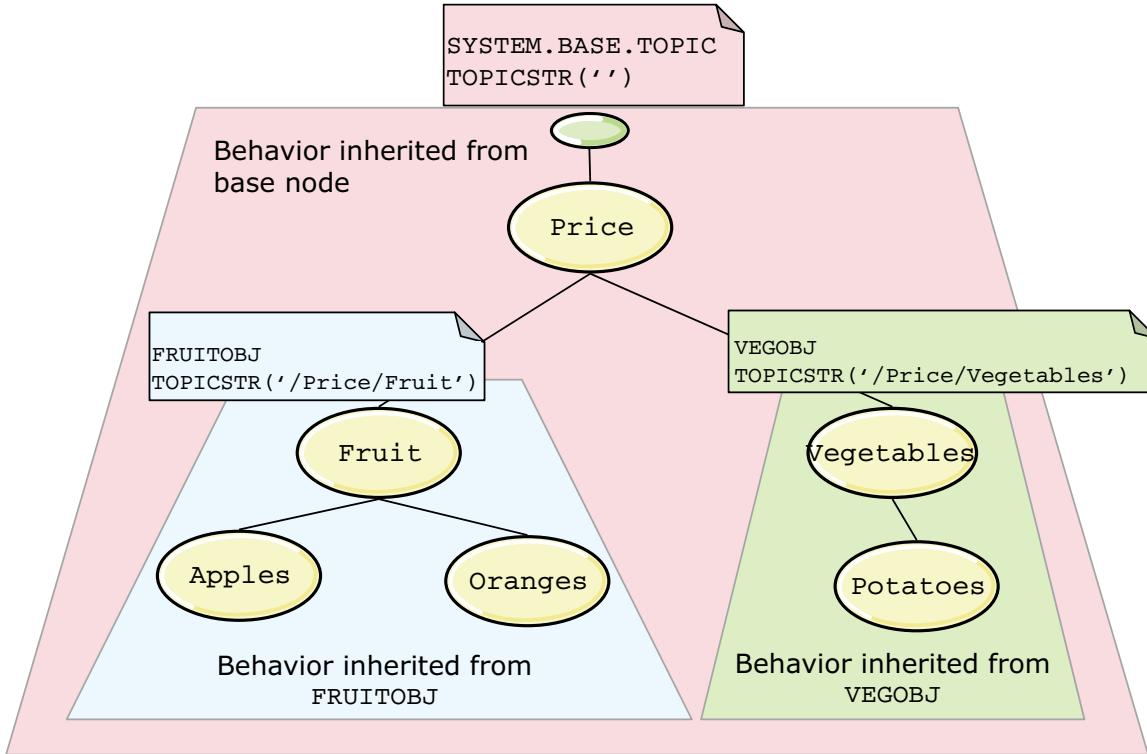
Topic objects

Publish/subscribe messaging

© Copyright IBM Corporation 2020

Figure 7-13. Topic objects

Topic objects



Publish/subscribe messaging

© Copyright IBM Corporation 2020

Figure 7-14. Topic objects

Topic objects are used to configure the behavior of a topic tree. You associate a topic object with a node in the topic tree as a point of administration.



Note

You are not required to define any **topic objects** to use publish/subscribe with IBM MQ.

If you have multiple applications that use separate branches of the tree, topic objects can be used to configure each of those branches so that they behave independently from each other.

When you first create queue manager, a default topic object, called SYSTEM.BASE.TOPIC, is defined for the whole topic tree. This base topic lives at the top of the topic tree, specified by a blank topic string: TOPICSTR('')

Any node that is generated underneath this topic object inherits the parent behavior, as indicated by the box around the tree.

To separate out parts of the tree for different behavior, you define a topic object on a node by defining a topic string that matches that point on the tree. In this example, the FRUITOBJ topic object is defined at the /Price/Fruit node. Underlying nodes inherit behavior from the FRUITOBJ

topic object, as indicated by the box around the Fruit branch of the tree. And the VEGOBJ topic object defines the behavior for the Vegetable branch of the tree. These objects mean you can define the publish/subscribe behavior for the price of Apples or Oranges in a different way than for Potatoes.

Topic object attributes

- Many attributes can be set on topic objects to change the behavior of a publisher or subscriber
- Dynamic nodes inherit their behavior from parent nodes
- ASPARENT value means that attribute value is inherited from the node above

Question: If you publish to /Price/Fruit/Oranges

- What message persistence is used?
- Is publication enabled?

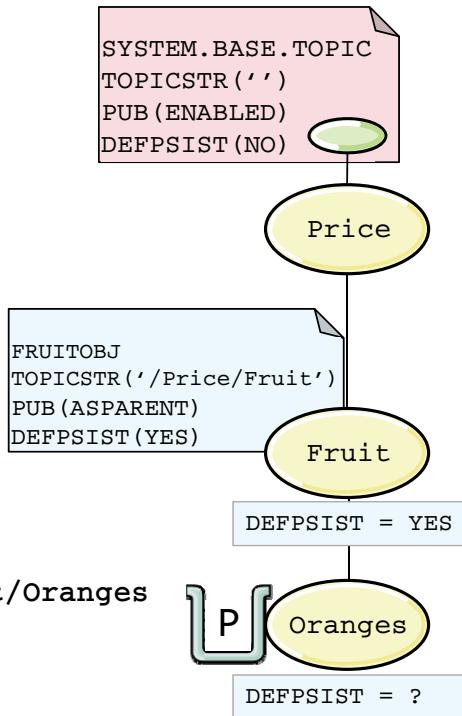


Figure 7-15. Topic object attributes

Many attributes can be set on topic objects to change the behavior of a publisher or subscriber. Dynamic nodes inherit their behavior from parent nodes. By default, a new topic object does not change anything.

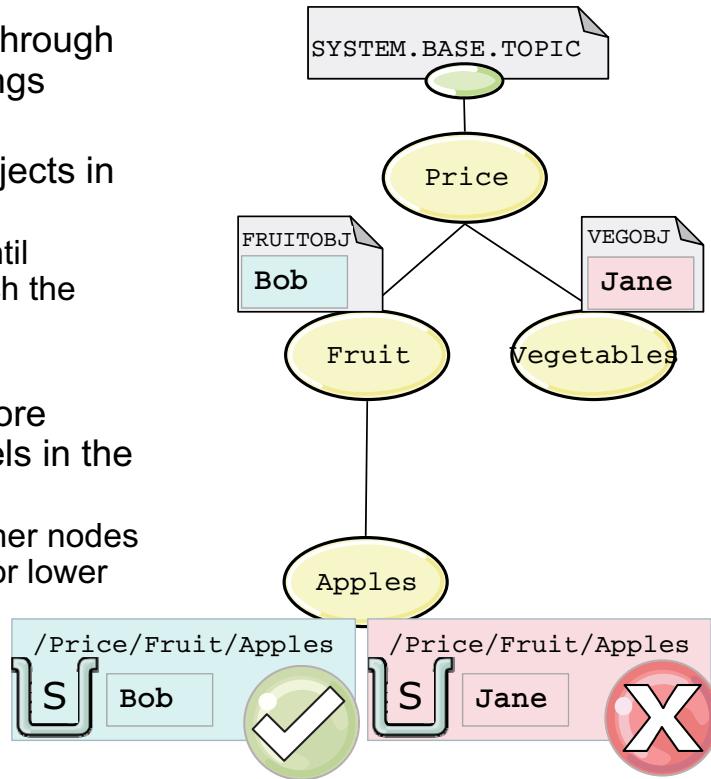
In this example, Price, Fruit, and Oranges inherit the attribute settings from SYSTEM.BASE.TOPIC.

To change the behavior for the Fruit node and nodes underneath it, you can set the topic object attributes on the FRUITOBJ object. In this example, the DEFPSIST (default persistence) attribute for Fruit is changed to YES. The PUB attribute for Fruit (which is the ability to publish, similar to "put" on a queue) is set to ASPARENT. The ASPARENT value means that this attribute is not set for this node. To know the value, you need to look at the node above. If the node above is also set ASPARENT, you keep moving up the tree to find the value. In this case, PUB is enabled on SYSTEM.BASE.TOPIC.

Use MQ Explorer to easily view topic status and see topic object attribute values.

Topic security

- Access control is managed through **topic objects**, not topic strings
- Authority checks on topic objects in the tree
 - Keep checking up the tree until authorization is found or reach the end of the tree
- Use caution when adding more access control at higher levels in the tree
 - Access that is granted at higher nodes allows wider authorizations for lower nodes



Publish/subscribe messaging

© Copyright IBM Corporation 2020

Figure 7-16. Topic security

Security is managed through topic objects. You use them to define who can publish and who can subscribe to specific topic strings in a tree. Rather than set security on specific topic strings, you define topic objects, place them in the tree, and then grant access to those topic objects.

In this example, to grant some users access to Fruit and others to Vegetables, you must create topic objects for those topic nodes, FRUITOBJ and VEGOBJ, and grant access individually to those objects.

For this example, Bob is granted access to Fruit, and Jane has access to Vegetables.

When Bob tries to subscribe to Apples, the first topic object, FRUITOBJ, is checked. In this case, access is granted in the topic object so the subscription is allowed.

When Jane also tries to subscribe to Apples, no access is found. Next, IBM MQ checks further up the tree at the next topic object, which is SYSTEM.BASE.TOPIC. No authorization is granted and the end of the tree was reached, so the subscription is prevented.

Make sure that you choose a suitable layer of the topic hierarchy and set access control at this layer. Use caution when granting access at higher levels in the tree. The higher the access is set, the more access to the tree is granted. Restrictions on lower nodes do not restrict access that is granted on higher nodes. Granting access to SYSTEM.BASE.TOPIC gives access to the entire tree.

7.4. Managing subscriptions

Managing subscriptions

Publish/subscribe messaging

© Copyright IBM Corporation 2020

Figure 7-17. Managing subscriptions

How do you create subscriptions

Two ways:

- Programmatically:
 - Applications can use an API to dynamically create and delete subscriptions
 - MQI uses MQSUB to create and MQCLOSE to delete
 - JMS uses `TopicSession.createDurableSubscriber()` and `TopicSession.unsubscribe()`
- Administratively:
 - An administrator defines subscriptions that can be accessed by applications
 - MQSC examples

```
DEFINE SUB('SUB1') DEST(Q1) OPICSTR('/Price/Fruit/Apples')
DELETE SUB('SUB1')
```

Figure 7-18. How do you create subscriptions

How do you retrieve the messages from the queue?

- By subscription

When you want messages, you open the subscription, you start accessing messages from subscription. You don't need to know about the queue or the queue name. Only one application can access a subscription at a time, unless you use JMS cloned or shared subscriptions

- From the queue

With unmanaged subscription queues, you can open the queue directly, which allows more freedom, and also allows for multiple concurrent consuming applications.

Subscription lifetime

- Durable
 - The subscription lifetime is not dependent on applications or connections
 - If an application disconnects and reconnects, messages continue to be delivered
- Non-durable
 - Created by applications only
 - The subscription lifetime is bound to the application connection
 - When the application disconnects, the subscription is deleted

Created by	Durable	Non-durable
Administrator	Yes	No
Application	Yes	Yes

Figure 7-19. Subscription lifetime

Subscriptions can be durable or nondurable. Durable subscriptions can be created by applications or by administrators.

If an application creates a durable subscription, then disconnects, the durable subscription continues to receive publications for the subscriber. When the application reconnects, it can process the messages that were delivered, so no messages are lost.

Non-durable subscriptions can be created only by an application, and the subscription is bound to that application connection. If the application disconnects, the subscription is automatically deleted, along with any messages that might be on it. If the application reconnects, no messages were delivered while it was disconnected.

You can use a topic object to prevent the creation of durable subscriptions for part of the topic tree. Set the topic object at the highest point where this behavior should start so the child nodes inherit this behavior.

Subscription queue management

- Managed
 - IBM MQ automatically creates a queue when you create a subscription
 - When you delete the subscription, the queue is also automatically deleted

- Unmanaged
 - Create your own queue and associate it with the subscription
 - When you delete the subscription, you must delete the queue
 - Allows you to access queue directly to get messages (as with point-to-point)

Created by	Managed		Unmanaged	
	Durable	Non-durable	Durable	Non-durable
Administrator	Yes	No	Yes	No
Application	Yes	Yes	Yes	Yes

Figure 7-20. Subscription queue management

Subscriptions are a mapping between topic strings and queues. So when you create a subscription, you also choose how you want to create the queue. You can choose managed or unmanaged.

When you create a managed subscription, IBM MQ automatically provides and manages the queue for that subscription. When the subscription is deleted, the queue is also automatically deleted. The durability of the subscription determines whether the managed queue remains when the subscribing application's connection to the queue manager is broken. For non-durable subscriptions, when the connection is closed, any unconsumed publications are automatically deleted, along with the queue.

If you choose unmanaged, you are responsible for creating the queue, and associating it with the subscription. While you have more control, you also have more to administer.

An advantage to using unmanaged queues is that you know the queue name and location, so you can open the queue directly to access messages, rather than accessing the messages only by the subscription. When messages are accessed through the subscription, only one application can access them. With unmanaged queues, you can enable multiple concurrent consuming applications or have multiple subscriptions (for different topic strings) that use a single queue. This type of queue usage becomes more like point-to-point, rather than publish/subscribe.

7.5. Managing publication

Managing publication

Publish/subscribe messaging

© Copyright IBM Corporation 2020

Figure 7-21. Managing publication

Publication success

- Persistence and transactions ensure integrity of successful publications
- What if publication fails?
 - Default behavior: Error returned when *persistent* messages cannot be delivered to one or more *durable* subscriptions
- Use topic objects to modify default behavior

Success or failure?	Persistent message published
Fail	Not delivered to one or more durable subscriptions
Success	Not delivered to one or more non-durable subscriptions
Success	Not delivered to one or more durable subscriptions but delivered to the dead-letter queue
Success	No matching subscriptions, no publications received

Figure 7-22. Publication success

Retained publications

- Publishers can choose to save a copy of each publication
 - A copy of the publication is distributed to all existing subscribers, plus an extra copy is saved for future subscribers
 - The queue manager stores retained publications on a queue
- Only one publication can be retained for each topic string
 - New publications overwrite the existing copy
- When a subscription is created, any matching retained message is delivered to it
- Do not confuse retained with persistent
 - Messages are saved after a system failure if they are persistent
 - You do not need retain publications in combination with persistence

Figure 7-23. Retained publications

By default, after a publication is sent to all existing subscriptions, the publication is discarded. When a new subscription is created, the subscriber must wait for the next publication. However, a publisher can specify that a copy of a publication be *retained*. As soon as a new subscription is created, a copy of the retained publication is delivered to the new subscription. Each new publication overwrites the previous one.

The queue manager stores retained publication messages on a queue. The queue manager can retain only one publication for each topic, so the existing retained publication of a topic is deleted when a new retained publication arrives at the queue manager. Using *retained* publications across thousands of topic strings means that the queue manager must store all that information, which is not the purpose of IBM MQ. The same recommendation of not building a deep application queue applies equally here. Calculate the total number of topic strings where publications would be retained when you consider the use of retained publications.

Messages can be retained after system failure when they are persistent, so retaining publications is not required for this purpose. You do not retain your publications and make them persistent.

7.6. Publish/subscribe topologies

Publish/subscribe topologies

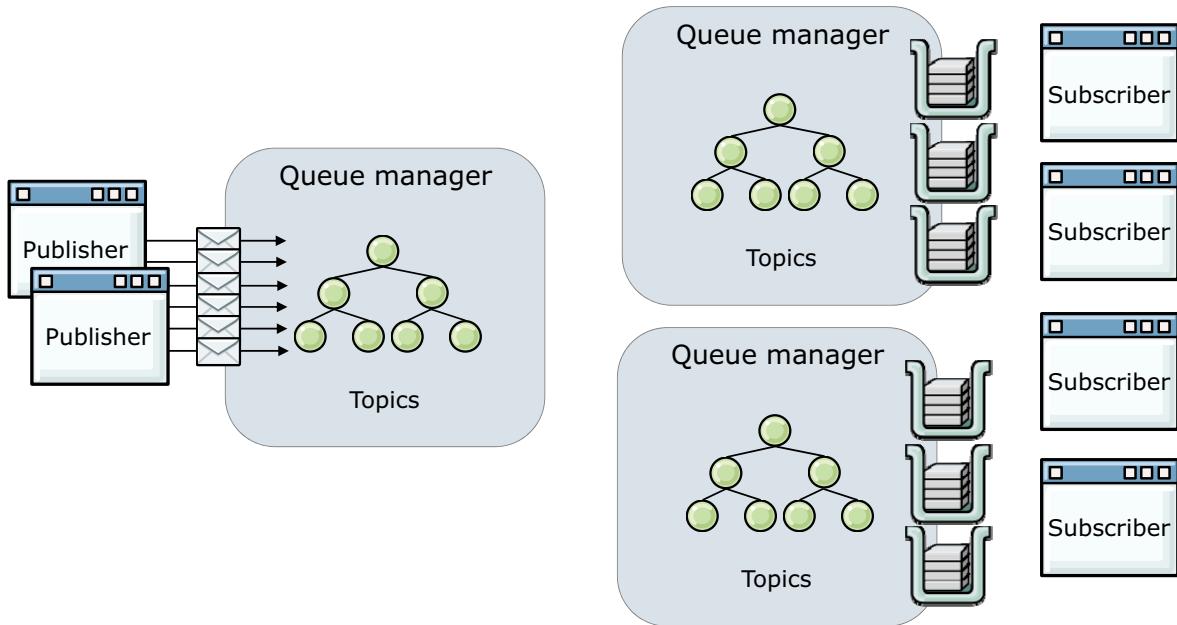
Publish/subscribe messaging

© Copyright IBM Corporation 2020

Figure 7-24. Publish/subscribe topologies

Distributed publish/subscribe

- Publish/subscribe on a single queue manager



Publish/subscribe messaging

© Copyright IBM Corporation 2020

Figure 7-25. Distributed publish/subscribe

When you do publish/subscribe on a single queue manager, all the publishers are connected to a single queue manager and all the subscriptions are defined on the same queue manager. It is simple for a queue manager to know where to deliver messages. Using MQ built-in topic tree structures, you can match message to subscriptions and deliver a copy to each message to each subscription.

To use a distributed environment with publishers connected to one queue manager and subscriptions defined on other queue managers, you want IBM MQ to work out automatically where it needs to deliver the messages.

The distributed environment must meet these goals:

- Identify where subscriptions are connected and deliver to the correct subscriptions in a reliable and efficient manner
- Applications do not need to know where subscriptions are located
- No configuration to manually specify where subscriptions are located

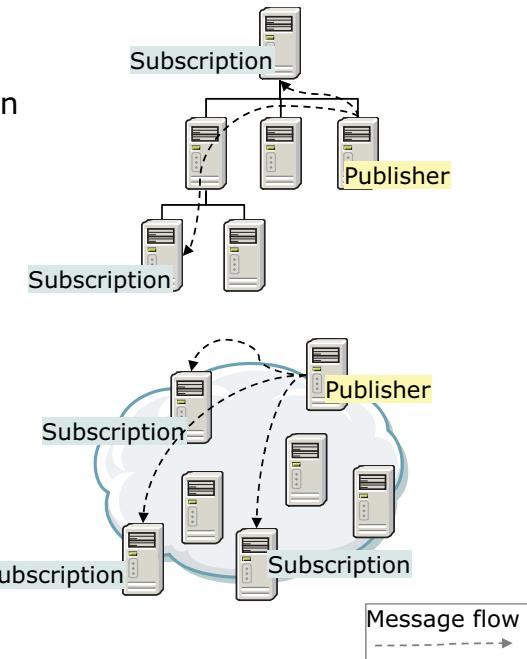
To achieve this goal, you can use these main topologies: hierarchy or cluster.

Distributed publish/subscribe topologies

- Publish/subscribe topologies can be created as a defined hierarchy or more dynamically as a cluster

Hierarchies

- Manually defines the relationship between each queue manager
- Messages flow via those relationships
- Parent-child relationship between queue managers



Clusters

- Any queue manager can publish and subscribe
- Published messages can flow directly between queue managers
- An existing cluster can be used for publish/subscribe

Publish/subscribe messaging

© Copyright IBM Corporation 2020

Figure 7-26. Distributed publish/subscribe topologies

The figure shows two main topologies for managing distributed publish/subscribe. The first topology uses the more traditional publish/subscribe hierarchies. The second topology is based on IBM MQ clusters.

- Hierarchies

Hierarchies are a way of manually defining connections between specific queue managers. You must manually identify which queue manager is the parent to each queue manager. You must also manually define channel connectivity. After you modify your queue managers to use the hierarchy, you can use the DISPLAY PUBSUB command to check the relationships.

When you create a subscription on queue manager in the hierarchy, any other queue manager can automatically send message to it. Messages flow from the publishing queue manager to the queue managers with the subscriptions by following the route of hierarchy definitions.

Messages do not flow directly from publishing queue manager to a subscription queue manager.

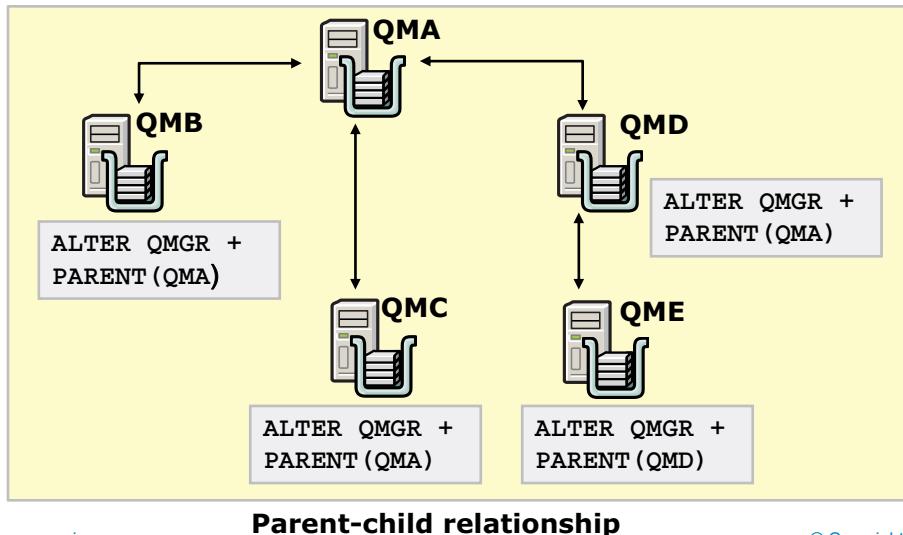
- Clusters

After you define a cluster of queue managers, more steps are required to enable publish/subscribe. You first create a topic object and associate it with the branch of your topic tree that is to be shared across all queue managers in cluster. The queue manager where the

topic object is defined sends that information to the full repositories in your cluster. The full repositories pass that information out to all other queue managers in the cluster.

Publish/subscribe hierarchies

- Manually defined relationship between each queue manager
- Messages flow by using those relationships
- Each queue manager must have publish/subscribe mode (PSMODE) enabled
- Use queue manager PARENT attribute to define relationships



Publish/subscribe messaging

© Copyright IBM Corporation 2020

Figure 7-27. Publish/subscribe hierarchies

A hierarchy manually defines the relationship between each queue manager. The queue manager relationships control the flow of messages.

By default, publications flow between queue managers from publishers to subscribers in a hierarchy.

When you implement a publish/subscribe hierarchy, consider the scalability of the *higher* nodes in the hierarchy, especially the root node as more publication traffic is expected to flow through these nodes.

The figure shows an example of publish/subscribe hierarchy. The PARENT queue manager identifies the relationship to the other queue managers in the hierarchy.

Channels must exist for the hierarchy to function.

- Transmit queues must have the same name as the remote queue manager.
- Each queue manager must be enabled for publish/subscribe with this command: **ALTER QMGR PSMODE(ENABLED)**

Publish/subscribe hierarchy configuration

- Every queue manager defines its *parent* in the hierarchy
- Each pair of directly related queue managers must be able to connect to each other by using IBM MQ channels
- By default all subscription knowledge is shared across the hierarchy
 - No requirement to define administered topic objects or modify applications
 - Controlled by *subscription scope* and *publication scope*
- After parent relationship, related queue managers should show each other when displaying PUBSUB status
- Informational messages are written to the queue manager logs when a relationship is established

Example:

AMQ5964: Pub/sub hierarchy connected.
EXPLANATION: A pub/sub hierarchy connection has been established with parent or child queue manager 'QMA'

Figure 7-28. Publish/subscribe hierarchy configuration

This figure lists some guidelines for defining a publish/subscribe hierarchy.

Publish/subscribe clusters

- Any queue manager can publish and subscribe to topics
- Published messages can go directly between queue managers or can be routed to specific queue manager
- Use **ALTER QMGR PSCLUS()** command to control whether this queue manager participates in publish/subscribe activity across any clusters in which it is a member

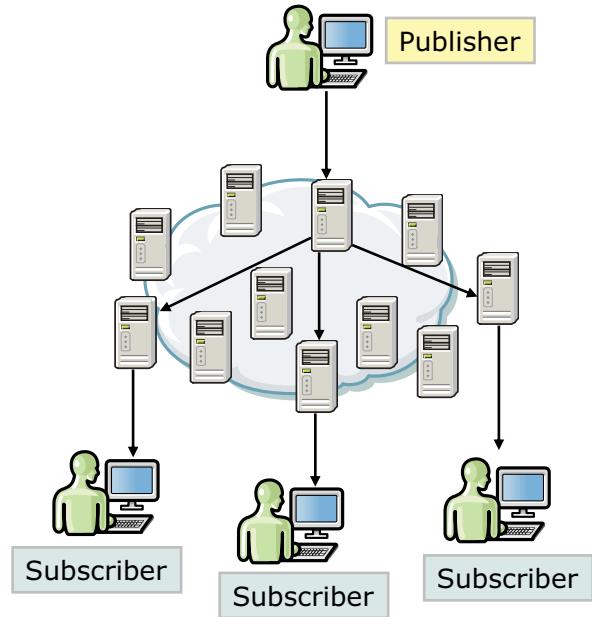


Figure 7-29. Publish/subscribe clusters

Using clusters in a publish/subscribe topology provide the following benefits.

- Messages that are destined for a specific queue manager in the same cluster can be transported directly to that queue manager and do not need to pass through an intermediate queue manager.
- There is no single point of failure in this topology. If one queue manager is not available, publications and subscriptions can still flow through the rest of the publish/subscribe system because each queue manager is directly connected with each other.
- If your clients are geographically dispersed, set up a cluster in each location, and connect the clusters (by joining a single queue manager in each cluster) to optimize the flow of publications and subscriptions.
- You can group clients according to the topics to which they publish and subscribe.
- A subscribing application can connect to its nearest queue manager to improve its own performance.
- The number of clients per queue manager can be reduced by adding more queue managers to the cluster to share workload, which makes a publish/subscribe cluster topology highly scalable.

If you have several queue managers in your publish/subscribe system, many channels are required to connect these queue managers together. However, the connections between queue managers can be created automatically to reduce the administrative work load.

Use the PSCLUS attribute on the ALTER QMGR command to control whether a queue manager participates in publish/subscribe activity across any clusters in which it is a member. No clustered topic objects can exist in any cluster when you change it from ENABLED to DISABLED.

Cluster topics

- Publish/subscribe in a cluster requires a *clustered* administered topic definition on *one* of the queue managers in the cluster

Example: `DEFINE TOPIC(FRUIT) TOPICSTR('/Price/Fruit') + CLUSTER(CLUS1)`

- Publish/subscribe cluster exists when one or more cluster topics exist
 - Cluster topic is propagated to all queue managers in the cluster with matching subscriptions
 - Channels are automatically defined between all queue managers in the cluster
- Every queue manager in the cluster automatically learns about the clustered topic
 - Shares subscription knowledge for any topic strings within that branch of the topic tree

Figure 7-30. Cluster topics

A publish/subscribe cluster is created when a clustered topic is defined. This definition is shared with all members of the cluster. So publications on the clustered topic are shared with all members of the cluster.

Publications do not flow between queue managers in a cluster unless the branch of the topic tree is clustered by setting the CLUSTER attribute of an administered topic object.

Like traditional clusters, publish/subscribe clusters are designed for a many-many queue manager connectivity. In traditional clusters, cluster objects are automatically defined based on usage, such as being put to a queue. The usage model is based on a putter that uses a set of cluster queues (which are not necessarily defined on all queue managers in the cluster). Therefore, in traditional clusters, it is unlikely that all queue managers are connected to all other queue managers by automatically defined channels.

In publish/subscribe clusters, cluster objects are automatically defined before usage at the time the first cluster topic is defined in the cluster because the usage model is different from traditional clusters. Channels are required from any queue manager through which a subscriber for a cluster topic is connected to all the other queue managers. The channels are required so that a proxy subscription can be fanned out to all the queue managers. Channels are also required back from any queue manager in which a publisher (for a cluster topic) is connected to those queue managers that have a connected subscriber. Therefore, in publish/subscribe clusters, it is much more likely that all of the queue managers are connected to all of the other queue managers by automatically

defined channels. For this reason, cluster objects are automatically defined before usage in publish/subscribe clusters.

To define a cluster topic, you must provide a topic object name, topic string, and cluster name. Make the queue manager on which the topic is defined a member of the specified cluster.

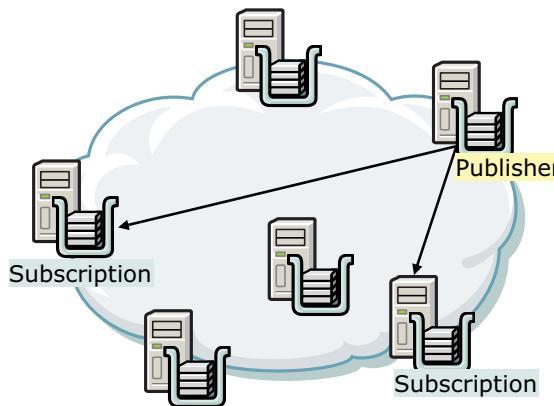
When displaying cluster topics, two types of objects are available.

- **Local:** Directly administrable objects
- **Cluster:** Cluster cache records, which are based on local objects

Cluster modes for routing publications

Direct routing

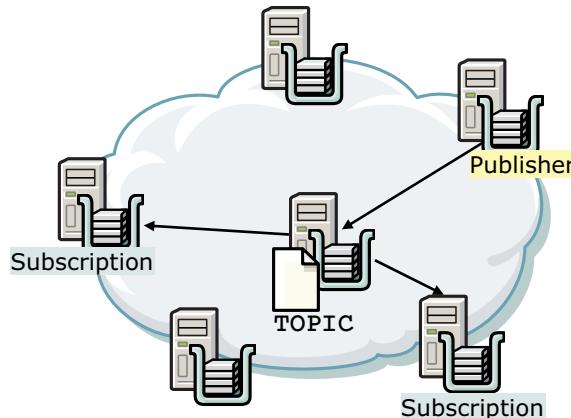
- Publications are sent directly from the publisher queue manager to every queue manager in the cluster with a matching subscription
- Default mode



[Publish/subscribe messaging](#)

Topic host routing

- Publications are routed through the queue manager where the clustered topic definitions are defined
- Multiple queue managers can host the same clustered topic



© Copyright IBM Corporation 2020

Figure 7-31. Cluster modes for routing publications

IBM MQ supports two types of routing for publish/subscribe clusters.

- Direct: In a direct routing cluster, all queue managers in the cluster are aware of every other queue manager and sends publications directly to other queue managers in the cluster with matching subscriptions. Publishers create channels directly to the queue manager with subscriptions. Direct routing is the default behavior.
- Topic host: Publishers set an attribute on a topic object. The queue manager where this topic object is defined is the topic host. Wherever you define that topic object, is where the messages are routed from. In a cluster that uses topic host routing, only the queue managers where the topic object is defined are aware of the other queue managers in the cluster. Publications are forwarded to these topic host queue managers, and they forward the publication to the queue managers with matching subscriptions.

Cluster topic routing

- Cluster topics can be defined on any queue manager in the cluster, which includes full or partial repository queue managers
- Identify topic message routing with **CLROUTE**
 - For direct routing (the default): **CLROUTE (DIRECT)**
 - For topic host routing: **CLROUTE (TOPICHOST)**
- Cluster topics can be defined on more than one queue manager
 - Make duplicate topic definitions identical
 - If a mismatch in the topic definitions is found, the conflict is reported
 - Typically used with topic host routing only

Example:

```
On QM1: DEF TOPIC(SPORTS) TOPICSTR('/Sport/Scores') +
CLUSTER(DEMO) CLROUTE(TOPICHOST)
On QM2: DEF TOPIC(SPORTS) TOPICSTR('/Sport/Scores') +
CLUSTER(DEMO) CLROUTE(TOPICHOST)
```

Figure 7-32. Cluster topic routing

A cluster topic host is a queue manager in which a clustered topic object is defined. Clustered topic objects can be defined on any queue manager in the publish/subscribe cluster. When at least one clustered topic exists within a cluster, the cluster is a publish/subscribe cluster.

To guard against a failure in a topic host routing cluster, you can identically define all clustered topic objects on two highly available queue managers. For example, take a scenario in which the single topic host of a clustered topic object is lost due to disk failure. Any cluster topic cache records that are based on the clustered topic object and that exist in the other queue managers' cluster cache are usable within the cluster for up to 30 days or until the cache is refreshed. The clustered topic object can be redefined on a functioning queue manager. If a new object is not defined, all members of the cluster report, up to 27 days after the host queue manager failure, that an expected object update was not received.

It is not necessary for full repositories and topic hosts to overlap, or be separated. In publish/subscribe clusters that have just two highly available servers among many servers, define both of the highly available servers as full repositories and cluster topic hosts. In publish/subscribe clusters with many highly available servers, define full repositories and cluster topic hosts on separate highly available servers. This configuration allows the operation and maintenance of one function without affecting the operation of the other functions.

Duplicating a topic on another queue manager is typically used for backup support when the publish/subscribe network used topic host routing. Direct routing makes the topic available to all queue managers in the cluster, so it is not necessary to define the same cluster topic.

Routing behavior for publish/subscribe clusters

- Setting the CLROUTE attribute at a point in the topic tree causes the entire branch beneath it to route topics by using that method
- All clustered definitions of the same named topic object in a cluster must have the same CLROUTE setting
- Verify the CLROUTE attribute for all topics on all hosts in the cluster with the MQSC command: **DISPLAY TCLUSTER (*) CLROUTE**
- After a topic object is clustered, you cannot change the value of the CLROUTE property unless you first remove the topic from the cluster (set CLUSTER to ' ' on the topic object)
- To stop a queue manager from acting as a topic host for a cluster topic either:
 - Delete the topic object
 - Use the PUB(DISABLED) attribute to quiesce message traffic for the topic

Figure 7-33. Routing behavior for publish/subscribe clusters

Topic host routing requirements

- Publications are forwarded to these topic host queue managers that then forward publications to queue managers with matching subscriptions
- Only the queue managers where the topic object is defined are aware of all other queue managers

Publish/subscribe messaging

© Copyright IBM Corporation 2020

Figure 7-34. Topic host routing requirements

For more information, see:

https://www.ibm.com/support/knowledgecenter/en/SSFKSJ_9.1.0/com.ibm.mq.pla.doc/q005133.htm

Subscription propagation

- Subscription for a topic string is created on a queue manager
- Other queue managers register *proxy subscriptions* to indicate interest in the topic on behalf of applications that subscribed locally
- When a message is published to the topic string, a copy of the message is sent over IBM MQ channels to each queue manager with a proxy subscription
- Receiving queue manager processes the message and gives a copy to each subscription it holds (including any proxies)

Figure 7-35. Subscription propagation

Proxy subscriptions are a special type of subscription that tells one queue manager that another queue manager needs a copy of any matching publications.

Proxy subscriptions can be viewed on a queue manager by using MQSC or IBM MQ Explorer. This proxy subscription shows which queue managers receive a publication for a topic.

If a queue manager has multiple subscriptions to the same topic string, a single proxy subscription is generated.

Proxy subscriptions do not include any selectors set on the subscription.

You view proxy subscriptions on a queue manager by using MQSC or IBM MQ Explorer.

7.7. Comparing topologies

Comparing topologies

Publish/subscribe messaging

© Copyright IBM Corporation 2020

Figure 7-36. Comparing topologies

Proxy subscription in a hierarchy

- Proxy subscriptions are sent from a queue manager to every **directly** connected queue manager in the hierarchy
- Each directly connected queue manager sends proxy subscriptions to other relations until the queue manager in the hierarchy knows about the topic string
- Publications are sent down the path of proxy subscriptions

Figure 7-37. *Proxy subscription in a hierarchy*

In a hierarchy, proxy subscriptions are sent from a queue manager to every directly connected queue manager in the hierarchy.

Proxy subscriptions in a cluster

- Direct routed cluster topics
 - Proxy subscriptions are sent from a subscribing queue manager to **every** other queue manager in the cluster
 - Every queue manager is aware of every other queue manager in the cluster
 - Any publication from a queue manager is sent directly to those queue managers that sent proxy subscriptions
- Topic host routed cluster topic
 - Proxy subscriptions are sent from a subscribing queue manager to **only** those queue managers that host a definition of the clustered topic
 - Only the topic hosts are aware of every other queue manager in the cluster
 - Any publication from a queue manager is **always** sent to one of the topic hosts, who then forwards the message to any subscribing queue managers

Figure 7-38. Proxy subscriptions in a cluster

In a direct routed publish/subscribe cluster, proxy subscriptions are sent from a subscribing queue manager to every other queue manager in the cluster.

In a topic host routed publish/subscribe cluster, proxy subscriptions are sent from a subscribing to only those queue managers that host a definition of the clustered topic only.

In a direct routed publish/subscribe cluster, every queue manager knows about every other queue manager in the cluster. In a topic host routed publish/subscribe cluster, only the topic hosts are aware of every other queue manager in the cluster.

Scaling

- Direct routed clusters
 - Share subscription knowledge across all queue managers
 - Result in the establishment of many queue manager channels, even to queue managers with no publishers or subscribers
- Topic routing clusters
 - Requires few channel connections
 - Hosting different topics on different queue managers can increase scaling
- Hierarchies
 - Channel connectivity is restricted
 - Share subscription knowledge across all queue managers

Figure 7-39. Scaling

This figure compares the scalability for each distributed publish/subscribe topology.

Publication flows

- Direct routed clusters
 - Ensure that publications take the shortest path between publishers and subscribers
- Topic host routed clusters
 - Can introduce an extra hop between publishers and subscribers
 - Requires careful placement of subscribers so that publishers can achieve the same single-hop route that direct clusters provide
- Hierarchies
 - Can have multiple hops between publishers and subscribers

Publish/subscribe messaging

© Copyright IBM Corporation 2020

Figure 7-40. Publication flows

This figure compares how the supported publish/subscribe topologies handle publication flows. Direct routed clusters ensure that the publications take the shortest path between publishers and subscribers. The other topologies can introduce one or more extra hops between publishers and subscribers.

Configuration

- Direct routed clusters
 - Allow queue managers to join and leave without much consideration after the overall plan is defined
- Topic host routed clusters
 - Allow queue managers to join and leave without much consideration after the overall plan is defined
 - Careful planning of topic host routers is required
- Hierarchies
 - Requires detailed planning and configuration of queue managers
 - Harder to alter the hierarchical structure

Publish/subscribe messaging

© Copyright IBM Corporation 2020

Figure 7-41. Configuration

This figure compares how configuration differs for the supported publish/subscribe topologies.

Direct routed clusters require the least amount of configuration. Queue manager join and leave the clusters without affecting the network.

Topic host routed clusters require some planning and configuration of topic host routers. Care must be taken when you remove queue managers from this type of topology.

Hierarchies require the most configuration and planning. It is also harder to add or remove queue managers from a publish/subscribe hierarchy without affecting other queue managers in the topology.

Availability

- Direct routed cluster
 - Queue manager that is not available affects the subscribers and publishers on that queue manager only
- Topic host routed cluster
 - If a queue manager that is not hosts the only definition of a topic, that queue manager can affect other queue managers
 - Hosting different topics on different queue managers can increase availability
- Hierarchies
 - Queue manager that is not available can cut off many other queue managers

Figure 7-42. Availability

This figure compares the effects of an unavailable queue manager in the different publish/subscribe topologies.

Comparison summary

- Scaling
 - Topic host routed clusters provide the best options for scaling
- Availability
 - Clusters can be configured to avoid single points of failures, unlike hierarchies
- Publication performance
 - All topologies use similar queue manager techniques for transferring messages
 - All topologies can be designed to minimize routes between queue managers
- Configuration
 - Clustering is the most dynamic and simplest solution, especially where clusters are already in use

Figure 7-43. Comparison summary

How do the three topologies compare?

Publish/subscribe clusters are highly available because the cluster is fully connected. Publication delivery is also fast due to direct connections. They are flexible, scalable, and have simpler administration. Topic host routed clusters provide the best options for scaling.

Hierarchies create proxy subscriptions when each queue manager is sent a proxy subscription. A hierarchy is scalable but the reorganization of the hierarchy can be labor-intensive. Availability is impacted if intermediate nodes are not running.

7.8. Troubleshooting

Troubleshooting

Publish/subscribe messaging

© Copyright IBM Corporation 2020

Figure 7-44. Troubleshooting

Tips

- Start small
- Do not put the root node, indicated by a forward slash (/), into a cluster
 - Make global topics obvious, such as /global or /cluster
- Use care when designing deep hierarchies
 - Shallow hierarchy with a highly available parent works well
- Monitor the depth of transmission queues
 - SYSTEM.CLUSTER.TRANSMIT.QUEUE
 - Hierarchy transmit queues
- Manage subscriptions
 - Do you need durable subscribers?
 - Did you stop subscribers before stopping the queue manager?
- Be careful when mixing traditional clusters
 - Large clusters and cluster topic objects require many channels

[Publish/subscribe messaging](#)

© Copyright IBM Corporation 2020

Figure 7-45. Tips

Start small with your choice of distributed clustering. It is easy to add nodes to the distribution topology when you use clustering or a hierarchy.

Do not put the topic root node forward slash (/) into a cluster. Create some *base* or *global* topic strings like /*global* or /*cluster*.

Be careful of deep hierarchies. Leaf node publications must traverse the tree to the destination. It is better to have a shallow hierarchy with highly available parent nodes.

Monitor the depth of transmission queues. Unavailable queue managers or channels can result in a significant buildup of undelivered publications that wait on the transmission queues. For clustered topologies, message buildup is apparent for SYSTEM.CLUSTER.TRANSMIT.QUEUE.

Stop subscriptions before you stop a queue manager to prevent a buildup on the transmission queues.

Use durable subscriptions when necessary only.

Be careful when you use existing clusters for publish/subscribe clustering. Publish/subscribe clusters must be fully connected. Many channels are automatically defined.

Problem determination (1 of 2)

- Default behavior in IBM MQ depends on the persistence of the message and the durability of the subscription
 - For persistent messages (**PMSGDLV=ALLDUR**), publish fails if it cannot be delivered to one or more **durable** subscriptions; failures to any **non-durable** subscriptions are *acceptable*
 - For non-persistent messages (**NPMSGDLV=ALLAVAIL**), any failures to subscriptions are acceptable, the publish succeeds (even if no subscribers can receive the message)
- If a dead-letter queue is configured, a failure to put a publication to a subscription queue results in an attempt to put it to the dead-letter queue
 - Put to dead-letter queue is classified as a success
 - Controlled by using the **USEDLQ** option on a topic definition

Figure 7-46. Problem determination (1 of 2)

The default behavior for publish/subscribe depends on the persistence of the message and the durability of the subscription. This behavior can be changed through configuration of the PMSGDLV and NPMSGDLV attributes of a topic.

If a dead-letter queue is configured, a failure to put a publication to a subscription queue results in an attempt to put it to the dead-letter queue.

Problem determination (2 of 2)

- Double check your configuration
- When in a cluster, check each cluster knowledge of the queue manager
- When in a hierarchy, check the queue manager relationships
- Check the channels
- Check the proxy subscriptions
- Check the topic status
- Watch for publication movement
- Look for a buildup of messages on a queue

[Publish/subscribe messaging](#)

© Copyright IBM Corporation 2020

Figure 7-47. Problem determination (2 of 2)

Double check your configuration. Topic objects inherit behavior from higher topics, which can affect how lower topics behave.

When in a cluster, check each queue manager's cluster knowledge. One option for checking the queue manager's cluster knowledge is by using the DISPLAY TOPIC(*) TYPE(CLUSTER) command.

When in a hierarchy, check the queue manager relationships by using the DISPLAY PUBSUB command or IBM MQ Explorer.

Check your channels. If the channels to and from the queue managers are not running, publication traffic, proxy subscriptions, topic configuration (clusters), or relationships (hierarchies) do not flow.

Check the proxy subscriptions on each queue manager by using the DISPLAY SUB(*) SUBTYPE(PROXY) command.

Check the topic status. Use the DISPLAY TPSTATUS() command to show the behavior of a topic string.

Watch for publication movement.

- Check the NUMMSGS value on the proxy subscriptions by using DISPLAY SBSTATUS to see whether publications were sent to the originator of the proxy subscription.

- Check the MSGS value on the channels to see whether they are flowing by using the DISPLAY CHSTATUS command.

Look for a build-up of messages. When problems exist, messages can build up on system queues. A build-up can occur when the system produces messages too quickly or an error occurs. Investigate to see whether messages are still being processed, but slowly, or no messages are processed.

Queues that should be empty when the publish/subscribe is running correctly are:

- SYSTEM.CLUSTER.TRANSMIT.QUEUE (or any other transmission queue involved): All queue manager outbound messages to other queue managers
- SYSTEM.BROKER.CONTROL.QUEUE: Requests to register/deregister subscriptions

If you are using publish/subscribe clusters, the following queues should also be empty:

- SYSTEM.INTER.QMGR.PUBS: Publications arriving from remote queue managers
- SYSTEM.INTER.QMGR.FANREQ: Requests to send proxy subscriptions
- SYSTEM.INTER.QMGR.CONTROL: Requests from other queue managers to register proxy subscriptions

If you are using publish/subscribe hierarchies, the following queues should also be empty:

- SYSTEM.BROKER.INTER.BROKER.COMMUNICATIONS: Requests from other queue managers in the hierarchy
- SYSTEM.BROKER.DEFAULT.STREAM: Publications arriving from remote queue managers (and local ‘queued’ publish/subscribe applications)

Loop detection

- Loops in the publish/subscribe network are not good
 - Subscribers receive multiple copies of the same publication
 - It is detrimental to other workloads
 - Takes up system resources such as the processor, network, and IBM MQ processes
- Messages are fingerprinted
 - Outbound of publish/subscribe cluster with cluster name
 - On each hierarchy queue manager with queue manager name

Publish/subscribe messaging

© Copyright IBM Corporation 2020

Figure 7-48. Loop detection

When you create a cluster, it is possible to create a loop that causes messages to cycle forever within the network.

As publications move around a publish/subscribe topology, each queue manager adds a unique fingerprint to the message header. Whenever a publish/subscribe queue manager receives a publication from another publish/subscribe queue manager, the fingerprints that are held in the message header are checked. If its own fingerprint is already present, the publication fully circulated around a loop, so the queue manager discards the message, and adds an entry to the error log.

In a distributed publish/subscribe network, it is important that publications and proxy subscriptions cannot loop. Looping results in a flooded network with connected subscribers that receive multiple copies of the same original publication.

Stopping a queue manager

- Disconnect subscribers before stopping a queue manager or channel initiator
 - When stopping a queue manager, local subscribers are disconnected, but they might not notify other queue managers in the publish/subscribe cluster or hierarchy
- If subscribers are not disconnected first
 - Durable subscribers: Transmission queue build-up is required
 - Nondurable subscribers: Transmission queue build-up might occur
- Resynchronize on start
- Try to avoid backlog of messages on SYSTEM.CLUSTER.TRANSMIT.QUEUE and hierarchy transmission queues

Figure 7-49. Stopping a queue manager

Shutting down a queue manager at a busy moment can result in a buildup of messages on the various transmission queues.

To avoid a buildup of messages, disconnect any subscribers before you shut down a queue manager. Dropping the subscriptions results in the removal of proxy subscriptions on remote queue managers and the prevention of message buildup on transmission queues.

Durable subscriptions are not dropped.

Unit summary

- Describe publish/subscribe messaging
- Explain distributed publish/subscribe topologies
- Manage publish/subscribe topics, subscriptions, and topologies
- Compare publish/subscribe topologies

Publish/subscribe messaging

© Copyright IBM Corporation 2020

Figure 7-50. Unit summary

Review questions

1. True or False: Publishers need information about subscribers before they can publish to a topic.
2. True or False: Always define topic objects for new topic nodes.
3. True or False: Security is defined on topic strings.



Publish/subscribe messaging

© Copyright IBM Corporation 2020

Figure 7-51. Review questions

Write your answers here:

- 1.
- 2.
- 3.

Review answers

1. True or False: Publishers need information about subscribers before they can publish to a topic.
The answer is False. Publishers do not need to know the identity or location of the subscriber applications that receive the messages.
2. True or False: Always define topic objects for new topic nodes.
The answer is False. If behavior changes are not required, do not create unnecessary topic objects, which adds administrative overhead and possible confusion.
3. True or False: Security is defined on topic strings.
The answer is False. Access control is defined on topic objects, not topic strings.



Exercise: Configuring publish/subscribe message queuing

Publish/subscribe messaging

© Copyright IBM Corporation 2020

Figure 7-53. Exercise: Configuring publish/subscribe message queuing

Exercise introduction

- Define a direct route publish/subscribe cluster
- Define a topic host route publish/subscribe cluster
- Test the publish/subscribe cluster
- Use the IBM MQ display route (dspmq rte) command to verify the route that the message takes through the publish/subscribe cluster



Publish/subscribe messaging

© Copyright IBM Corporation 2020

Figure 7-54. Exercise introduction

Unit 8. Implementing basic security in IBM MQ

Estimated time

01:30

Overview

In this unit, you learn how IBM MQ protects its objects by using access control lists (ACLs), and how the IBM MQ Object Authority Manager (OAM) uses these ACLs when a user attempts to access these objects.

How you will check your progress

- Review questions
- Lab exercise

Unit objectives

- Describe how the object authority manager (OAM) provides security for IBM MQ resources
- Protect IBM MQ resources by using the OAM
- Implement basic channel authentication

Implementing basic security in IBM MQ

© Copyright IBM Corporation 2020

Figure 8-1. Unit objectives

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 8-2. 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 2020

Figure 8-3. 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

Figure 8-4. 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 2020

Figure 8-5. 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.

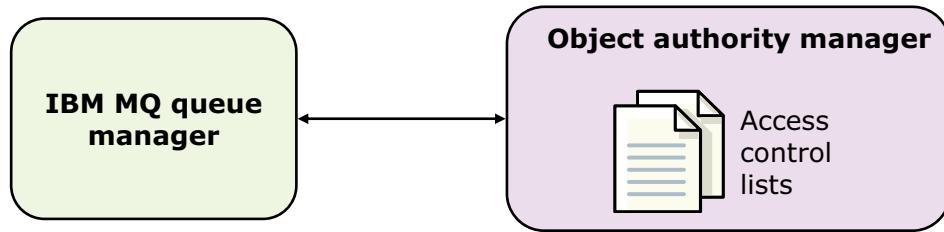
All queue managers on distributed operating systems provide access control facilities to control which users have access to which MQ resources. These queue managers use the OAM component of MQ to provide access control for MQ resources. By using the OAM, the MQ administrator can control access at the 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

Figure 8-6. 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 2020

Figure 8-7. *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 2020

Figure 8-8. 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 8-9. 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 (dlt) 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 8-10. Set or reset authorization

Three control commands provide control and verification of the security environment for MQ: **setmqaut**, **dspmqaut**, and **dmqmqaout**. 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 8-11. 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 8-12. 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 8-13. 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.

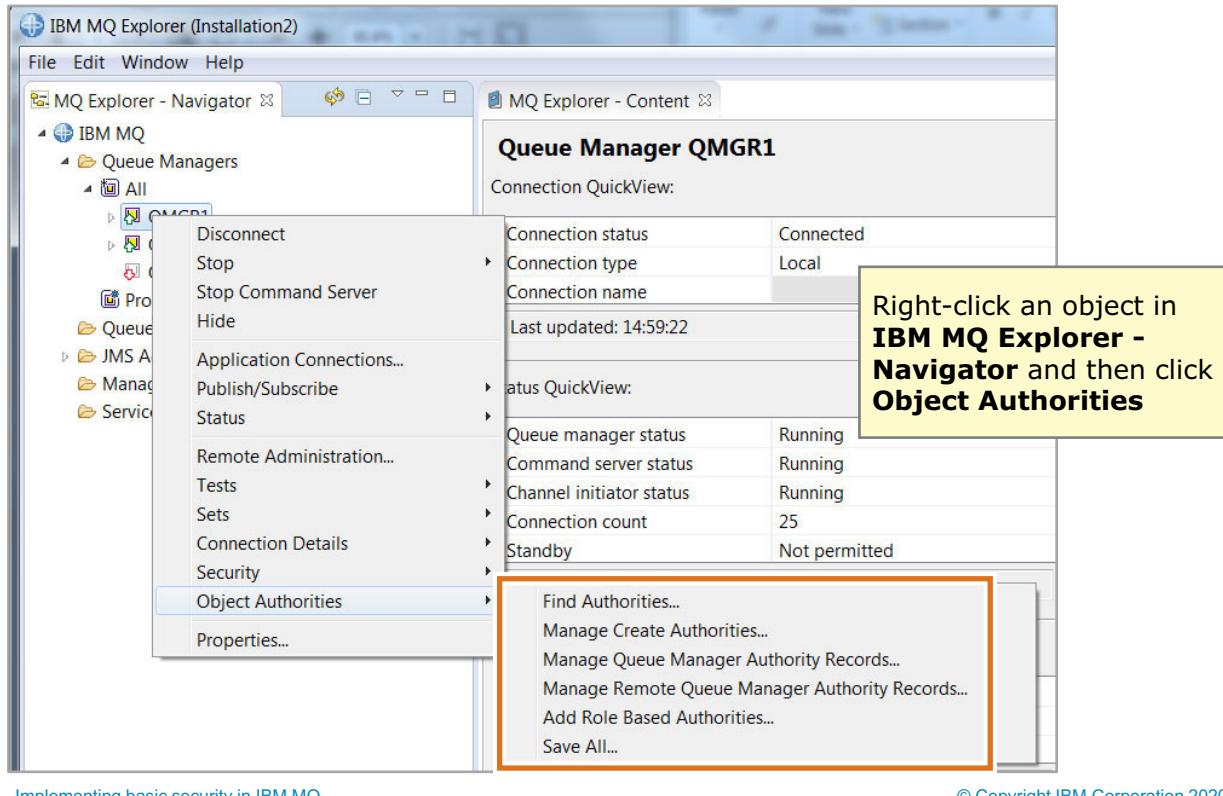
The list of authorizations to add and the list of authorizations to remove must not overlap. For example, you cannot add “display” authority and “remove display” authority with the same command. This rule applies even if the authorities are expressed by using different options.

For example, the following command fails because DSP authority overlaps with ALLADM authority:

```
SET AUTHREC PROFILE(*) OBJTYPE(QUEUE) PRINCIPAL(PRINC01) AUTHADD(DSP)
AUTHRMV(ALLADM)
```

IBM Training

Using IBM MQ Explorer to manage authorization



Implementing basic security in IBM MQ

© Copyright IBM Corporation 2020

Figure 8-14. Using IBM MQ Explorer to manage authorization

As an option, you can manage object authorities with MQ Explorer.

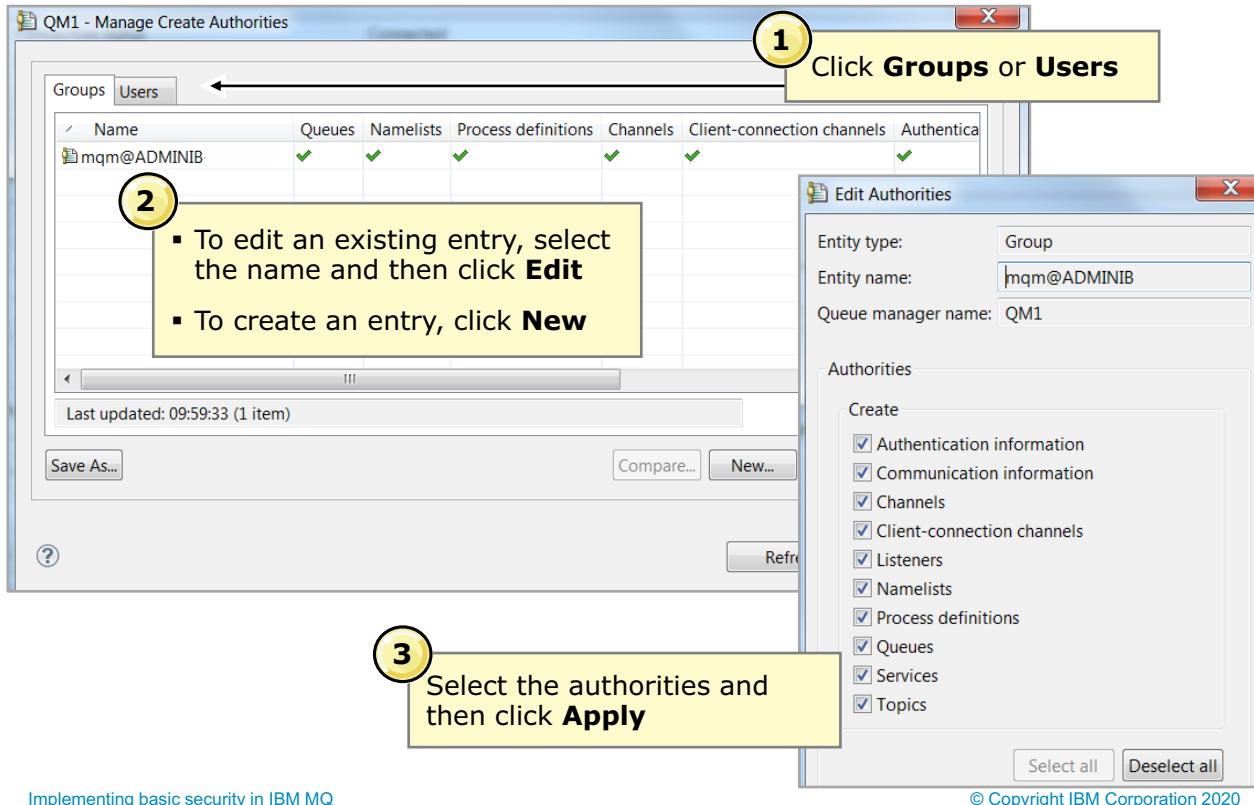
In MQ Explorer, you can:

- 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 2020

Figure 8-15. Queue manager authorization in IBM MQ Explorer

To modify an existing authority record, complete the following steps:

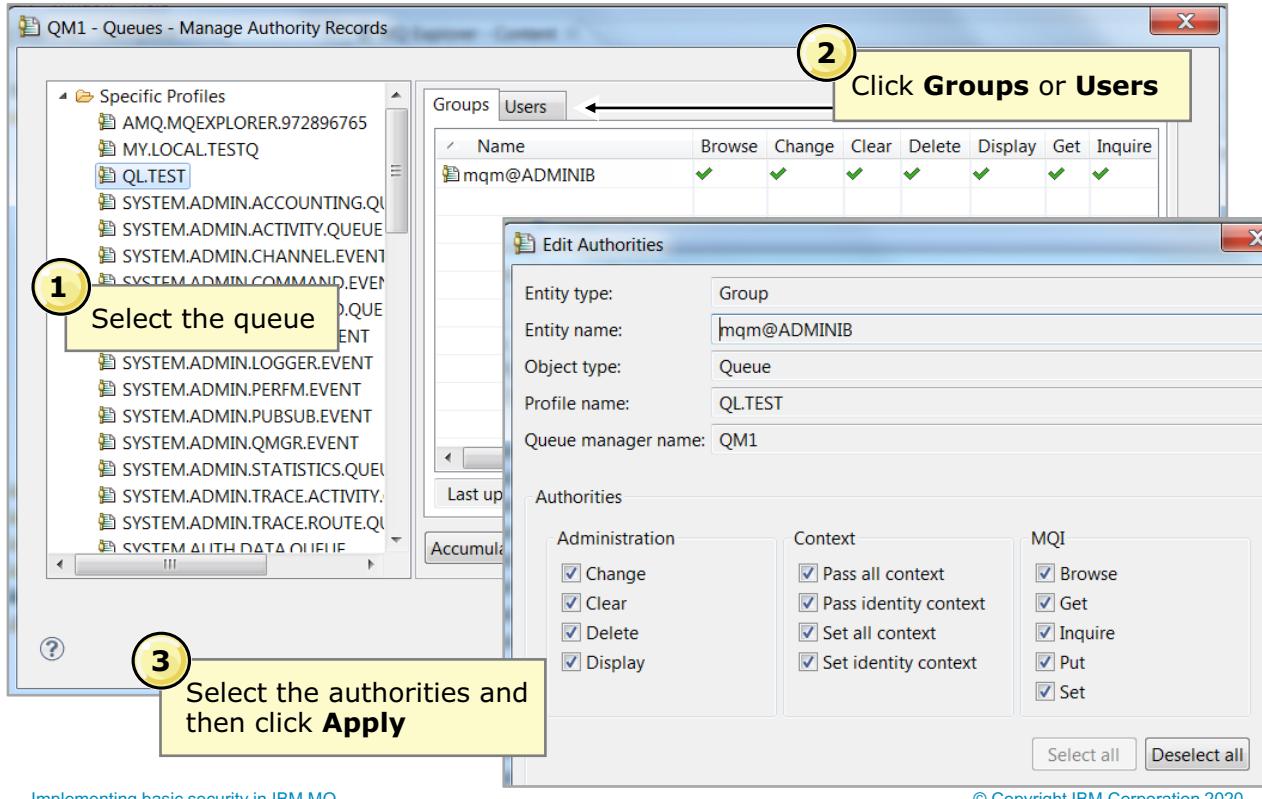
1. Click the **Groups** tab to modify a group record, or click the **Users** tab to modify the record for a specific user.
2. Select the group or user and then click **Edit**.
3. Modify the authorities and then click **OK**.

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 2020

Figure 8-16. Queue authorization in IBM MQ Explorer

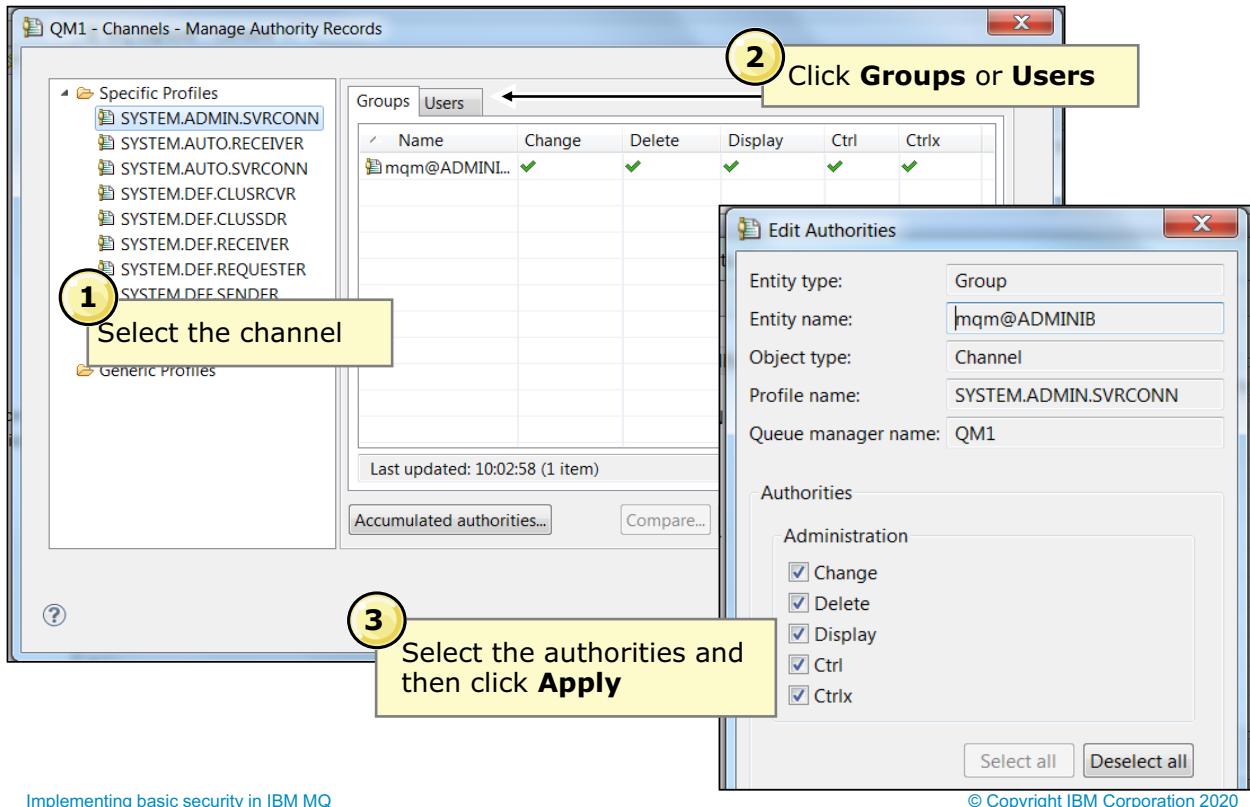
You can modify or add queue authority records in MQ Explorer, by right-clicking the queue in the **Queue** content view and then clicking **Object Authorities > Manage Authority Records**.

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 2020

Figure 8-17. 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

Figure 8-18. 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 8-19. 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`

[Implementing basic security in IBM MQ](#)

© Copyright IBM Corporation 2020

Figure 8-20. 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 QMCO1 -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, reason code `MQRC_NOT_AUTHORIZED` is returned

[Implementing basic security in IBM MQ](#)

© Copyright IBM Corporation 2020

Figure 8-21. 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 8-22. 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 8-23. 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

Figure 8-24. 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 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

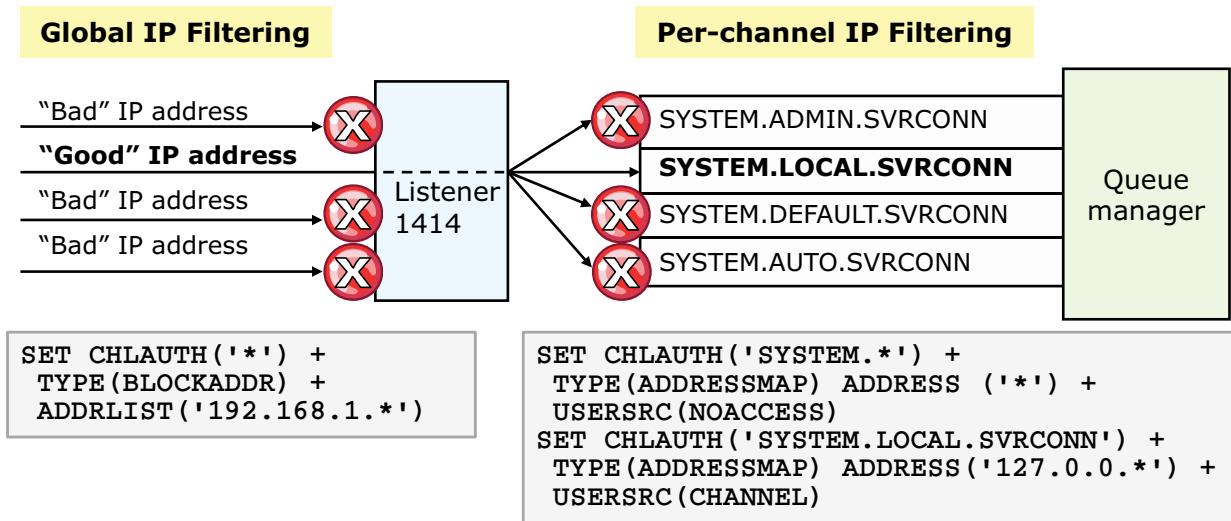
Figure 8-25. Channel authentication commands

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 2020

Figure 8-26. Channel authentication example

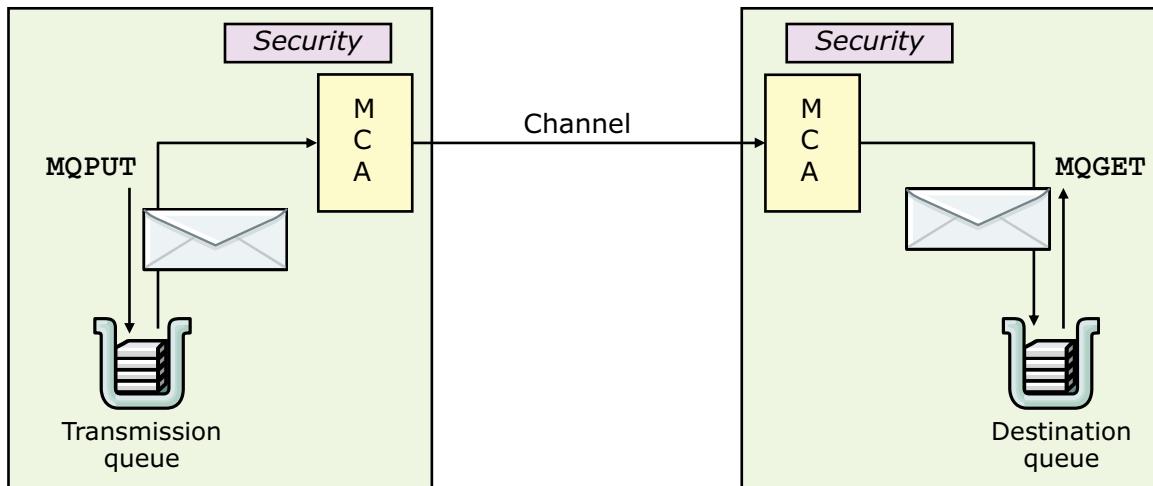
This figure provides an example of channel authentication.

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 2020

Figure 8-27. 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.

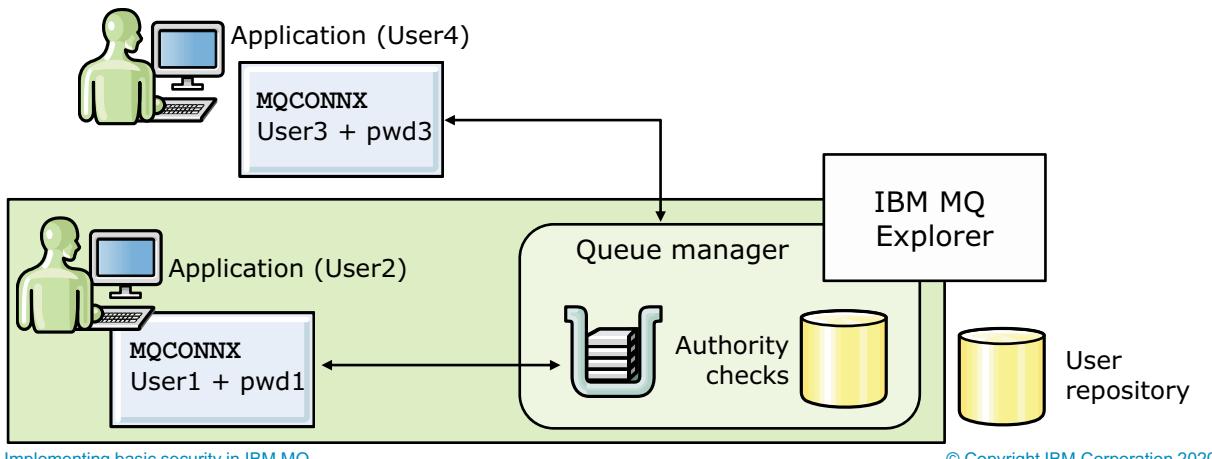


Note

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 2020

Figure 8-28. 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

Implementing basic security in IBM MQ

© Copyright IBM Corporation 2020

Figure 8-29. 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 8-30. 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.

Unit summary

- Describe how the object authority manager (OAM) provides security for IBM MQ resources
- Protect IBM MQ resources by using the OAM
- Implement basic channel authentication

Implementing basic security in IBM MQ

© Copyright IBM Corporation 2020

Figure 8-31. 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 allows 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 2020

Figure 8-32. Review questions

Write your answers here:

- 1.
- 2.
- 3.

Checkpoint 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 allows 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 2020

Figure 8-33. Checkpoint answers

Exercise: Controlling access to IBM MQ

Implementing basic security in IBM MQ

© Copyright IBM Corporation 2020

Figure 8-34. Exercise: Controlling access to IBM MQ

Exercise introduction

- Define and display access control on a queue
- Manage authority records
- Enable and monitor authority events
- Test security



Implementing basic security in IBM MQ

© Copyright IBM Corporation 2020

Figure 8-35. Exercise introduction

Unit 9. Securing IBM MQ channels with TLS

Estimated time

01:30

Overview

In this unit, you learn how to use Transport Layer Security (TLS) to secure IBM MQ channel communications that include mutual authentication.

How you will check your progress

- Review questions
- Lab exercise

Unit objectives

- Describe the certificate infrastructure that is supported in IBM MQ
- Manage certificates with IBM Key Management
- Describe cipher specifications and their support in IBM MQ
- Use certificate revocation lists or Online Certificate Status Protocol (OCSP) to validate currency of certificates
- Use TLS to secure IBM MQ channel communications

Topics

- TLS overview
- Implementing TLS in IBM MQ

9.1. TLS overview

TLS overview

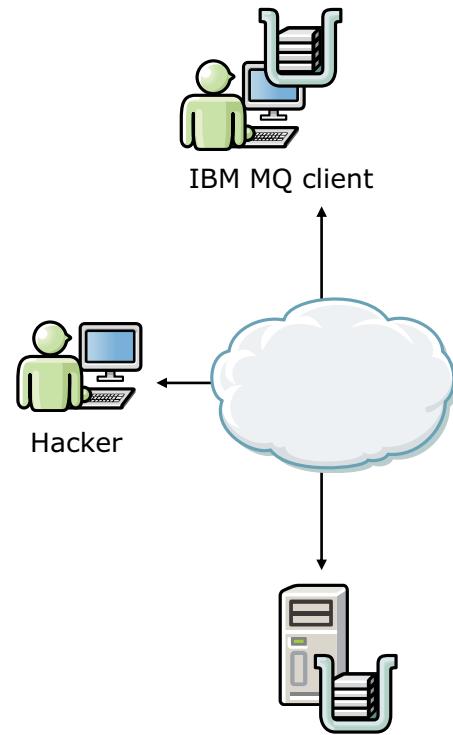
Securing IBM MQ channels with TLS

© Copyright IBM Corporation 2020

Figure 9-3. TLS overview

The security problems

- **Eavesdropping and message privacy**
How can you stop someone from seeing the information that you send?
- **Tampering and message integrity**
How can you stop someone from changing the information that you send?
- **Impersonation and authentication**
How do you know that a person is who they say they are?
- **Replay**
An extension of eavesdropping



Securing IBM MQ channels with TLS

© Copyright IBM Corporation 2020

Figure 9-4. The security problems

Whenever data is being transmitted over a network, many security problems can occur.

- The data can be captured and read, and a hacker can use the information. The problem is message privacy.
- The data can be damaged or changed, either accidentally or intentionally, before it arrives at its destination. The problem is message integrity.
- The communicating parties normally want to be sure about the identity of the partner. The sender of a message wants to know who receives the data, and the receiver wants to be sure of the identity of the originator. The problem is user authentication.

This unit looks at the different ways you can encrypt your message and the problems that are associated with security.

Data encryption and signing

- Encrypt data with the public key of the owner
 - Only the owner can decrypt it
- Encrypt with a private key
 - If they have the public key, anyone can decrypt data
 - Identifies message source; it is a signed message
- Encrypt with a private key and a public key
 - Only you can decrypt it
 - Identifies message source; it is a signed message
- Hash the message and encrypt the hash (change detection)
 - Message digest or Message Authentication Code (MAC)
 - Easy to compute
 - Difficult to reverse
- *Key ring or keystore*: a repository for keys
 - Keep private key private

Figure 9-5. Data encryption and signing

To deal with the problem of eavesdropping, encrypt the information before you send it so that an eavesdropper cannot read the information.

Various combinations of encryption that use both public and private keys are shown here. The implications of public and private keys are also described.

Message digests are fixed-size numeric representations of the contents of messages, which are inherently variable in size. A *hash function* computes a message digest, which is a transformation that meets two criteria:

1. The hash function must be one way. It must not be possible to reverse the function to find the message corresponding to a message digest, other than by testing all possible messages.
2. It must be computationally infeasible to find two messages that hash to the same digest. A message digest is also known as a *message authentication code* because it can provide assurance that the message was not modified. The message digest is sent with the message itself. The receiver can generate a digest for the message and compare it with the sender digest. If the two digests are the same, the message integrity is verified. Any tampering with the message during transmission almost always results in a different message digest.

Elementary encryption: Symmetric key

- Monoalphabetic cipher: Change A to D, B to Z, and C to Q
 - With enough data, it can be decrypted ABBA = DZZD
 - Low-cost encryption
 - If they know the key, anyone can read data

- Polyalphabetic cipher
 - Multiple schemes
 - Harder to break
 - Low-cost encryption
 - If they know the keys and schemas, anyone can read data

Figure 9-6. Elementary encryption: Symmetric key

Cryptography is the process of converting between readable text, called *plaintext*, and an unreadable form, called *ciphertext*.

1. The sender converts the plaintext message to ciphertext. This part of the process is called *encryption*.
2. The ciphertext is transmitted to the receiver.
3. The receiver converts the ciphertext message back to its plaintext form. This part of the process is called *decryption*.

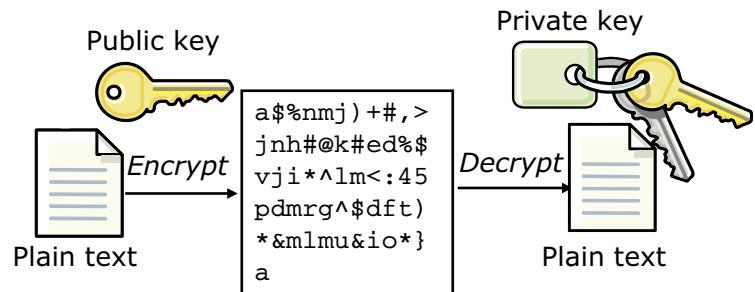
The conversion involves a sequence of mathematical operations that change the appearance of the message during transmission but do not affect the content. Cryptographic techniques can ensure confidentiality and protect messages against unauthorized viewing because an encrypted message is not understandable.

Cryptographic techniques involve a general algorithm, which is made specific by the use of keys. Cryptographic techniques that require both parties to use the same secret key are known as symmetric algorithms.

Monoalphabetic cipher is also known as a cryptogram, which uses a key that consists of rearranging the letters of the alphabet. Polyalphabetic substitution uses several substitution alphabets instead of just one.

Elementary encryption: Asymmetric keys

- Uses key pairs
 - Encryption key is public
 - Decryption key is private



- Each person has a unique key
- Based on large prime numbers
- Standard key sizes:

▪ 512 bits	Low-strength key
▪ 768 bits	Medium-strength key
▪ 1024 - 4096 bits	High-strength key
- Asymmetric keys cannot decrypt even if you know both the public key and algorithm
- Asymmetric algorithms are much slower than symmetric ones

Figure 9-7. Elementary encryption: Asymmetric keys

Asymmetric keys can be used to implement digital signing, as anything encrypted with the private key of the owner can be used to show that it came from them, if the private key is secure.

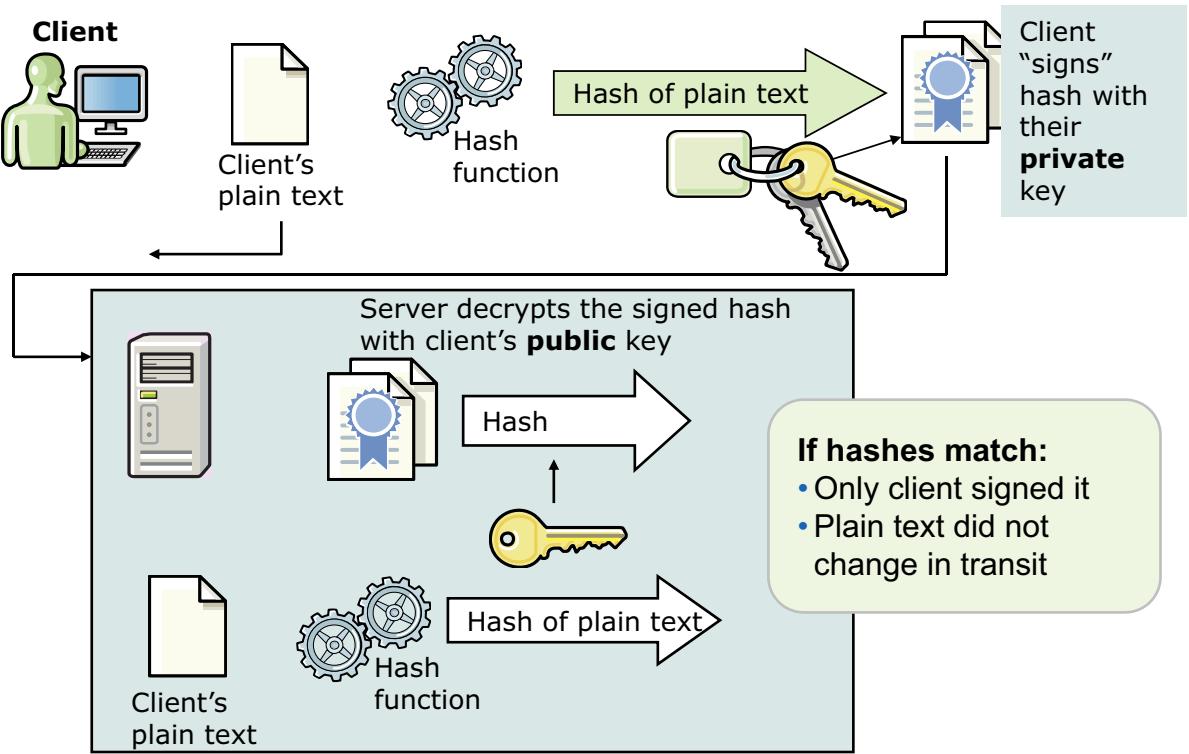
These two keys are mathematically related and they form a key pair. One of these two keys is kept private and the other can be made public.

A private key is typically used for encrypting the message-digest. In such an application, private-key algorithm is called *message-digest encryption algorithm*. A public key is typically used for encrypting the secret-key with a private-key algorithm called a *key encryption algorithm*.

Key sizes determine the strength of level of security. For example, for ordinary use a key size of 768 bits might be acceptable, but for corporate use a key size of 1024 - 4096 bits provides a higher level of encryption.

IBM Training

Digital signature



Securing IBM MQ channels with TLS

© Copyright IBM Corporation 2020

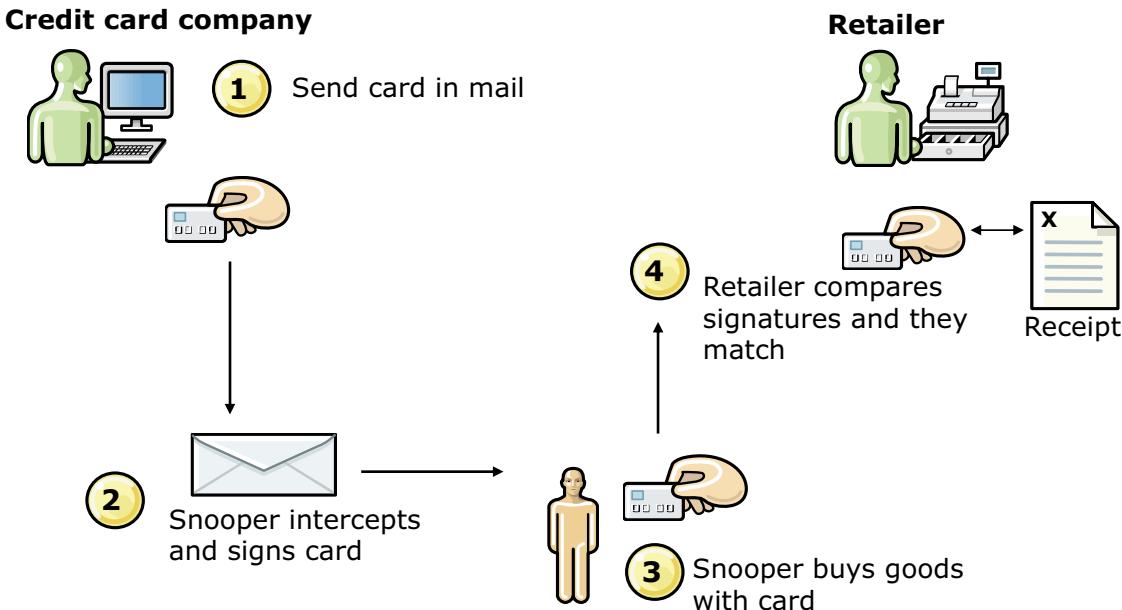
Figure 9-8. Digital signature

Digital signatures combine the use of the one-way hash function and public/private key encryption.

The message is hashed to provide a small message digest or hash. For demonstration purposes, think of it as a unique number that is generated from the plaintext message.

The client private key encrypts this hash or number to create the digital signature. The recipient of the message can also hash the received plaintext message to get a hash number. It can use the client public key to decrypt the digital signature to get the original hash number. If these numbers match, then the message came from the client and it did not change since it was signed.

Relative authentication



Securing IBM MQ channels with TLS

© Copyright IBM Corporation 2020

Figure 9-9. Relative authentication

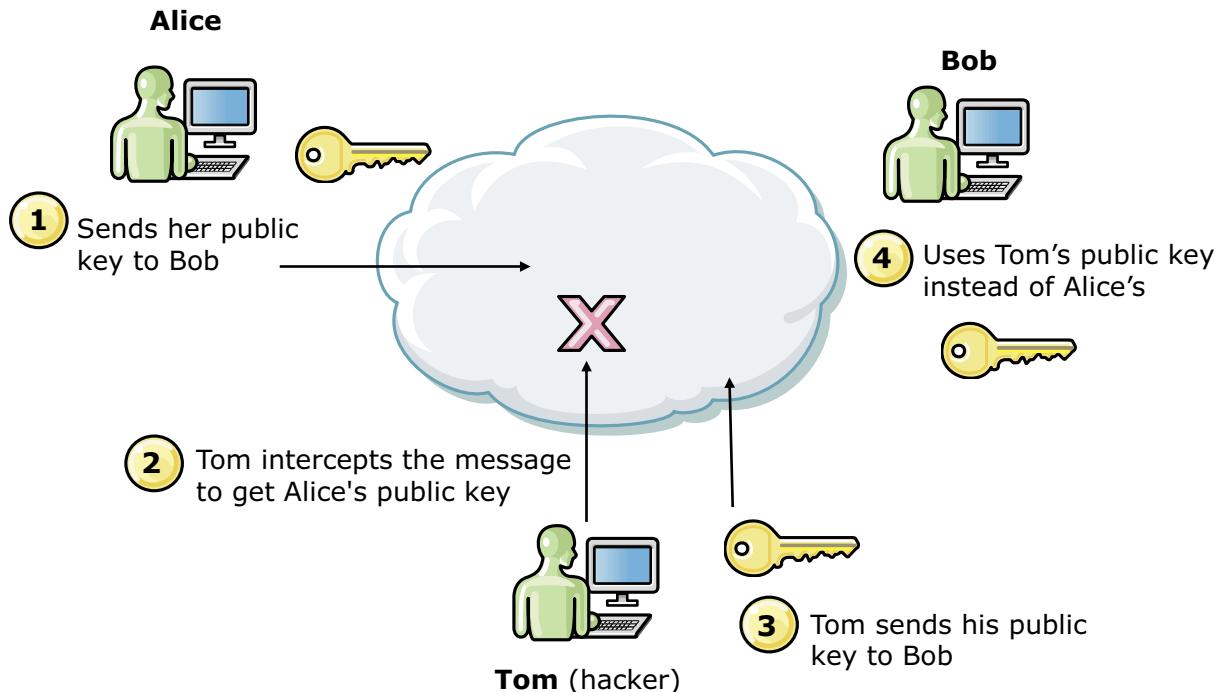
The figure shows examples of relative authentication and the problems that are associated with it.

1. When a credit card company sends a new credit card, they send it in the mail to your address.
2. If someone else picks up the mail before you, they can take your new credit card and sign your name (which is printed on the front of the card) in their own handwriting.
3. The identify thief can then use the credit card to purchase goods.
4. A retailer that checks the signature on the receipt would see that they matched, but this step verifies only that you are the person who signed the card.

The signature on the back of your credit card is an example of *relative authentication*. All the retailer can check is if the signature on the back of the card is the same as the signature on the receipt. This process does not verify that you are the person that is named on the credit card. Another authentication method is necessary to verify that you are the person that is named on the credit card. Other authentication methods include checking a driver's license or requiring a personal identification number (PIN).

Sometimes relative authentication is all that is required but it might not provide enough security.

The key distribution problem



Securing IBM MQ channels with TLS

© Copyright IBM Corporation 2020

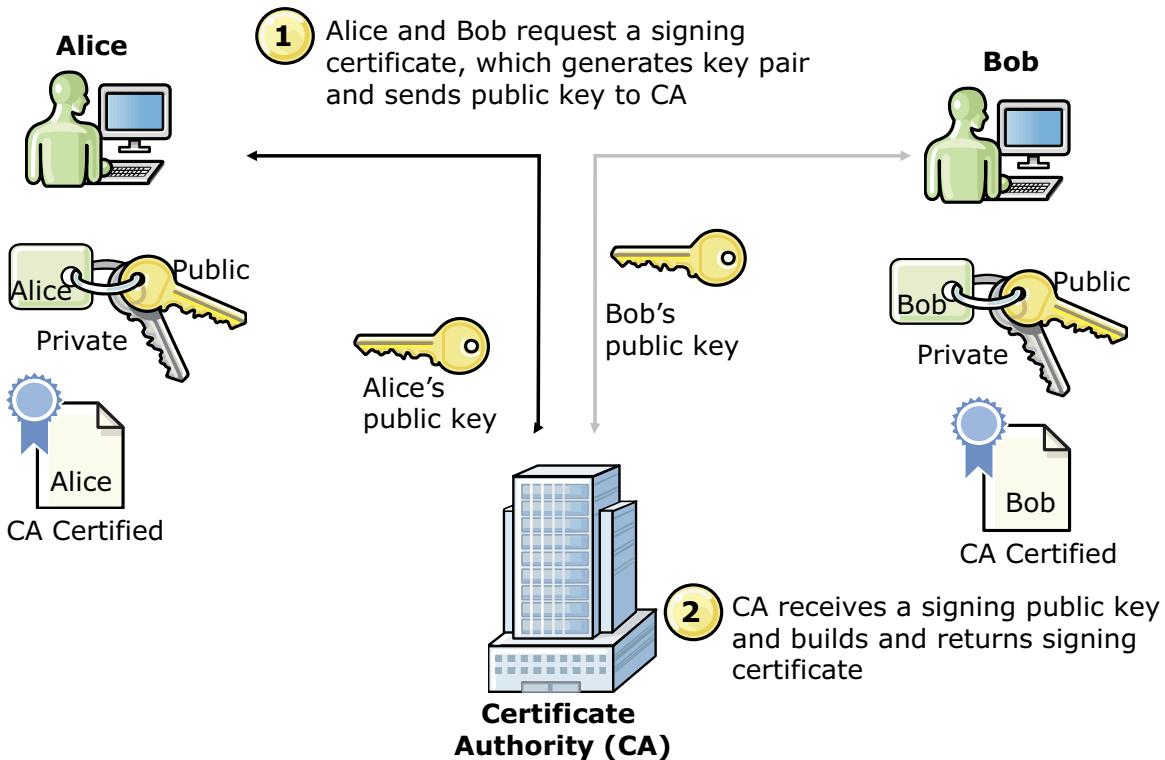
Figure 9-10. The key distribution problem

The initial transfer of public keys can be subject to a *man-in-the-middle* attack.

For example, Alice wants to send her public key to Bob so that he can use it to encrypt transmissions back to her. However, a hacker, Tom, intercepts the message from Alice and replaces Alice's public key with his own before he sends it to Bob. When Bob receives the key and encrypts his message with it, Tom can intercept the message and decrypt it with his private key to read their messages.

All Bob can prove is that the person who owns the private key that is paired with the public key he was sent is the same person that writes the message that Bob gets. This scenario is another example of relative authentication. Bob cannot authenticate that the public key belongs to Alice.

Digital certificates



Securing IBM MQ channels with TLS

© Copyright IBM Corporation 2020

Figure 9-11. Digital certificates

A digital certificate contains information about the individual, for example, their name, company, and public key. The certificate is signed with a digital signature by a certificate authority (CA), which is a trustworthy authority.

To resolve the problem that is described in the previous example, Alice and Bob would request a signing certificate from a CA. Instead of sending each other their public keys, they would send them to the CA. The CA would verify the identity of the sender, and then create the certificates for Alice and Bob.

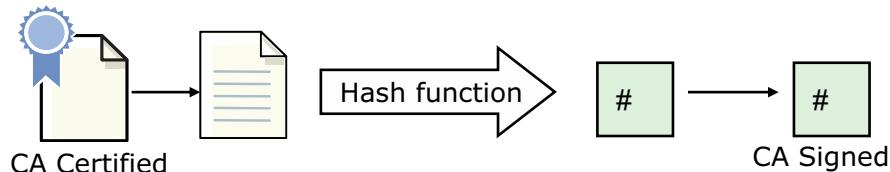
A CA is an independent and trusted third party that provides digital certificates. Digital certificates provide you with an assurance that the public key of an entity truly belongs to that entity.

A CA has the following roles:

- On receiving a request for a digital certificate to verify the identity of the requester before building, signing, and returning the personal certificate.
- To provide the public key of the CA in its CA certificate.
- To publish lists of certificates that are no longer trusted in a certification revocation list (CRL).

Trusting a digital certificate

- Digital certificate is plain text
- Can be subject to tampering
- CA signs at creation



Digital signature of the CA allows tampering to be detected:

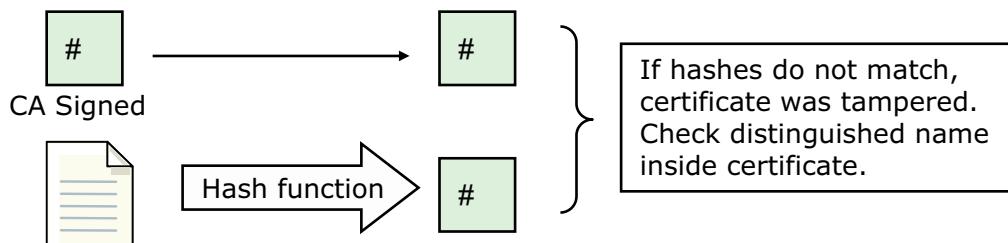


Figure 9-12. Trusting a digital certificate

If a key can be intercepted, can a certificate also be intercepted? How can you trust the certificate that was sent to you? Can someone tamper with a certificate to pretend to be someone they are not?

A digital certificate can be thought of as a piece of plaintext that can be subject to tampering. It is a file on your computer.

The “Digital certificate” example on the previous page showed Alice signing the plaintext document before she sent it to Bob. Bob can then check the signature to ensure that the message from Alice was not altered. The same technique is used to determine whether anyone tampered with a digital certificate.

The CA calculates the hash value of the plaintext (your certificate) and then signs that hash value with the CA private key to generate a CA digital signature. To check that the certificate is valid, the CA digital signature can be decrypted by using the CA public key to check that the hash values match. Well-known CA public keys are installed in many of the security products that use SSL/TLS for security.

Digital certificates do not contain your private key. You must keep your private key secret.

Certificate revocation

- What happens if a certificate is no longer trusted?



Valid from Jan 1, 2019
Valid to Jan 1, 2020

- Certificate revocation list (CRL) holds a list of certificates that CA marks as no longer trusted
- Authority revocation list (ARL) is a form of CRL that contains certificates that are issued to CA

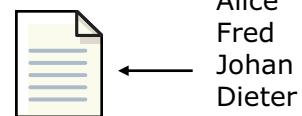


Figure 9-13. Certificate revocation

A digital certificate has two dates that are associated with it. It has a date from which it is valid and it has a date after which it is invalid, or an expiration date.

What happens when a certificate was sent and it is no longer considered to be trusted before its expiration date passed? A certificate authority can revoke a certificate that is no longer trusted by publishing it in a certificate revocation list (CRL). When a certificate is received, it can be checked against this list to ensure that it was not revoked.

An authority revocation list (ARL) is a form of CRL that contains certificates that are sent to certifying authorities; contrary to CRL, which contains revoked end-entity certificates.

Digital certificate details

- X.509 standard
- Digital certificates contain at least the following information about the entity that is certified:
 - Public key of the owner
 - Distinguished name of the owner
 - Distinguished name of the CA that is sending the certificate
 - Date from which the certificate is valid
 - Expiration date of the certificate
 - Version number
 - Serial number
- X.509 V2 certificates also contain:
 - Issuer identifier
 - Subject identifier
- X.509 V3 certificates can contain even more information

Figure 9-14. Digital certificate details

Digital certificates that IBM MQ uses comply with the X.509 standard. The X.509 standard determines the specific pieces of information the digital certificates contain and the format for sending digital certificates.

Digital certificates contain at least the following information about the entity that is certified:

- The public key of the owner
- The distinguished name of the owner
- The distinguished name of the CA that is providing the certificate
- The date from which the certificate is valid
- The expiration date of the certificate
- A version number
- A serial number

An X.509 V2 certificate also contains an issuer identifier and a subject identifier.

An X.509 V3 certificate can contain some other extensions such as the basic constraint extension. Some extensions are standard, but others are implementation-specific. An extension can be critical, in which case a system must be able to recognize the field; otherwise, the system must reject the certificate. If an extension is not critical, the system can ignore the extension.

Distinguished name

- Uniquely defines a user or entity
- X.509 format is well defined

Example:

```
CN="Thomas J Watson" L="Yorktown Heights" ST=NY O=IBM
OU="IBM Headquarters" C=US
```

CN	Common name
T	Title
L	Locality name
ST/SP/S	State or province name
O	Organization name
OU	Organizational unit name
C	Country

Figure 9-15. Distinguished name

The distinguished name (DN) uniquely identifies an entity in an X.509 certificate. The format for the DN is described in the figure. The X.509 standard provides for a DN that is specified in a string format.

In the DN, the common name (CN) can describe an individual user or any other entity, for example, a web server.

The DN can contain multiple organization unit (OU) attributes, but only one instance of each of the other attributes is allowed. The order of the OU entries is significant. The order specifies a hierarchy of organizational unit names, with the highest-level unit first.

The X.509 standard defines other attributes that do not usually form part of the DN but can provide optional extensions to the digital certificate.

Transport Layer Security (TLS) concepts

- Provides privacy and data integrity for data that is exchanged over a network
- Composed of two layers
 - TLS Record Protocol layer provides connection security
 - TLS Handshake Protocol layer supports client and server authentication and encryption algorithm and cryptographic key negotiation before any data is exchanged
- Used whenever a cipher specification is prefaced with “TLS” or “ECDHC” is specified

Securing IBM MQ channels with TLS

© Copyright IBM Corporation 2020

Figure 9-16. Transport Layer Security (TLS) concepts

The TLS protocol allows two parties to communicate with privacy and data integrity. Although they are similar, TLS and SSL are not interoperable.

The TLS protocol provides communications security over the internet, and allows client/server applications to communicate in a way that is private and reliable. The protocol has two layers:

- TLS record protocol
- TLS handshake protocol

The TLS layer requires a transport protocol such as TCP/IP.

For more information about the TLS protocol, see the information that the TLS Working Group provides on the website of the Internet Engineering Task Force at <http://www.ietf.org>.

9.2. Implementing TLS in IBM MQ

Implementing TLS in IBM MQ

Securing IBM MQ channels with TLS

© Copyright IBM Corporation 2020

Figure 9-17. Implementing TLS in IBM MQ

TLS support in IBM MQ

- Authentication
 - Queue managers or clients that initiate an TLS-enabled connection are assured of the identity of the queue manager that they are connecting to
 - Queue managers that are receiving connections can check the identity of the queue manager or client that initiates the connection
- Message privacy
 - If configured, TLS uses a unique session key that encrypts all information that is exchanged over the connection to ensure that unauthorized parties cannot view information
- Message integrity
 - Data cannot be tampered with over the connection
- Certificate authority chain
 - Each certificate in the CA chain is signed by the entity that its parent certificate in the chain identifies
 - Root CA always signs root CA certificate at the head of the chain
 - Signatures of all certificates in the chain must be verified

Figure 9-18. TLS support in IBM MQ

IBM MQ supports some cipher specifications that use the TLS V1.2 protocols.

An optional feature on the TLS handshake is the authentication of the client certificate and the server certificate.

SSL can allow for the client to pass many cipher specifications on the SSL handshake and the server can choose one that it supports. IBM MQ imposes the restriction that only one cipher specification can be supplied on the channel definition, which must match at both ends.

SSL can allow its sessions to be reused. Session reuse might be useful for short-lived web queries. However, IBM MQ channels are likely to be longer-running, so the session reuse feature of SSL is not used.

TLS is the same as SSL in terms of IBM MQ resource definitions. The cipher specification is the only setting for TLS.

Cipher specification support in IBM MQ

IBM MQ V9 on Windows, Linux, and UNIX supports the following TLS cipher specifications:

- TLS 1.0
 - TLS_RSA_WITH_AES_128_CBC_SHA (TLS 1.0)
 - TLS_RSA_WITH_AES_256_CBC_SHA (TLS 1.0)
- TLS 1.2
 - ECDHE_ECDSA_AES_128_CBC_SHA256
 - ECDHE_ECDSA_AES_256_CBC_SHA384
 - ECDHE_ECDSA_AES_128_GCM_SHA256
 - ECDHE_ECDSA_AES_256_GCM_SHA384
 - ECDHE_RSA_AES_128_CBC_SHA256
 - ECDHE_RSA_AES_256_CBC_SHA384
 - ECDHE_RSA_AES_128_GCM_SHA256
 - ECDHE_RSA_AES_256_GCM_SHA384
 - TLS_RSA_WITH_AES_128_CBC_SHA256
 - TLS_RSA_WITH_AES_256_CBC_SHA256
 - TLS_RSA_WITH_AES_128_GCM_SHA256
 - TLS_RSA_WITH_AES_256_GCM_SHA384

[Securing IBM MQ channels with TLS](#)

© Copyright IBM Corporation 2020

Figure 9-19. Cipher specification support in IBM MQ

Not all of the TLS cipher specifications are available for specification on IBM MQ channels.

From IBM MQ Version 8.0.2, SSLv3 protocol and cipher specifications are deprecated.

From IBM MQ Version 8.0.3, some TLS1.0 and TLS1.2 cipher specifications are deprecated.

By default, you are not allowed to specify a deprecated cipher specification on a channel definition.

If you attempt to specify a deprecated cipher specification, you receive message AMQ8242: *SSL/CIPH definition wrong*, and PCF returns MQRCCF_SSL_CIPHER_SPEC_ERROR.

You cannot start a channel with a deprecated cipher specification. If you attempt to start a channel with a deprecated cipher specification, the system returns *MQCC_FAILED* (2) with a reason of *MQRC_SSL_INITIALIZATION_ERROR* (2393) to the client.

It is possible for you to re-enable one or more of the deprecated cipher specifications for defining channels at run time on the server by setting the environment variable `AMQ_SSL_WEAK_CIPHER_ENABLE`.

IBM MQ connection procedure with TLS

- When a queue manager connects to another queue manager
 - Queue managers exchange and validate certificates
 - Must configure both queue managers and the channels with appropriate certificate settings
- When messages are sent from one queue manager to another queue manager along a channel
 - Data is generally encrypted by using a session key that is established during the certificate exchange
 - Must configure the channels with appropriate cipher specifications

Securing IBM MQ channels with TLS

© Copyright IBM Corporation 2020

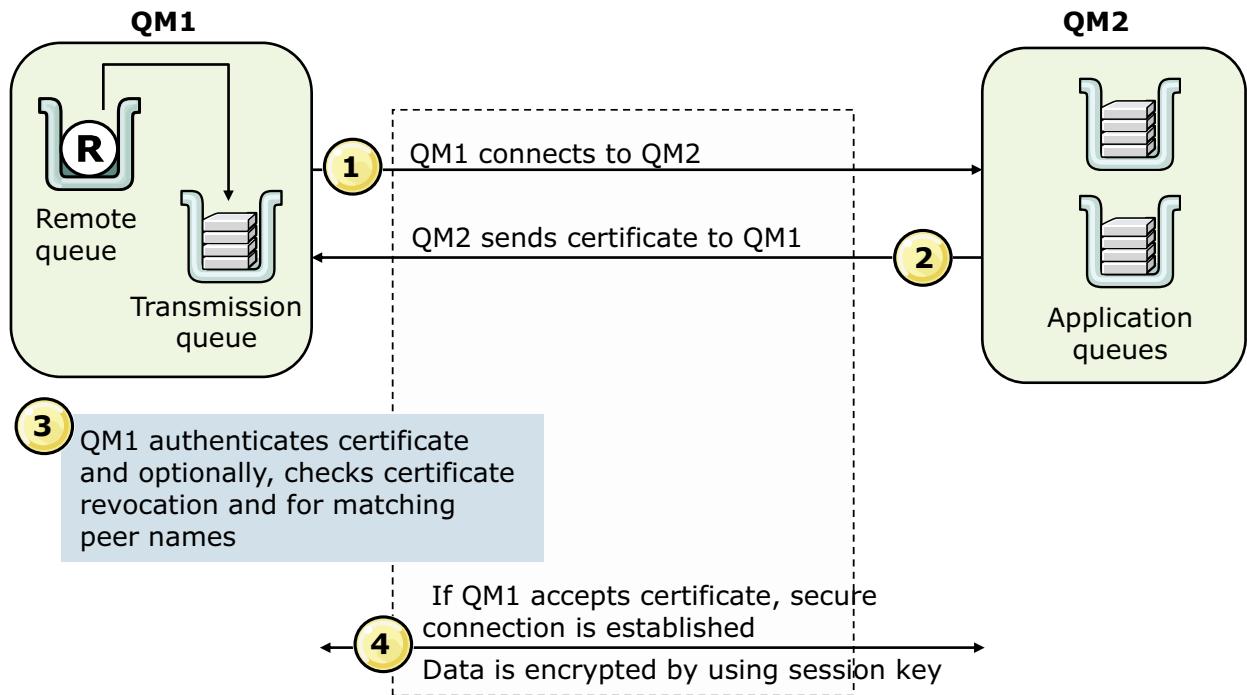
Figure 9-20. IBM MQ connection procedure with TLS

You can implement mutual authentication between two queue managers by using self-signed TLS certificates.

When messages are sent on a message channel, the data is generally encrypted by using a session key that is established during the queue manager certificate exchange. Message encryption assumes that the channels are configured with appropriate cipher specifications.



TLS and IBM MQ certificate exchange



Securing IBM MQ channels with TLS

© Copyright IBM Corporation 2020

Figure 9-21. TLS and IBM MQ certificate exchange

This figure summarizes the queue manager certificate exchange process.

1. QM1 connects to QM2.
2. The personal certificate that is used by QM2 is sent to QM1.
3. QM1 authenticates the personal certificate against the chain of certificate authority certificates. QM1 optionally checks for certificate revocation if Online Certificate Status Protocol (OCSP) is supported on the server. OCSP is an Internet protocol that is used for obtaining the revocation status of an X.509 digital certificate.
4. QM1 optionally checks the personal certificate against the CRL.
5. QM1 optionally applies a filter to accept personal certificates that meet any defined peer names only.
6. If QM1 accepts the personal certificate from QM2, the secure connection is established.

Configuring TLS on queue managers

1. Create the queue manager key repository
 - Store certificates that the queue manager uses
 - Use IBM Key Management

2. Specify the queue manager key repository location by using one of these options:
 - In IBM MQ Explorer queue manager properties
 - By using the `ALTER QMGR SSLKEYR` command
 - By specifying the **Key Repository** queue manager attribute

3. Use CRL or OCSP to ensure that the certificate is valid

4. Configure for cryptographic hardware (if support is required)

[Securing IBM MQ channels with TLS](#)

© Copyright IBM Corporation 2020

Figure 9-22. Configuring TLS on queue managers

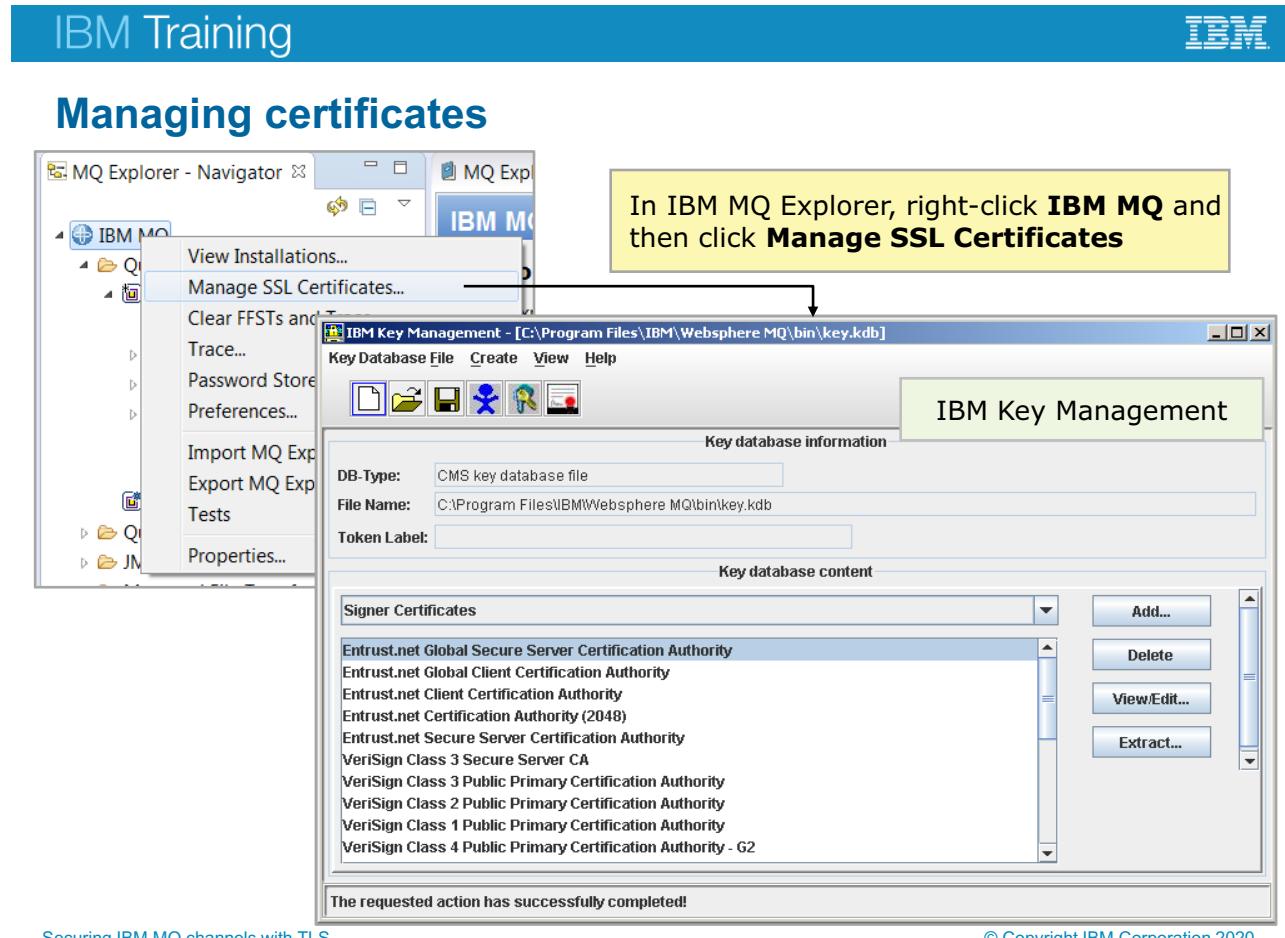
This figure lists the configuration tasks that are required for setting up IBM MQ to use TLS.

The first step is to create the queue manager key repository by using the IBM Key Management application or commands. The key repository is a file that stores the certificates that the queue manager uses. On Windows, Linux, and UNIX, the key repository is known as the *key database* file.

After you create the key repository, you must configure the queue manager with the location of the key repository.

CAs can revoke certificates that are no longer trusted by publishing them in a CRL. The next step is to configure CRL checking, if CRL checking is required. When a queue manager receives a certificate, it can be checked against the CRL on an LDAP server or by using OCSP to ensure that it is not revoked. CRL checking is not mandatory for messaging with TLS, but it can ensure the trustworthiness of user certificates.

IBM MQ can support cryptographic hardware when necessary. The queue manager must be configured to support cryptographic hardware.



Securing IBM MQ channels with TLS

© Copyright IBM Corporation 2020

Figure 9-23. Managing certificates

The IBM Key Management ([iKeyman](#)) application can manage the TLS certificates that IBM MQ provides.

On Linux or Windows, start the IBM Key Management application by running the `strmqikm` command.

On Windows, you also start the IBM Key Management application by clicking **IBM MQ > IBM Key Management** from the Windows **Program** menu.

If you have IBM MQ Explorer installed, you can start the IBM Key Management application by right-clicking **IBM MQ > Manage SSL Certificates**.

Creating certificates

- Certificates contain the distinguished name
1. Create key database file by using IBM Key Management
 2. Specify **Generate a certificate request**
 - This request is written to a file or data set
 - Send it to the CA (can be sent by using their website)
 - Receive your signed certificate from the CA
 3. Import the certificate into the repository and label it

Figure 9-24. Creating certificates

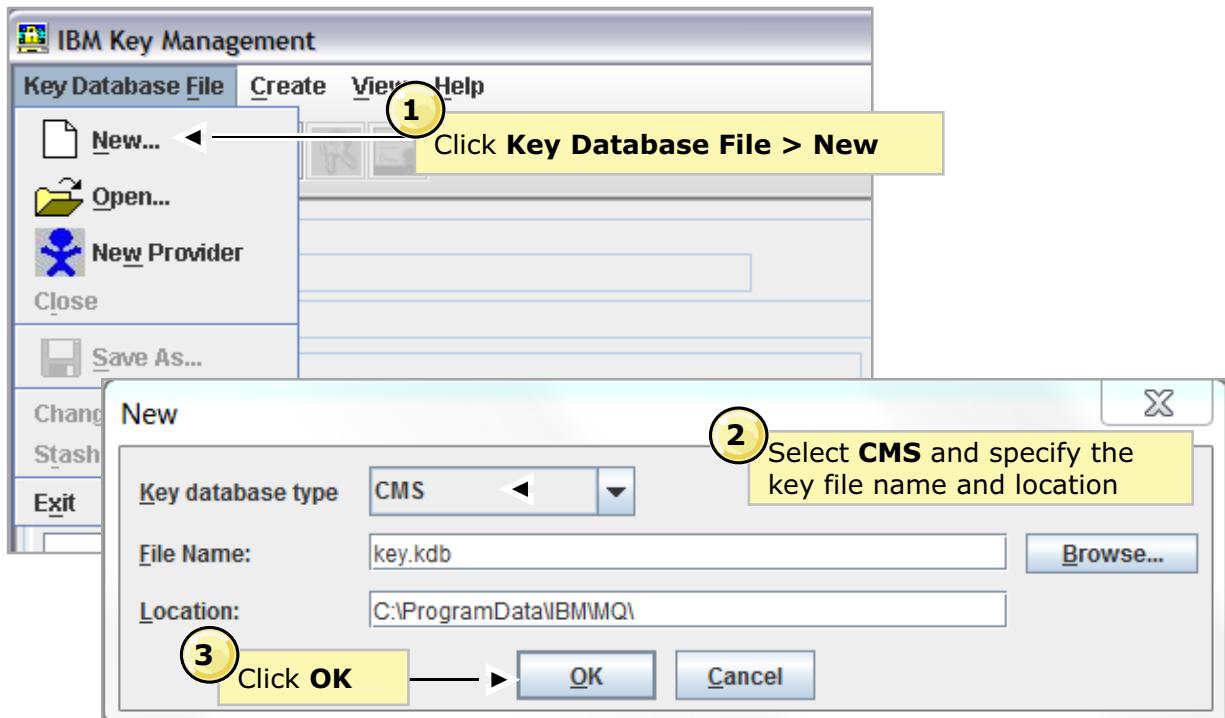
This figure summarizes the steps for creating certificates.

A digital certificate can be created on your system and self-signed or that your site CA signed. These types of certificates might be useful for internal use certificates or for testing purposes. If you want to communicate with an external entity, you must obtain a certificate that a CA signed.

1. A TLS connection requires a key repository at each end of the connection. The queue manager and IBM MQ client must have access to a key repository. Create the key repository, also known as the key database file, by using IBM Key Management.
2. Generate a certificate request, or make a request that is based on an existing certificate in your repository, and send the certificate request to the CA.
3. After you receive a signed certificate, import it into the repository. The certificates are associated with individual queue managers by using a label, or with client logons for client applications.

IBM Training IBM

Creating a key repository with IBM Key Management (1 of 2)



Securing IBM MQ channels with TLS

© Copyright IBM Corporation 2020

Figure 9-25. Creating a key repository with IBM Key Management (1 of 2)

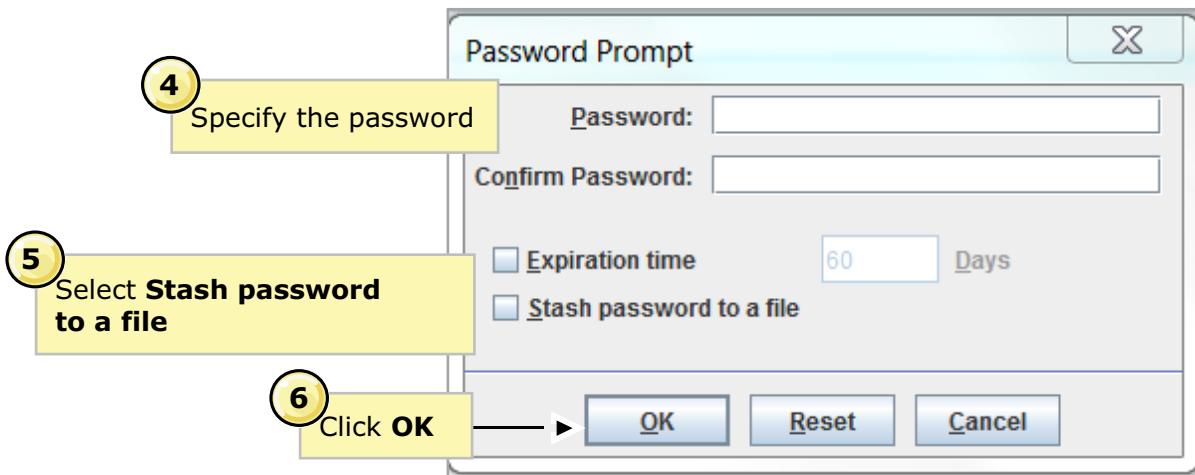
To create a key repository with IBM Key Management:

1. Select **Key Database File > New**.
2. Select **CMS** and specify the key file name and location.

The **File Name** field already contains the text `key.kdb`. If your stem name is `key`, leave this field unchanged. If you specified a different stem name, replace `key` with your stem name but you must not change the `.kdb` extension.

3. Click **OK**.

Creating a key repository with IBM Key Management (2 of 2)



Securing IBM MQ channels with TLS

© Copyright IBM Corporation 2020

Figure 9-26. Creating a key repository with IBM Key Management (2 of 2)

1. Specify a password for the key repository.
2. Select the **Stash the password to a file** option.

If you do not stash the password, attempts to start TLS channels fail because they cannot obtain the password that is required to access the key database file.

3. Click **OK**. The signer certificates window displays, containing a list of the CA certificates that are provided with IBM Key Management and preinstalled in the key database. Set the access permissions.

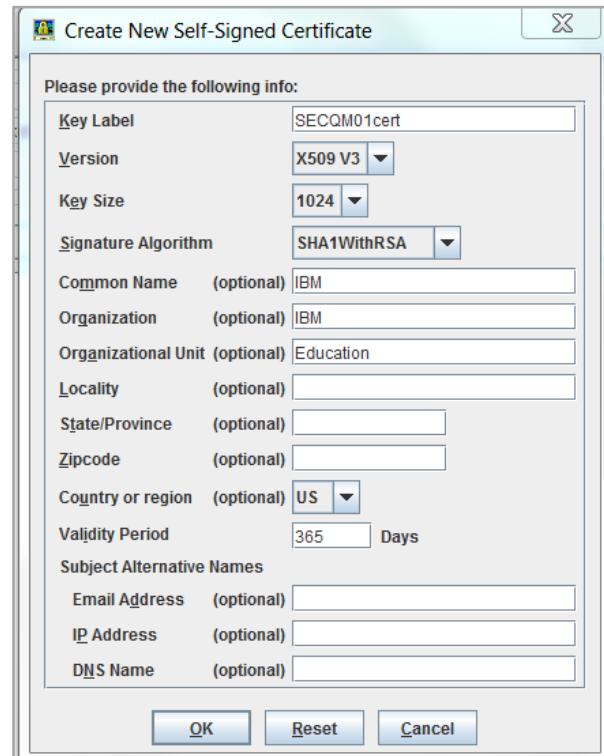
When IBM Key Management is used to create the key database file, the access permissions for the key database file are set to give access only to the user ID that used IBM Key Management.

The MCA accesses the key database file, so ensure that the user ID under which the MCA runs has permission to read both the key database file and the password stash file.

MCAs can run under the “mqm” user ID, which is in the “mqm” group. After you create your queue manager key database file, work with the same user ID to add read permission for the “mqm” group.

Using IBM Key Management to create a self-signed personal certificate

1. Click **Key Database File > Open**
2. Click **Key database type** and select **CMS**
3. Specify the name and location of the key database file in which you want to save the certificate
4. Type the password that you set when you created the key
5. Click **Create > New Self-Signed Certificate**
6. In the **Key Label** field, enter the certificate label
7. Type or select a value for any field in the distinguished name



Securing IBM MQ channels with TLS

© Copyright IBM Corporation 2020

Figure 9-27. Using IBM Key Management to create a self-signed personal certificate

For testing purposes, you can create a self-signed personal certificate. This figure lists the steps for creating a self-signed personal certificate by using IBM Key Management.

The **Key Label** value must be one of the following values:

- The value of the **CERTLABEL** attribute, if it is set on the queue manager or client
- The default **ibmwebspheremq** with the name of the queue manager that is appended, all in lowercase characters
- The default **ibmwebspheremq** with MQI client logon user ID appended, all in lowercase

Using IBM Key Management to add a CA certificate

1. Click **Key Database File > Open**
2. Click **Key database type** and select **CMS**
3. Specify the key database file to which you want to add the certificate
4. Enter the key database password
5. In the **Key database content** field, select **Signer Certificates**
6. Click **Add**
7. Specify the certificate file name and location
8. In the **Enter a Label** window, type the name of the certificate

Figure 9-28. Using IBM Key Management to add a CA certificate

This figure lists the steps to use IBM Key Management to add a CA certificate or the public part of a self-signed certificate to the key repository.

If the certificate that you want to add is in a certificate chain, you must also add all the certificates that are above it in the chain. You must add the certificates in the descending order and start from the root, followed by the CA certificate immediately below it in the chain.

Command options for managing certificates and keys

- **`runmqckm`** and **`runmqakm`** commands
 - Create the type of CMS key database files that IBM MQ requires
 - Create certificate requests
 - Import personal certificates
 - Import CA certificates
 - Manage self-signed certificates
 - Include `-help` to display syntax and options

	<code>runmqckm</code>	<code>runmqakm</code>
Creation of certificates and certificate requests with Elliptic Curve public keys	No	Yes
Stronger encryption of the key repository file	No	Yes
Certified as FIPS 140-2 compliant	No	Yes
Supports JKS and JCEKS key repository file formats	Yes	No

Figure 9-29. Command options for managing certificates and keys

As an option, you can use the `runmqckm` command (Windows and UNIX), and the `runmqakm` command (Windows, UNIX and Linux) to manage keys, certificates, and certificate requests.

You can use these commands to create the type of CMS key database files that IBM MQ requires, create certificate requests, import certificates, and manage self-signed certificates.

The table in the figure identifies the differences between the two commands.

Using runmqakm to create and initialize a key database

- To create and initialize a key database file:

```
runmqakm -keydb -create -populate -db <filename>.kdb  
-pw <password> -stash
```

- **db** is the file name for the new key database
- **pw** is the password to use to protect the key database file
- **populate** populates the key database with some predefined trusted CA certificates (optional)
- **stash** saves the key database password locally in the .sth file so that it does not have to be entered on the command line

Figure 9-30. Using runmqakm to create and initialize a key database

This figure summarizes the syntax for creating and initializing a key database by using the **runmqakm** command.

Using runmqakm to generate a self-signed certificate

- To generate a self-signed certificate and store it in the key database:

```
runmqakm -cert -create -db server.kdb -stashed  
-dn "CN=myserver,OU=mynetwork,O=mycompany,C=mycountry"  
-expire 7300 -label "My self-signed certificate"  
-default_cert yes
```

- dn** specifies the distinguished name to use on the public key certificate
- expire** indicates the number of days the certificate is valid
- label** is a name to use for the self-signed certificate within the key database
- default_cert** makes the new certificate the default (optional)

Figure 9-31. Using runmqakm to generate a self-signed certificate

This figure summarizes the syntax for using the `runmqakm` to generate a self-signed certificate and store it in the key database.

Key repository



- Contains digital certificate of entity
 - For queue manager, default label of `ibmwebspheremq<QmgrName>`
 - For client, with default label of `ibmwebspheremq<logonUserId>`
 - From various certificate authorities
- On queue manager, specify key repository path
 Example: `ALTER QMGR SSLKEYR('/var/mqm/qmgrs/QM1/tls/key')`
- On clients, specify **SSLKeyRepository** entry in **SSL** stanza in **mqclient.ini** file
 Example: `SSL:`
`SSLKeyRepository=C:\key`
- In environment variable, specify MQSSLKEYR
 Example: `export MQSSLKEYR=/var/mqm/ssl/key`

Do not include the **.kdb** extension in the file name as the queue manager appends this extension automatically

Figure 9-32. Key repository

Apart from the certificates for the queue manager and client, the key repository can also contain many signed digital certificates from various CAs. These signed digital certificates allow the key repository to be used to verify certificates that it receives from its partner at the remote end of the connection.

The naming of the certificate is crucial; it allows a queue manager to locate its own certificate. The structure and capitalization of the certificate name must be correct for the queue manager's operating system.

- By default, the certificate name is `ibmwebspheremq<qmgr-name>`, all in lowercase, on both. The label name can be overridden with a custom value.
- On clients, the default label is `ibmwebspheremq` followed by the logon user ID, mapped to lowercase characters. The label name can be overridden with a custom value.

On the queue manager, a digital certificate contains the identity of the owner of that certificate. Each queue manager has its own certificate. On all operating systems, this certificate is stored in a key repository by using your digital certificate management tool, such as IBM Key Management.

The key repository is specified on the queue manager object by using the `ALTER QMGR` command. On distributed operating systems, this name is the path and the stem of the file name for the key database file.

On an IBM MQ client, each user typically has a separate key repository file with access that is restricted to that user. This key repository file is accessed by using the environment variable **MQSSLKEYR**. An application can also specify it on the MQCONNX **KeyRepository** parameter.

Labeling the certificate

- Identifies which personal certificate in the key repository is sent to the remote peer
- For the queue manager, use the **CERTLBL** attribute

Example: `ALTER QMGR SSLKEYR('/var/mqm/qmgrs/QM1/ssl/key') CERTLBL('QM1Certificate')`

- For a client, use the **CertificateLabel** entry of the SSL stanza

Example: `SSL:
SSLKeyRepository=C:\key
CertificateLabel=MyCert`

- For the environment, use the environment variable **MQCERTLBL**

Example: `export MQCERTLBL=MyCert`

Figure 9-33. Labeling the certificate

You can provide a label name for the IBM MQ queue manager or client to use. For the queue manager, you can use **CERTLBL** attribute on the **ALTER QMGR** command.

For clients, the certificate label can be provided in the locations:

- By the application in the MQSCO structure with the **SSLKeyRepository** location
- In the SSL stanza in the `mqclient.ini` file with the **SSLKeyRepository** location
- By using the environment variable **MQCERTLBL**

You do not need to run the **REFRESH SECURITY TYPE(SSL)** command if you change the certificate label. However, you must run a **REFRESH SECURITY TYPE(SSL)** command if you change the certificate label name on the queue manager.

Benefits of labeling the certificate

- Administrator can label certificates to follow company policy
- Simplifies migration to new certificate when current certificate is ready to expire

Example:

Current certificate needs to change to new certificate 'QM1 Cert 2017'.

```
ALTER QMGR CERTLBL('QM1 Cert 2017')
REFRESH SECURITY TYPE(SSL)
```

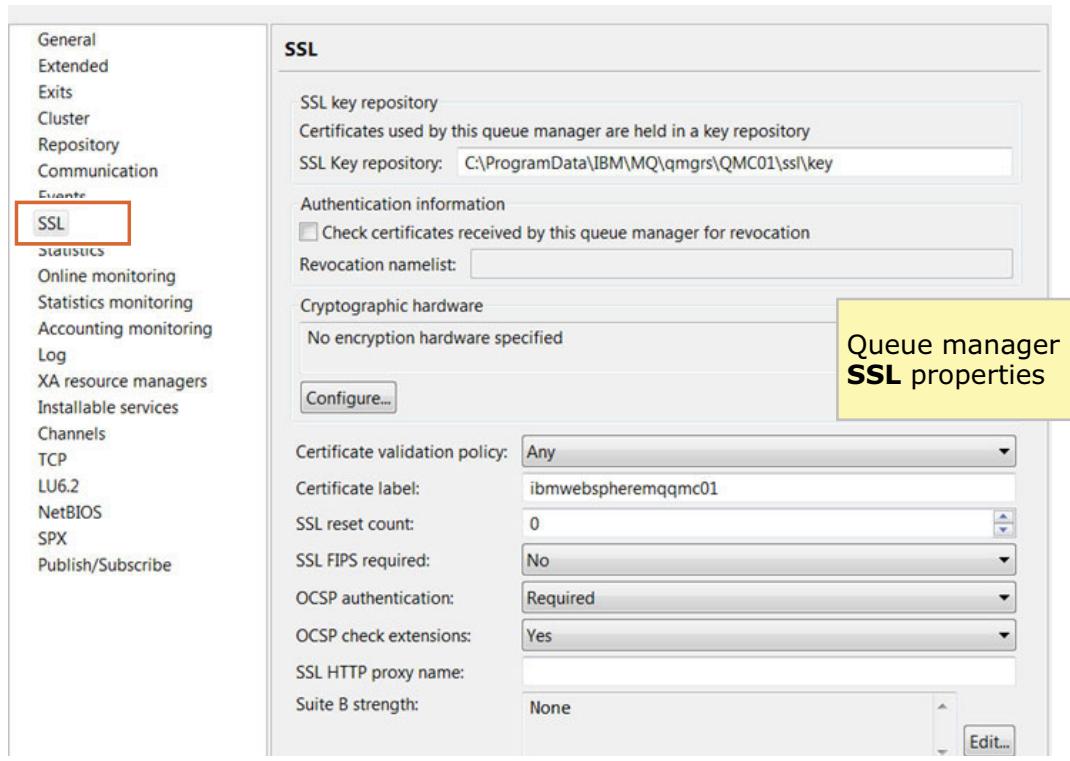
Figure 9-34. Benefits of labeling the certificate

A benefit of labeling a certificate is that you can label your own certificate instead of following the pattern that IBM MQ mandates.

Another benefit of labeling the certificate is that the administrator can change certificates that the channel or client uses. The job of installing the new certificate can be done at any prior point and labeled as you want. That label does not have to change to get the queue manager to use it; the IBM MQ administrator just needs to identify the label to the queue manager and then refresh the queue manager.

IBM Training 

Configuring TLS on queue manager with IBM MQ Explorer



Securing IBM MQ channels with TLS

© Copyright IBM Corporation 2020

Figure 9-35. Configuring TLS on queue manager with IBM MQ Explorer

You can configure TLS properties by using MQSC or IBM MQ Explorer. The configuration properties for TLS are on the **SSL** properties page.

Configuring TLS on channels

- Define the cipher specification by using MQSC or **SSL** channel properties in IBM MQ Explorer
- Option to configure a channel to accept only certificates with attributes in the distinguished name of the owner that match given values
 - Can use the wildcard character (*) at the beginning or the end of the attribute value in place of any number of characters
- Option to configure a queue manager channel so that the queue manager refuses the connection if the initiating party does not send its own personal certificate

Distinguished Name matching support in IBM MQ

Attribute name	Description
SERIALNUMBER	Certificate serial number
MAIL	Email address
UID or USERID	User identifier
CN	Common name
T	Title
OU	Organizational unit name
DC	Domain component
O	Organization name
STREET	Street or first line of address
L	Locality name
ST (or SP or S)	State or province name
PC	Postal code
C	Country
UNSTRUCTUREDNAME	Host name
UNSTRUCTUREDDADDRESS	IP address
DNQ	Distinguished name qualifier

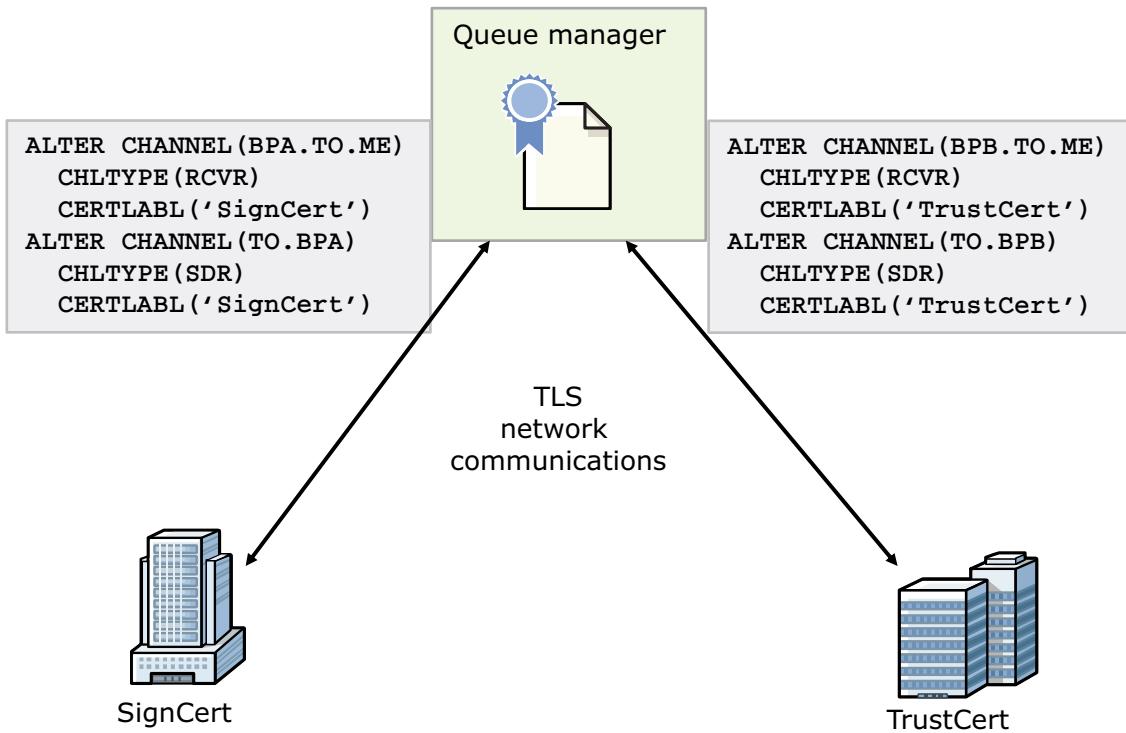
Securing IBM MQ channels with TLS

© Copyright IBM Corporation 2020

Figure 9-37. Distinguished Name matching support in IBM MQ

This table summarizes the DN matching support in IBM MQ.

Defining certificates for channels (1 of 2)



Securing IBM MQ channels with TLS

© Copyright IBM Corporation 2020

Figure 9-38. Defining certificates for channels (1 of 2)

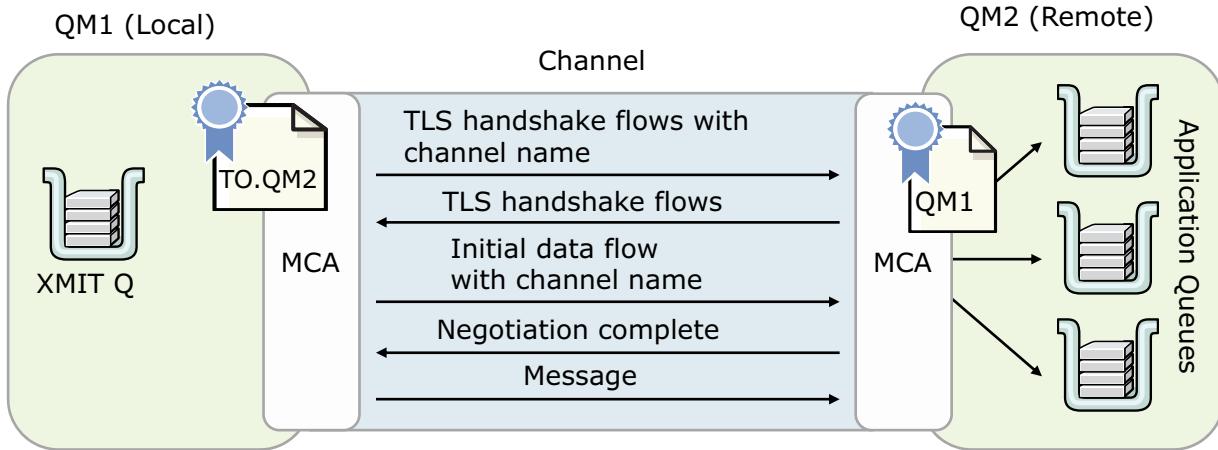
With IBM MQ, you can define certificates for each channel.

The feature provides more flexibility than a single queue manager certificate for connecting to partners with different CA requirements and connecting to servers with different levels of SSL support.

Channel certificates are configured by using the CERTLBL attribute on the channel definition.

- If the CERTLBL attribute is blank, the channel uses the queue manager certificate.
- If the CERTLBL attribute is set to a value, the channel uses the certificate that the label identifies. You can use a CERTLBL for TLS cipher specifications.

Defining certificates for channels (2 of 2)



- **CERTLBL** channel attribute identifies which personal certificate in the key repository is sent to the remote peer
- Both ends of the channel must be same version of IBM MQ

Figure 9-39. Defining certificates for channels (2 of 2)

Enhancements to the TLS protocol allow the provision of information as part of the TLS handshake. This information can be used to determine which certificate to use for this particular connection. This enhancement is known as Server Name Indication (SNI).

IBM MQ uses SNI to provide a channel name in the TLS handshake.

- The sender (or client) end of the channel puts the channel name into the SNI hint for the TLS handshake.
- The receiver (or server-connection) end of the channel retrieves the channel name from the SNI hint and selects the appropriate certificate that is based on that information.

Channel attributes for TLS

- **CERTLBL** identifies certificate label for this channel to use
- **SSLCIPH** specifies cipher specification that is used on the channel
- **SSLPEER** that is used by the local queue manager or client to filter the certificate that the peer queue manager or client at the other end of the channel sends
- **SSLCAUTH** defines whether IBM MQ requires a certificate from the TLS client
 - Can request whether the client end is required to provide a certificate for authentication
 - Valid for receiver, server-connection, cluster-receiver, server, and requester channels
 - For channels with **SSLCIPH** specified

Figure 9-40. Channel attributes for TLS

This figure lists the channel attributes for configuring TLS at the channel level.

The end of the channel that initiates the TLS connection is the client. The client always authenticates the server certificate, but might not necessarily send a certificate to the server to be authenticated. The SSLCAUTH attribute is used on the TLS server-end of the connection to indicate whether to expect a certificate for authentication from the TLS client, or if only the server-end has its certificate authenticated. The server-end certificate authentication can be useful for IBM MQ client connections, for lightweight queue managers, or for any initiating partner where authentication is not necessary.

The SSLCAUTH attribute can have the value of either OPTIONAL or REQUIRED. The default is REQUIRED.

SSLPEER

- Used to check the distinguished name (DN) of the certificate from the peer queue manager or client at the other end of an IBM MQ channel
- Can use wildcards
- Multiple organizational units (OU)
 - Must be specified in descending hierarchical order (such as: OU=Big Unit, OU=Medium Unit, OU=Small Unit)
- Certificate DN mapping with expanded pattern matching and allow/deny capability

Examples:

```
SSLPEER ('CN="Thomas J Watson", O=IBM')
SSLPEER ('OU=Enablement*, O=IBM')
```

Figure 9-41. SSLPEER

The SSLPEER attribute identifies the DN of the certificate of the partner. This field can have wildcards to allow generic matching.

If the DN received from the peer does not match the SSLPEER value, the channel does not start.

SSLPEER is an optional attribute. If the field is left blank or is not present, then the peer DN is not checked when the channel is started.

SSLPEER mapping

- SSLPEER attribute of the channel filters connections that are based on DN of the connection requester
- An alternative way of restricting connections into channels by matching against the TLS subject DN is to use channel authentication records
 - Map certificate DNs to MCAUSER values or specify certificate DNs for which access is denied
 - Rules are hierarchical so it is possible to set “deny all” policy and then override with more specific access

Example:

```
SET CHLAUTH(*) TYPE(SSLPEERMAP) SSLPEER('CN=*')
  USERSRC(NOACCESS)
SET CHLAUTH(*) TYPE(SSLPEERMAP) SSLPEER('OU=ADMIN',O=IBM')
  USERSRC(CHANNEL)
SET CHLAUTH(*) TYPE(SSLPEERMAP) SSLPEER('CN="SWATTS",
  OU=ADMIN, O=IBM') USERSRC(NOACCESS)
```

Figure 9-42. SSLPEER mapping

An alternative way of restricting connections into channels by matching against the TLS Subject Distinguished Name, is to use channel authentication records.

With channel authentication records, different TLS Subject Distinguished Name patterns can be applied to the same channel. If both SSLPEER on the channel and a channel authentication record are used to apply to the same channel, the inbound certificate must match both patterns to connect.

SSLPEER mapping is described in more detail in Unit 3.

Displaying certificate details

- **DISPLAY CHSTATUS** shows partner certificate details
 - **SSLPEER**: The distinguished name of the partner
 - **SSLCERTI**: The certificate issuer's distinguished name of the partner

Examples:

- CA-signed:

```
SSLPEER (CN=MQ01, T=QMGR, O=IBM, L=Sydney, ST=NSW, C=AUST)
SSLCERTI (CN=IBM CA, O=IBM, L>New York, ST=NY, C=US)
```

- Self-signed:

```
SSLPEER (CN=MQ23, T=QMGR, O=IBM, L=Hursley, ST=Hampshire, C=UK)
SSLCERTI (CN=MQ23, T=QMGR, O=IBM, L=Hursley, ST=Hampshire, C=UK)
```

Figure 9-43. Displaying certificate details

The DISPLAY CHSTATUS command provides some information about SSL partner certificate in the SSLPEER and SSLCERTI fields.

Do not confuse SSLPEER on DISPLAY CHSTATUS with SSLPEER on DISPLAY CHANNEL.

- On the channel definition, SSLPEER contains the filter that is matched against the partner distinguished name.
- On channel status, SSLPEER shows the actual DN of the partner certificate.

SSLCERTI contains the issuer DN from the partner certificate.

If SSLPEER and SSLCERTI are the same, the certificate is self-signed. These fields are also passed to the security exit so that more complicated decisions can be made based on this information.

Checking certificate status with OCSP

- IBM MQ determines which OCSP responder to contact in one of two ways:
 - By using the AuthorityInfoAccess (AIA) certificate extension in the certificate
 - By using a URL that is specified in an authentication information object or specified by a client application
- To allow TLS channels to AIA extensions:
 - Ensure that the OCSP server that is named in them is available, correctly configured, and accessible over the network
- If IBM MQ receives an OCSP outcome of “Unknown”, its behavior depends on the setting of the **OCSP authentication** attribute
 - For queue managers, set the `OCSPAuthentication` in the **SSL** stanza of the `qm.ini` file to `WARN` or `OPTIONAL` (`REQUIRED` is the default)
 - For clients, set the `ClientRevocationChecks` attribute in the **SSL** stanza of the client configuration file `client.ini` to `OPTIONAL` or `DISABLED` (`REQUIRED` is the default)

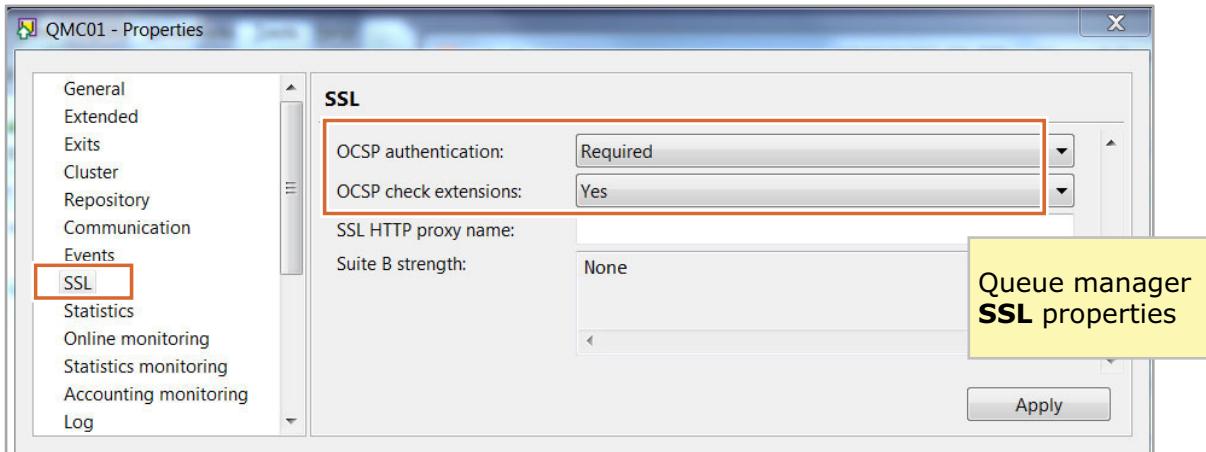
Figure 9-44. Checking certificate status with OCSP

IBM MQ supports OCSP for checking status. Based on configuration settings, IBM MQ determines the responder to use, and handles the response that is received.

IBM MQ determines which OCSP responder to contact by using the AIA certificate extension in the certificate or by using a URL that is specified in an authentication information objects. A URL specified in an authentication information object or by a client application takes priority over a URL in an AIA certificate extension.

The **OCSP authentication** attribute determines the behavior of an “unknown” outcome. On a queue manager, this attribute can be set to REQUIRED, WARN, or OPTIONAL. On a client, the attribute can be set to REQUIRED, OPTIONAL, or DISABLED. REQUIRED is the default value for both clients and queue managers.

Setting OSCP properties in IBM MQ Explorer (1 of 2)



- **OCSP authentication** dictates outcome of a connection when the OCSP call returns “Unknown” response
 - **Required:** IBM MQ rejects the connection
 - **Optional:** Connection succeeds
 - **Warn:** Connection succeeds but IBM MQ writes an AMQ9717 message into the error log
- **OCSP check extensions** controls whether to use the OCSP server details in the AIA certificate extensions for digital revocation check

[Securing IBM MQ channels with TLS](#)

© Copyright IBM Corporation 2020

Figure 9-45. Setting OSCP properties in IBM MQ Explorer (1 of 2)

You can configure the OSCP properties in IBM MQ Explorer on the queue manager **SSL** properties page.



Setting OSCP properties in IBM MQ Explorer (2 of 2)

To override the OSCP responder in the AIA certificate extension:

1. Under queue manager in the **MQ Explorer – Navigator** view, click **Authentication Information > New > OCSP Authentication Information**
2. Enter a name
3. Specify the OCSP responder URL



Figure 9-46. Setting OSCP properties in IBM MQ Explorer (2 of 2)

The URL for OSCP responder that is set in the AIA can be overridden by using IBM MQ Explorer. This figure lists the steps for overriding the OSCP responder URL.

Accessing CRLs and ARLs

- Can be stored in and accessed from LDAP servers

1. Define on AUTHINFO objects

```
DEFINE AUTHINFO (LDAP1)
  AUTHTYPE (CRLLDAP)
  CONNAME ('ldap(389)')
  LDAPUSER ('cn=user')
  LDAPPWD (...)
```

LDAP Connname
LDAP Username
LDAP Password



LDAP Server

2. Put AUTHINFO objects into a namelist

```
DEFINE NL (LDAPNL) NAMES (LDAP1, LDAP2, ...)
```

3. Associate namelist with the queue manager

```
ALTER QMGR SSLCRLNL (LDAPNL)
```



- Clients

- Client channel table contains the LDAP definitions that are present when the table is copied
- Details of the LDAP servers can be held in the Active Directory (Windows only)

Figure 9-47. Accessing CRLs and ARLs

As an alternative to OCSP, you can verify certificates by using CRLs and ARLs.

Queue managers

For queue managers, CRLs and ARLs can be stored in and accessed from LDAP servers. The LDAP CRL and ARL servers are generally defined to be publicly readable.

A few parameters must be specified to access an LDAP server that contains CRLs and ARLs. These parameters are the DNS name or IP address of the LDAP server with an optional TCP/IP port number. The DN of the entry that binds to the directory and the password that is associated with the distinguished name can also be included as optional parameters.

These parameters are defined on a queue manager AUTHINFO object. Several of these objects might be needed to ensure redundancy. Redundancy is provided so that, for example, if the first LDAP server to which a queue manager connects is down, it can connect to another server that can supply the same information. On distributed operating systems, up to 10 of these AUTHINFO objects can be supplied for CRL and ARL checking.

The list of AUTHINFO objects is named in a namelist and this namelist is specified on the SSLCRLNL queue manager attribute by using the ALTER QMGR command.

Clients

The client channel definition table LDAP CRL and ARL server information does not have to match the definitions that are current on the queue manager system when the channel runs.

The MQCONN call provides a structure for IBM MQ authority information records that specify the LDAP CRL and ARL server information.

On Windows, records that specify the access details of the LDAP servers that hold CRL and ARL information are created in the Active Directory by using the `setmqcrl` command. Details of the LDAP server can be held in Active Directory, but the Active Directory cannot be used for CRLs or ARLs.

The CLNTCONN attributes for LDAP servers do not have to match the CLNTCONN attributes of the corresponding SVRCONN channel.

Accessing CRLs and ARLs by using IBM MQ Explorer

1. In the **MQ Explorer**, right-click **Authentication Information** and click **New > CRL LDAP Authentication Information**
 - a. Enter a name for the CRL LDAP object
 - b. Enter the LDAP server name as either the network name or the IP address
 - c. If the server requires login details, provide a user ID and password
2. In the **MQ Explorer**, right-click **Namelists** and click **New > Namelist**
 - a. Enter a name for the namelist
 - b. Add the name of the CRL LDAP object (from Step 1A) to the list
3. On the queue manager **SSL** properties page
 - a. Enable **Check certificates received by this queue manager for revocation Lists**
 - b. Type the name of the namelist (from Step 2A) in the **Revocation Namelist** field

Securing IBM MQ channels with TLS

© Copyright IBM Corporation 2020

Figure 9-48. Accessing CRLs and ARLs by using IBM MQ Explorer

You can identify the LDAP CRL and ARL information by using MQSC or IBM MQ Explorer. This figure lists the steps for configuring the access to CRLs and ARLs by using IBM MQ Explorer.

Possible authentication failures

- A certificate was found in a CRL or ARL
- An OCSP responder identified a certificate as “revoked” or “unknown”
- A matching CA root certificate does not exist or the certificate chain is incomplete

Securing IBM MQ channels with TLS

© Copyright IBM Corporation 2020

Figure 9-49. Possible authentication failures

Authentication failures during the TLS handshake can occur for a number of reasons.

- A CA can revoke a certificate that is no longer trusted by publishing it in a CRL or ARL.
- An OCSP responder can return a response of “revoked”, which indicates that a certificate is no longer valid. A response of “unknown” indicates that OCSP does not have revocation data for that certificate.
- Each digital certificate from a CA also provides a root certificate that contains the public key for the CA. The CA signs the root certificates. If the key repository on the computer that is authenticating the certificate does not contain a valid root certificate for the CA that sent the incoming user certificate, authentication fails.

Other possible reasons for authentication errors include the following reasons:

- A certificate expired or is not active
- A certificate is corrupted
- The TLS client does not have a certificate

Using secret key reset

- IBM MQ supports the resetting of secret keys on queue managers and clients when a specified number of encrypted bytes of data flowed across the channel
- If channel heartbeats are enabled, the secret key is reset before data is sent or received following a channel heartbeat
- Set a specified number of bytes

```
ALTER QMGR SSLRKEYC(999 999 999)
```
- Display channel status shows:
 - Number of times the key was reset
 - Time and date of the last reset

```
DISPLAY CHSTATUS(*) SSLRKEYS SSLKEYDA SSLKEYTI
```

Figure 9-50. Using secret key reset

A TLS channel periodically renegotiates its secret key if secret key reset is enabled. This renegotiation takes place after a specified amount of data is moved across the channel. It also takes place before data is sent across a channel that was idle for a time and sent heartbeat flows.

For a queue manager, use the command ALTER QMGR with the parameter SSLRKEYC to set the values that are used during key renegotiation.

Fields in the channel status show when the last reset was completed and how many resets were completed for the life of the channel instance.

By default, MQI clients do not renegotiate the secret key. You can make an MQI client renegotiate the key in any of three ways. In the following list, the methods are shown in order of priority. If you specify multiple values, the highest priority value is used.

1. Use the KeyResetCount field in the MQSCO structure on an MQCONN call.
2. Use the environment variable MQSSLRESET.
3. Set the SSLKeyResetCount attribute in the MQI client configuration file.

Refreshing TLS on IBM MQ

- Cached views of key repository
- Changes require refresh of cached views
 - New certificates
 - Deleted certificates
 - Updated certificates that replace expiring certificates
- MQSC command **REFRESH SECURITY TYPE(SSL)**
 - Stops all TLS channels
 - New cached views of the key repository are created
 - Restarts all sender TLS channels
 - Restarts receiver channels when partner tries again
 - Updates new key repository and LDAP CRL and ARL locations

[Securing IBM MQ channels with TLS](#)

© Copyright IBM Corporation 2020

Figure 9-51. Refreshing TLS on IBM MQ

The TLS environment that is set up to run TLS channels in a channel process has a cached view of the key repository that is made at initialization time.

If you change your key repository for any reason, you must refresh the cached view so that the TLS channels use start the new certificates.

To refresh this cached view of the TLS environment without disrupting any channels that are not enabled for TLS, use the REFRESH SECURITY TYPE(SSL) command. The REFRESH command stops all the TLS channels on the queue manager. New cached views of the key repository are made and all the sending type channels are started again. Receiving type channels are restarted when the partner tries to connect again.

When you enter the REFRESH SECURITY TYPE(SSL) MQSC command, all running TLS channels are stopped and restarted. Sometimes TLS channels can take a long time to shut down. IBM MQ sets a time limit of 10 minutes for a TLS refresh to complete.

Also, you can use this command to recognize other changes, such as new key repository locations, or new LDAP CRL and ARL locations.

Security administration tasks

- Creating certificates and certificate requests
 - Different tools are used on various operating systems
- Storing certificates, public keys, and private keys in key database files
- Managing binary certificates cross-platform
 - DER, CER, BER encodings, such as PKCS #7 DER encoded X.509 certificate
 - PKCS #12 DER encoded X.509 certificate (password protected)
- Managing text certificates cross-platform
 - Must be transferred with text conversion, for example, ASCII to EBCDIC
 - Privacy Enhanced Mail (PEM) encoded X.509 certificate
 - Base64 encoded certificate

Figure 9-52. Security administration tasks

Security implementation requires some administration.

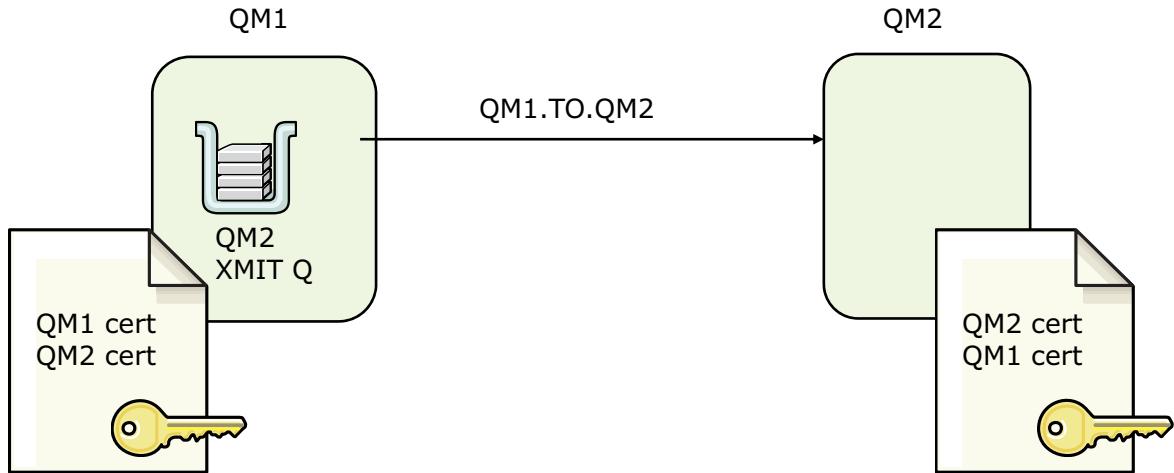
For example, security implementation requires the administration of key repositories and digital certificates. Certificate authority certificates are stored in the key repository so that they can be used to validate certificates that are received from the partner system.

Certificates can be created on the system. They can also come from outside the system, in which case they must be imported onto the system. Several standard formats exist. Some of these formats are binary and must be transported in their exact binary format. In contrast, text formats must be transported as text. If they are transported between an ASCII and EBCDIC system, the data must be converted from ASCII to EBCDIC.

PKCS #12 files contain a personal certificate and optionally, its private key and CA certificates for its signing CAs.

TLS implementation scenario (1 of 3)

- Two queue managers, QM1 and QM2, need to communicate securely and require mutual authentication
 - Channels are configured and verified without security
 - Configure and test the secure communication by using self-signed certificates



Securing IBM MQ channels with TLS

© Copyright IBM Corporation 2020

Figure 9-53. TLS implementation scenario (1 of 3)

This scenario summarizes the TLS administration tasks by using a TLS implementation example. In this scenario, two queue managers need to communicate securely and require mutual authentication. Self-signed certificates are used to test the implementation of the security requirements.

In this scenario, the key repository for queue manager QM1 contains the certificate for QM1 and the public certificate from QM2. The key repository for queue manager QM2 contains the certificate for QM2 and the public certificate from QM1.

TLS implementation scenario (2 of 3)

1. Prepare the key repository on each queue manager
2. Create a self-signed certificate for each queue manager
3. Extract a copy of each certificate
4. Transfer the public part of the QM1 certificate to QM2 and the public part of the QM2 certificate to QM1 by using a utility such as FTP
5. Add the partner certificate to the key repository for each queue manager
6. Stop the sender channel
7. On QM1, modify the sender channel for TLS and refresh security

```
ALTER CHANNEL (QM1.TO.QM2) CHLTYPE (SDR) +
SSLCIPH (TLS_RSA_WITH_AES_256_GCM_SHA384) +
DESCR ('Sender using TLS')
REFRESH SECURITY TYPE (SSL)
```

Figure 9-54. TLS implementation scenario (2 of 3)

This figure lists the first seven steps for implementing TLS for this scenario. The detailed information for completing each of these steps is provided in previous pages on this unit.

TLS implementation scenario (3 of 3)

8. On QM2, modify the receiver channel and refresh security

```
ALTER CHANNEL (QM1.TO.QM2) CHLTYPE (RCVR) +
SSLCAUTH (REQUIRED) +
DESCR ('Receiver using TLS')
REFRESH SECURITY TYPE(SSL)
```

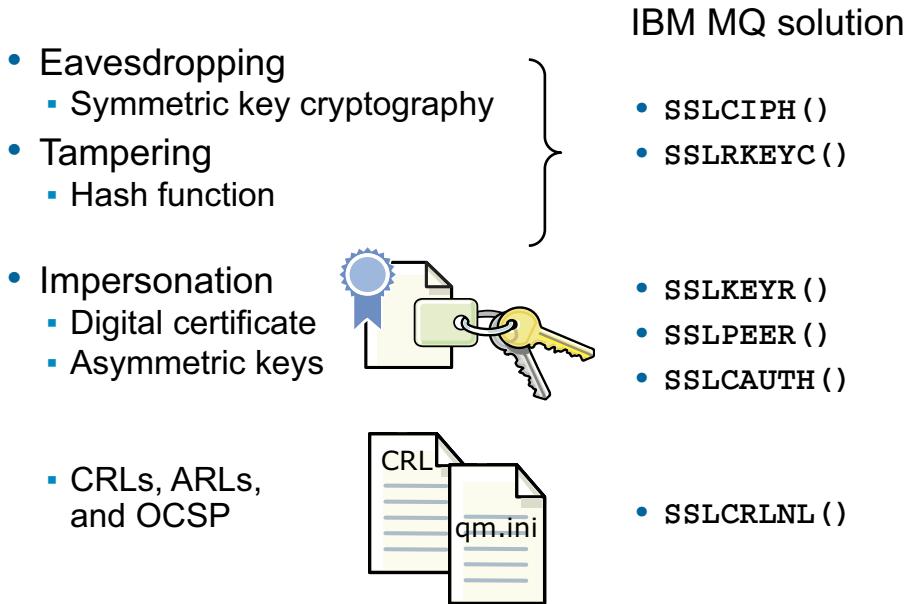
The channel uses the same cipher specification that is specified on QM1

9. Start the channel

Figure 9-55. TLS implementation scenario (3 of 3)

This figure lists the final two steps for implementing security for this scenario.

Security problems and solutions



[Securing IBM MQ channels with TLS](#)

© Copyright IBM Corporation 2020

Figure 9-56. Security problems and solutions

Three main security problems, eavesdropping, tampering, and impersonation were described in this unit.

- For eavesdropping, you can use symmetric key cryptography.
- For tampering, you can use the hash function.
- For impersonation, you can use digital certificates, asymmetric keys, and certificate revocation lists.

IBM MQ uses of all of these techniques to provide solutions to security problems. You can specify the symmetric key cryptography algorithm and the hash function to use by providing IBM MQ with a cipher specification.

Digital certificates and public keys are found in a key repository, which can be specified to IBM MQ. You can check to make sure that you are talking to the partner you expect. You can choose to authenticate both ends of the connection or only the TLS server end of the connection. Also, you can use CRLs, ARLs, or an OCSP responder to verify the certificate.

IBM MQ Advanced Message Security

- Provides a high level of protection for sensitive data that flows through the IBM MQ network, while not impacting the end applications
- Secures sensitive or high-value transactions that IBM MQ processes
- Detects and removes rogue or unauthorized messages before a receiving application processes the messages
- Verifies that messages were not modified while in transit from queue to queue
- Protects the data not only as it flows across the network but also when it is put on a queue
- Secures existing proprietary and customer-written applications for IBM MQ
- IBM MQ Advanced license includes entitlement for IBM MQ Advanced Message Security

[Securing IBM MQ channels with TLS](#)

© Copyright IBM Corporation 2020

Figure 9-57. IBM MQ Advanced Message Security

The focus of this unit is channel-level security. For message-level security, you can use IBM MQ Advanced Message Security to provide the following functions:

- Protect message data while the message is in the queue and as it flows across the network.
- Attach digital signatures to individual messages so that you can verify whether someone tampered with them. You can also identify and trace messages to their point of origin.
- Encrypt messages to shield them from disclosure to unauthorized parties and centrally define and enforce security policies by using a web-based interface.
- Provide security-specific, fine-grained auditing records to document adherence to security policy, thus protecting highly sensitive transactions records.
- Help accelerate new application deployment by reducing development time and costs by providing application level security without requiring complex security coding.

IBM MQ Advanced Message Security is a separately licensed component of IBM MQ. Entitlement to IBM MQ Advanced Message Security is included with the IBM MQ Advanced license.

Unit summary

- Describe the certificate infrastructure that is supported in IBM MQ
- Manage certificates with IBM Key Management
- Describe cipher specifications and their support in IBM MQ
- Use certificate revocation lists or Online Certificate Status Protocol (OCSP) to validate currency of certificates
- Use TLS to secure IBM MQ channel communications

[Securing IBM MQ channels with TLS](#)

© Copyright IBM Corporation 2020

Figure 9-58. Unit summary

Review questions

1. SSLPEER is an optional parameter that:
 - A. Defines whether the channel needs to receive and authenticate TLS certificates from a TLS client.
 - B. Defines a single cipher specification for a TLS connection.
 - C. Is used to check the DN of the certificate from the peer queue manager or client at the other end of an IBM MQ channel.

2. On UNIX and Linux the digital certificates have labels that are used to:
 - A. Find the correct certificate.
 - B. Allow indexing.
 - C. Provide a description of the certificate.
 - D. Associate a certificate with a queue manager.

3. True or False: IBM Key Management can be used to sign personal certificates as a CA.



[Securing IBM MQ channels with TLS](#)

© Copyright IBM Corporation 2020

Figure 9-59. Review questions

Write your answers here:

1.

2.

3. .

Review answers

1. SSLPEER is an optional parameter that:
 - A. Defines whether the channel needs to receive and authenticate TLS certificates from a TLS client.
 - B. Defines a single cipher specification for a TLS connection.
 - C. Is used to check the DN of the certificate from the peer queue manager or client at the other end of an IBM MQ channel.

The answer is C.

2. On UNIX and Linux the digital certificates have labels that are used to:
 - A. Find the correct certificate.
 - B. Allow indexing.
 - C. Provide a description of the certificate.
 - D. Associate a certificate with a queue manager.

The answer is D.

3. True or False: IBM Key Management can be used to sign personal certificates as a CA.

The answer is False. IBM Key Management can be used to create keys and manage SSL certificates. It cannot act as a CA.

Figure 9-60. Review answers

Exercise: Securing channels with TLS

Securing IBM MQ channels with TLS

© Copyright IBM Corporation 2020

Figure 9-61. Exercise: Securing channels with TLS

Exercise introduction

- Use IBM Key Management to create a certificate request
- Secure channels by using TLS on the channel



Securing IBM MQ channels with TLS

© Copyright IBM Corporation 2020

Figure 9-62. Exercise introduction

Unit 10. Authenticating channels and connections

Estimated time

01:30

Overview

In this unit, you learn how to use channel authentication to control the access that is granted to connecting systems at a channel level. You learn how to modify the queue manager to use the local operating system or an LDAP server to authenticate user IDs and passwords of clients or applications that are requesting access to IBM MQ resources. This unit also describes channel exit programs and administration.

How you will check your progress

- Review questions
- Lab exercise

Unit objectives

- Determine the current level of authentication that is enabled on a queue manager and a connection
- Add authentication to a channel
- Add authentication to a connection
- Identify and fix channel authentication and connection authentication problems
- Implement a channel exit program for securing messaging channels

Topics

- Authentication and authorization overview
- Object authorization review
- Connection authentication
- Channel authentication
- Channel exit programs

Authenticating channels and connections

© Copyright IBM Corporation 2020

Figure 10-2. Topics

10.1. Authentication and authorization overview

Authentication and authorization overview

Authenticating channels and connections

© Copyright IBM Corporation 2020

Figure 10-3. Authentication and authorization overview

Authentication

- Establishing the identity of the user
- Basic authentication: Asking for a user ID and password
- Multifactor authentication can be a combination of:
 - Something that you know, such as a password
 - Something that you have, such as a certificate or hardware token
 - Something that you are, such as a fingerprint or retinal scan
- Provides some level of assurance that the identity that is presented is genuine
- Complexity of authentication typically depends on the importance of keeping data protected

Authenticating channels and connections

© Copyright IBM Corporation 2020

Figure 10-4. Authentication

Authentication is the process of establishing the identity of the user. The user in this case might be an application or another queue manager.

Basic authentication is a user ID and password. Extra authentication provides some level of assurance that the identity that is presented is genuine.

Authorization

- Control access to resources on a per-ID, per-role, or per-profile basis
- Authorization response is an absolute concept where either a "Yes" or "No" answer is returned for any access request
- A resource can support many different functions such as CONNECT, OPEN, and CLOSE, and many different options such as INPUT, OUTPUT, BROWSE, and INQUIRE
- Answers the question “Does user X have access type Y to resource Z?”

Figure 10-5. Authorization

Authorization is the control of access to resources based on an ID, user role, or user profile.

Identification and authentication in IBM MQ

- You can implement identification and authentication by using:
 - Message context information
 - *Mutual authentication* by the MCA at each end of a channel when a message channel starts
 - Connection authentication
- IBM MQ requires a user ID to make authorization decisions
- IBM MQ needs to understand where the user ID comes from
- What leads to security exposure?
 - Not understanding how IBM MQ determines the user ID
 - Thinking that you understand it
 - Incorrect settings in your definitions

Figure 10-6. Identification and authentication in IBM MQ

In IBM MQ, you can implement identification and authentication by using message context information. You can also implement mutual authentication by the MCA at each end of a channel when a message channel starts.

Approach to planning security

- Secure administrative access
 - If no meaningful authentication exists, then security can be circumvented
- Any more advanced security hardening depends on securing administrative access
 - If administrative access is not locked down, no accountability exists and security controls can easily be bypassed
- Resulting security model provides three types of access:
 - Administrator
 - Remote queue manager (for queue manager to queue manager access)
 - Client channel user (client or application)

Figure 10-7. Approach to planning security

When planning or reviewing IBM MQ security, you first need to secure administrative access.

Securing administrative access means making sure that nobody can programmatically assert a user ID with IBM MQ administration privileges.

Any security plan must consider all types of access to data and IBM MQ objects.

Secure connections in IBM MQ Explorer

- **IBM MQ Explorer Preferences**
 - Default values for client-connection security exits, TLS default values, and user ID and password
 - Passwords that IBM MQ Explorer uses
- **Add Remote Queue Manager** wizard for each remote queue manager connection
- Channel authentication precedence mapping

Authenticating channels and connections

© Copyright IBM Corporation 2020

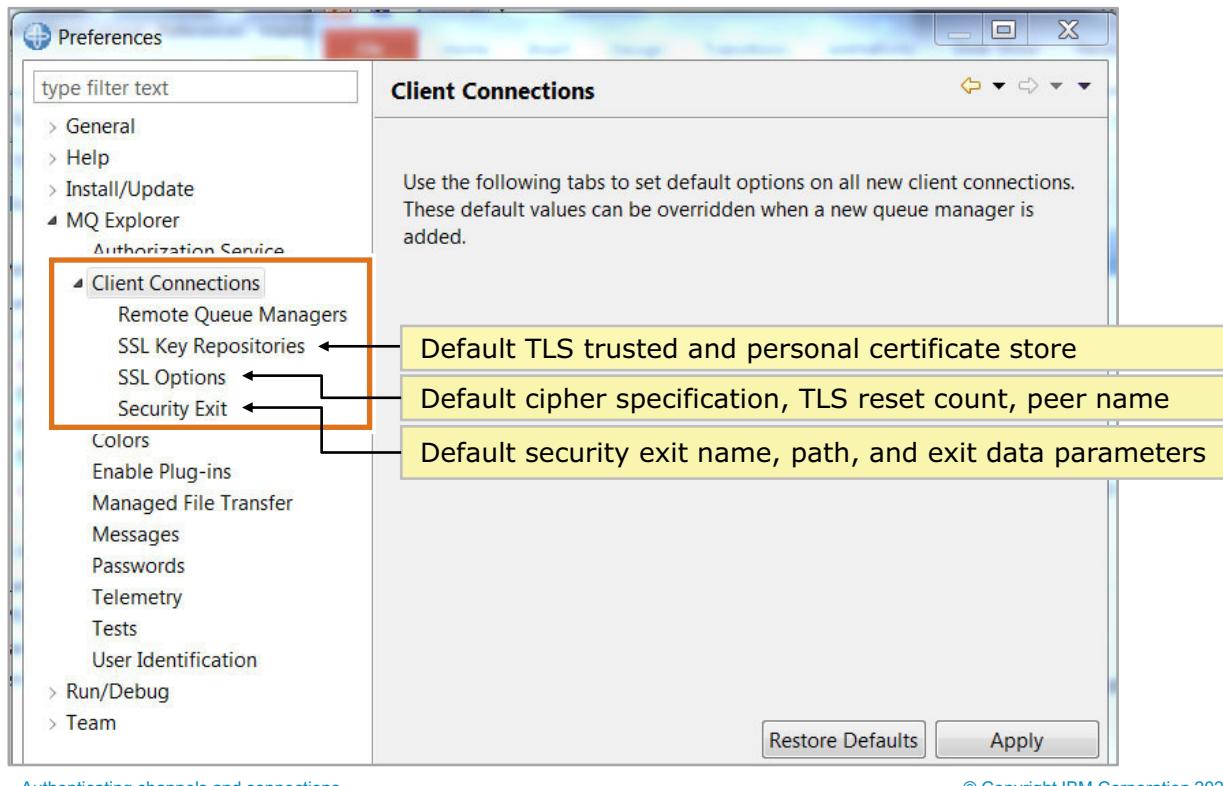
Figure 10-8. Secure connections in IBM MQ Explorer

The security provisions in IBM MQ include securing channels with TLS and controlling access to IBM MQ objects. You can manage both TLS security and object authorities in IBM MQ Explorer.

IBM MQ secures the connection to a queue manager with a user ID and password. You can also configure different security options and channel authentication (CHLAUTH) mapping for each queue manager.



Setting client connection default values



Authenticating channels and connections

© Copyright IBM Corporation 2020

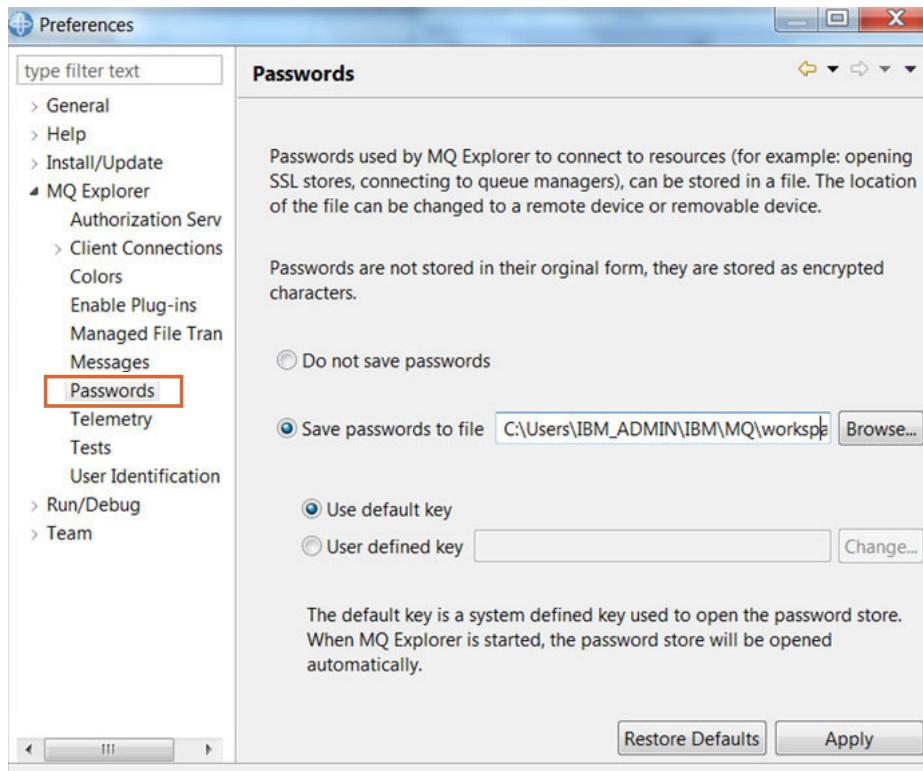
Figure 10-9. Setting client connection default values

You set security preferences for client connections in IBM MQ Explorer through the **Window > Preferences** in menu. Alternatively, you can right-click **IBM MQ** in the **MQ Explorer - Navigator** view and then click **Preferences**.

In the left pane of **Preferences** window, expand **MQ Explorer** and then expand **Client Connections**.



Setting passwords that IBM MQ Explorer uses



Authenticating channels and connections

© Copyright IBM Corporation 2020

Figure 10-10. Setting passwords that IBM MQ Explorer uses

Passwords that IBM MQ Explorer uses to connect to resources, such as opening TLS stores or connecting to queue managers, can be stored in a file.

To set default preferences for passwords in IBM MQ Explorer, from the **Window > Preferences** menu, expand **MQ Explorer** and click **Passwords**.

When using the option **Use default key**, the password store opens automatically when it is needed. The option **User defined key** means that the password store must be accessed with a specific password. In this case, the password window is always displayed when the password store needs to be opened for the first time after IBM MQ Explorer is started.

Security for remote queue manager connection

- **Add Remote Queue Manager** wizard security options
 - Security exit name, path, and exit data
 - User ID and password for client connections
 - TLS trusted certificate store, and personal certificate store
 - Cipher specification, TLS reset count, and peer name

Authenticating channels and connections

© Copyright IBM Corporation 2020

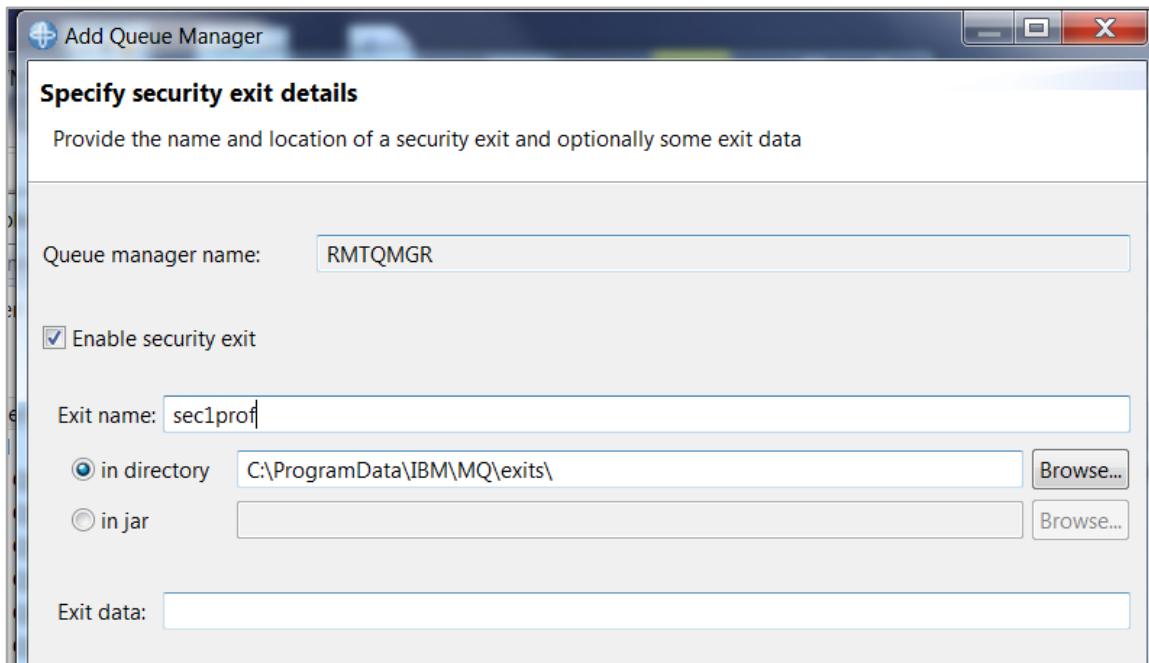
Figure 10-11. Security for remote queue manager connection

The **Add Remote Queue Manager** wizard in IBM MQ Explorer allows you to define specific security connection information for each queue manager.

In the wizard, you define the basic connection information such as queue manager name, TCP/IP address, and port. You can also define security information for the remote queue manager connection.



Security settings for remote queue managers (1 of 4)



Authenticating channels and connections

© Copyright IBM Corporation 2020

Figure 10-12. Security settings for remote queue managers (1 of 4)

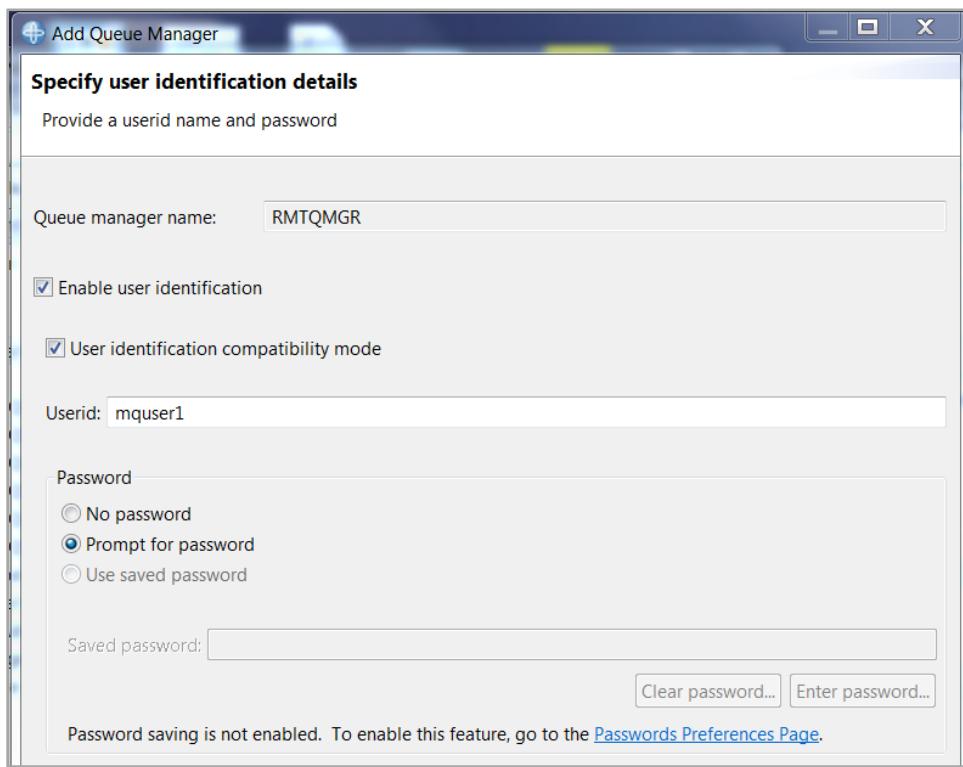
The first two pages in the **Add Remote Queue Manager** wizard define the basic connection information. The next four pages define the security specifications.

To define a connection to a remote queue manager, right-click the **Queue Managers** folder in the **MQ Explorer - Navigator** view and then click **Add Remote Queue Manager**.

The main purpose of this wizard is to connect to a remote queue manager that is enabled for remote administration. The security pages that are displayed in the figure and the next three pages show four of the six pages that are configured during the connection.

The security exit details page that is shown in the figure, is used to enable a security exit. Select **Enable security exit** to provide the name and location of a security exit. You can also choose to add some exit data.

Security settings for remote queue managers (2 of 4)



Authenticating channels and connections

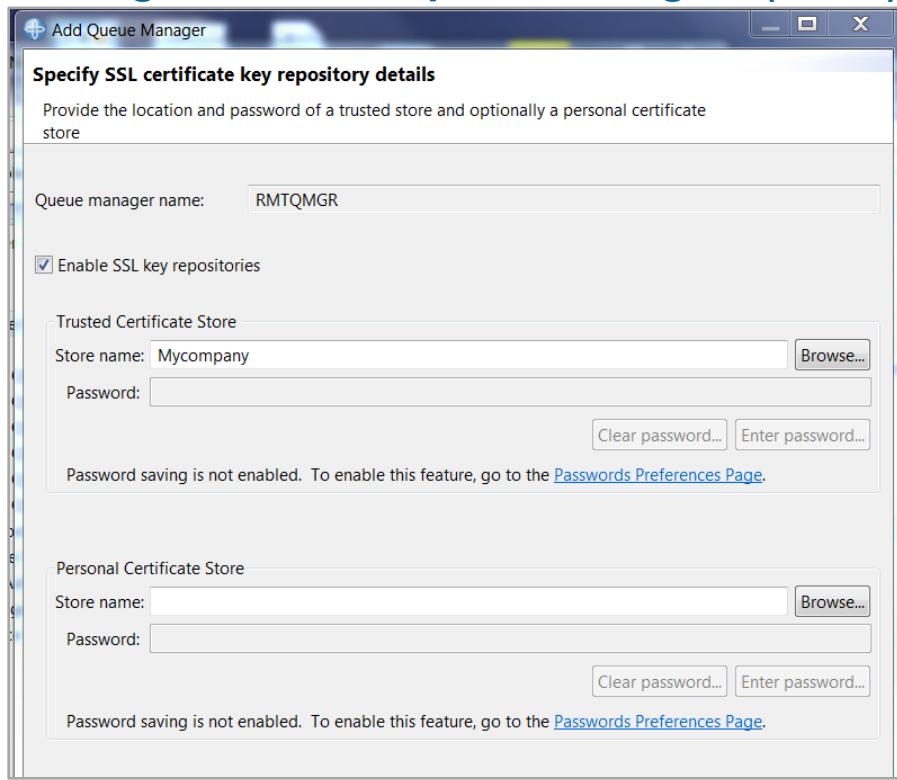
© Copyright IBM Corporation 2020

Figure 10-13. Security settings for remote queue managers (2 of 4)

In the next page in the **Add Remote Queue Manager** wizard, you can enable user identification.

Select **Enable user identification** to provide a user ID name and password.

Security settings for remote queue managers (3 of 4)



Authenticating channels and connections

© Copyright IBM Corporation 2020

Figure 10-14. Security settings for remote queue managers (3 of 4)

On the next page in the **Add Remote Queue Manager** wizard, you can provide TLS certificate key repository details.

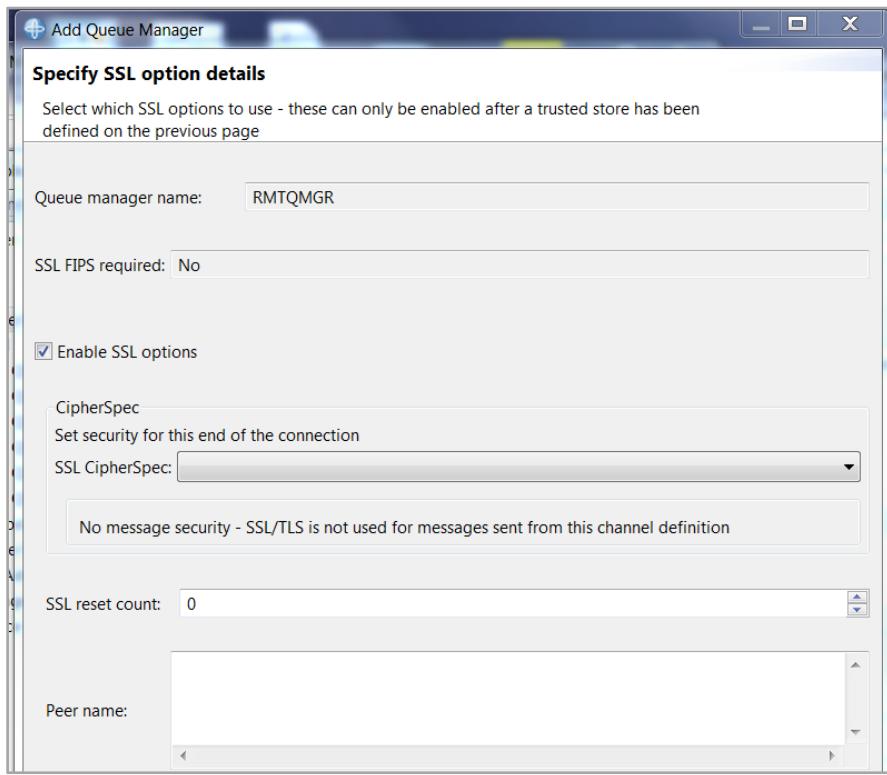
Select **Enable SSL key repositories** to provide the location and password of a trusted store. You can also choose to add the details of a personal certificate store. TLS security configuration was described in detail in a previous unit.



Note

In this version of IBM MQ, all SSL cipher specifications are deprecated so you must use TLS for security. You enable and configure TLS by using the SSL options in the IBM MQ Explorer and commands options in MQSC.

Security settings for remote queue managers (4 of 4)



Authenticating channels and connections

© Copyright IBM Corporation 2020

Figure 10-15. Security settings for remote queue managers (4 of 4)

On the next page in the **Add Remote Queue Manager** wizard, you can enable the TLS options.



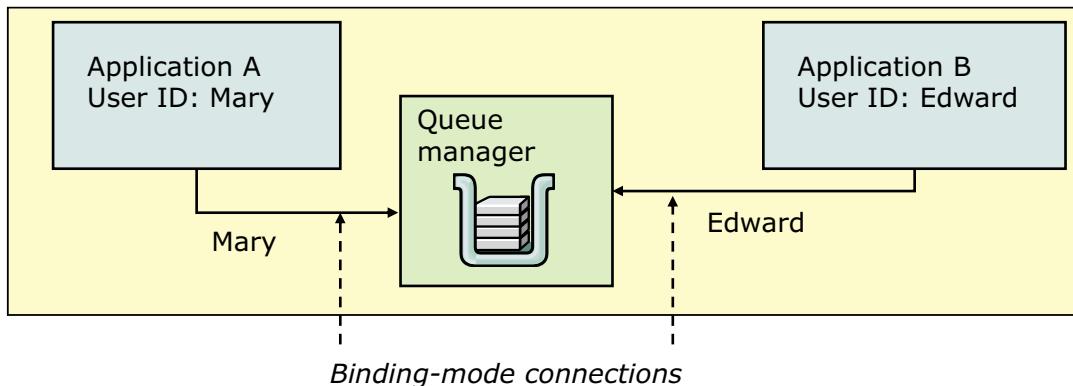
Note

TLS options can be enabled only after a trusted store is defined in the wizard.

Select the **Enable SSL options** check box to identify the TLS cipher specification and configure the other TLS options.

User ID when using bindings mode

- Applications connecting to queue manager in bindings mode
 - Application process user ID is assigned through an operating system process, such as user logon
 - User ID flows to the queue manager
- For local bindings-mode connections, the IBM MQ security model default is to “deny all”



[Authenticating channels and connections](#)

© Copyright IBM Corporation 2020

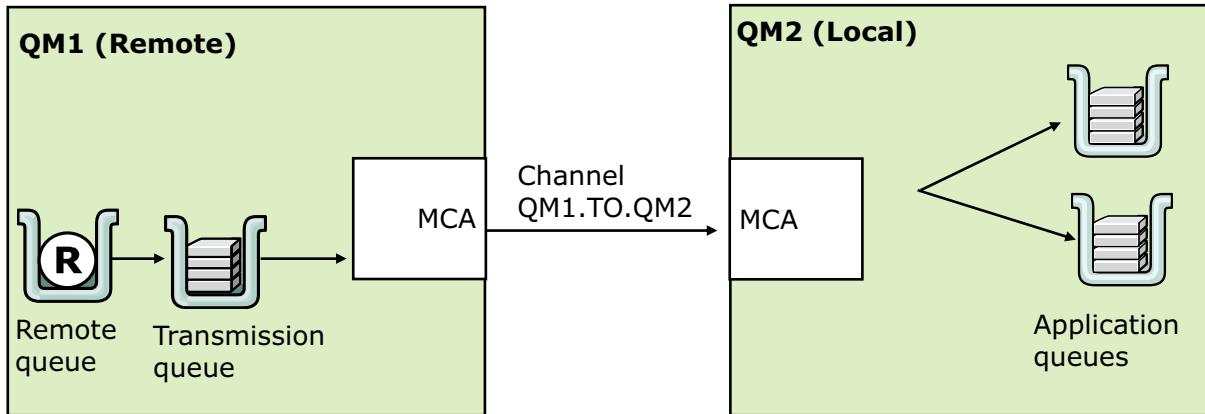
Figure 10-16. User ID when using bindings mode

When an application runs on the same server as the queue manager, it can connect in bindings mode. Bindings mode means that the application process communicates with the queue manager process by using memory. No channels are used when bindings mode is used. The application process has a user ID associated with it. When it connects to the queue manager, the user ID under which the application process runs is also the user ID that the queue manager uses to make authorization decisions.

In this example, the operating system authenticated the user ID. The queue manager trusts the identity that the operating system presents.

Queue manager to queue manager

- MCA at each end of the channel between queue managers
 - User ID used is that of the MCA
 - By default, MCA puts messages with full administrative authority
 - Process can be overridden by setting the **MCAUSER** attribute on the channel



Authenticating channels and connections

© Copyright IBM Corporation 2020

Figure 10-17. Queue manager to queue manager

When the local queue manager receives messages from other queue managers, the user ID of the MCA is used to make authorization decisions, such as whether the message can be written to a queue. If no MCAUSER value is set in the receiving channel definition, then the MCA puts messages with administrative authority.

These rules apply only if PUTAUT(CTX) is not set on the channel. If PUTAUT(CTX) is set on the channel, the user ID from the context information associated with the message is used as an alternative user ID. IBM MQ uses this user ID to make authorization decisions.

Use of PUTAUT(CTX) implies that messages that arrive over that channel can obtain full administrative authority.

Why using TLS can lead to a false sense of security

- IBM MQ can use TLS over all channel types
- TLS provides assurance that someone who starts an MQI channel possesses a valid certificate
- TLS optionally encrypts the data so that data cannot be viewed in transit
- TLS starts and ends with the MCA
 - Authentication that TLS provides does not extend to the API calls
 - After the connection is established, everything outside the TLS exchange (the connection request, the API calls) works exactly as it does without TLS
- TLS does not affect authorization behavior

Figure 10-18. Why using TLS can lead to a false sense of security

IBM MQ allows you to specify the use of TLS over channels. Sometimes administrators might think their IBM MQ environment is secure because they enabled the use of TLS on all channels, which leads to a false sense of security. TLS, when properly configured, limits connections to legitimate remote nodes. For MCA channels (such as RQSTR, RCVR, CLUSRCVR), TLS does not change the fact that they run with full administrative authority by default.

TLS does not limit the queues onto which an MCA channel can put messages.

TLS does not prevent the legitimate user of an MQI (SVRCONN) channel from asserting any arbitrary user ID. Therefore, TLS does not establish a context that extends to the API on which the OAM can make authorization decisions.

The use of TLS begins and ends with the connection.

The MCA does not directly use any information that is available from the context of the TLS session.

TLS does not affect anything at the API layer. Therefore, TLS allows you to authenticate the remote node. However, you still do not have a trusted identity for API calls.

IBM MQ Advanced Message Security

- Provides the flexibility to add application-level data protection and remote security policy administration, which includes:
 - Signing each message with a unique private key that is associated with the sending application
 - Encrypting individual messages under unique keys, thus helping to remove the threat of compromising the encryption key through repetitive use
- Provides the remote administration of security policies on queue managers and on individual queues
- Protects message data while the message is in queue and as it flows across the network
- Attaches digital signatures to individual messages so you can:
 - Verify whether anyone tampers with them
 - Identify and trace messages to their point of origin

Authenticating channels and connections

© Copyright IBM Corporation 2020

Figure 10-19. IBM MQ Advanced Message Security

IBM MQ Advanced Message Security extends IBM MQ security.

IBM MQ Advanced Message Security supports the following features for both IBM MQ servers and clients:

- Application-level, end-to-end data protection for your point-to-point messaging infrastructure, by using either encryption, or digital signing of messages
- Comprehensive security without writing complex security code or modifying or recompiling existing applications
- Use of Public Key Infrastructure (PKI) technology to provide authentication, authorization, confidentiality, and data integrity services for messages
- Administration of security policies for mainframe and distributed servers

10.2. Object authorizations review

Object authorizations review

Authenticating channels and connections

© Copyright IBM Corporation 2020

Figure 10-20. Object authorizations review

Object authority manager (OAM)

- OAM is started by default when the queue manager starts
- For MQI requests, OAM checks the authorization of the entity that is associated with the operation to check:
 - Can the requested operation be completed?
 - Can the specified queue manager resources be accessed?
- IBM MQ Explorer, `setmqaut` commands, and MQSC SET AUTHREC commands define authorization rules in the OAM
- Specify authority profiles in the `setmqaut` command:
 - SETALL, ALLMQI, ALLADMIN, ALL, and PASSALL
 - These authority profiles confer administrative privileges, which should not be granted routinely
 - Can use discrete privileges, such as put, get, and inquire

Figure 10-21. Object authority manager (OAM)

The authorization service component that is supplied with the IBM MQ products is called the Object Authority Manager (OAM).

When an MQI request is made or a command is entered, the OAM checks the authorization of the entity that is associated with the operation to see whether it can complete the requested operation and access the specified queue manager resources.

By default, the OAM is active and works with the control commands `dsprmqaut` (display authority), `dmpmqaut` (memory dump authority), and `setmqaut` (set or reset authority).

The OAM works with the entity of a principal or group:

- On UNIX and Linux systems, a principal is a user ID, or an ID associated with an application program that runs on behalf of a user; a group is a system-defined collection of principals.
- On Windows systems, a principal is a Windows user ID, or an ID associated with an application program that runs on behalf of a user; a group is a Windows group.

Authorizations can be granted or revoked at the principal or group level.

Managing authority records in MQSC

- **SET AUTHREC** to set authority records that are associated with a profile name

Example:

```
SET AUTHREC PROFILE(APP.QUEUE) OBJTYPE(QUEUE) +
GROUP('appuser') AUTHADD(INQ, DSP, BROWSE, PUT, GET)
```

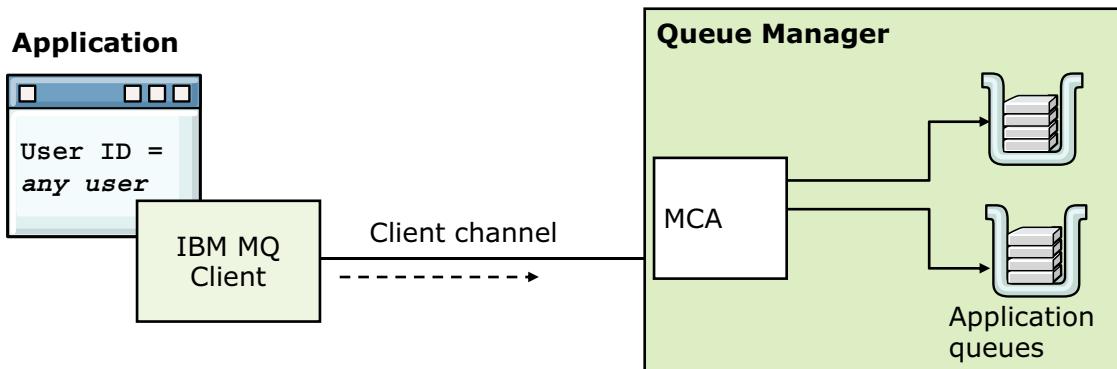
- **DISPLAY AUTHREC** to display the authority records associated with a profile name
- **DELETE AUTHREC** to delete authority records that are associated with a profile name

Figure 10-22. Managing authority records in MQSC

You can also manage authority records by using the SET AUTHREC, DISPLAY AUTHREC, and DELETE AUTHREC commands.

Application to queue manager

- Application can be:
 - A long-running batch type process
 - Interactive users
- If MCAUSER is not set in SVRCONN channel definition, the application can claim to be any user ID



Authenticating channels and connections

© Copyright IBM Corporation 2020

Figure 10-23. Application to queue manager

When an application connects to a queue manager over a client channel, the default behavior for the IBM MQ client is to get the logged on user ID. The client then passes the user ID to the queue manager, where it is set in the MCAUSER field. This behavior can be overridden with application code or channel security exits on the client-side to present any arbitrary user ID. So, it is never safe to trust the ID presented over a server-connection (SVRCONN) channel.

Blank MCAUSER: Main cause of security exposures

- Any inbound channel definition with a blank MCAUSER value implicitly allows:
 - User impersonation
 - Administrative authority
- To check your system:
 1. In MQSC, enter: DISPLAY CHANNEL (*) MCAUSER
 2. If any inbound channel has MCAUSER(), then you have a security exposure

Note: This security exposure applies to any channels that are created automatically

Figure 10-24. Blank MCAUSER: Main cause of security exposures

If you have a channel with a blank MCAUSER value, an unauthorized user can access your queue manager with full administrative authority. If someone used this channel, you would not be able to determine who it was.

Use the DISPLAY CHANNEL command to check for a blank MCAUSER on a channel.

Secure the SYSTEM.DEF and SYSTEM.AUTO channel definitions. IBM MQ administrators sometimes incorrectly assume that the SYSTEM.AUTO.* definitions are not usable by IBM MQ if the CHAD (automatic channel definition) attribute of the queue manager is set to DISABLED but it is not the case.

MCAUSER remedy

- Provision two low-privileged user IDs and corresponding private groups
 - One for MCA channels
 - One for MQI (SVRCONN) channels
- In the channel definition:
 - Set **MCAUSER** to the user ID depending on channel type
 - Set **PUTAUT** to the value of **DEF**
- Authorize the private group to the application queues
 - Equivalent to using a blank MCAUSER in that only one authorization profile is used, but it is a low-privilege profile
- If clustering is used, also authorize the user ID to the **SYSTEM.CLUSTER.COMMAND.QUEUE**
- Do not authorize this user ID to any transmit queues
 - If multihop routing is required, use remote queue definitions

[Authenticating channels and connections](#)

© Copyright IBM Corporation 2020

Figure 10-25. MCAUSER remedy

The figure lists the configuration options that you can implement to ensure that an MCAUSER is not blank.

An alternative way of providing a user ID for a channel to run under is to use channel authentication records. With channel authentication records, different connections can use the same channel while using different credentials. If both MCAUSER on the channel is set and channel authentication records are used to apply to the same channel, the channel authentication records take precedence. The MCAUSER on the channel definition is only used if the channel authentication record uses USERSRC(CHANNEL).

Secure remote administration

- Setting MCAUSER might mean you cannot use remote IBM MQ administration
 - IBM MQ administrators must log on to the server locally
 - IBM MQ provides ways to securely allow secure remote administration
- Client-channel security exit
 - Provides a way to authenticate the user at the client end
 - Passes this user ID to the server end
- Use of TLS and client certificate:
 - Can use TLS authentication to secure client channels
 - Configure it to allow only administrators to connect

Figure 10-26. Secure remote administration

While setting MCAUSER can prevent someone from inappropriately obtaining administration access to a queue manager, the downside is that it also prevents legitimate access. The IBM MQ administrator now needs to log on to the server on which the queue manager is running to administer it. IBM MQ administration might not be much of a problem if you have a few queue managers only. But if there are many queue managers, or if they are clustered, then not being able to use a tool like IBM MQ Explorer to manage multiple queue managers creates inefficiencies. So, it is still necessary to provide secure remote administration.

To provide secure remote administration, you need to configure the environment so that you can be sure that only those people who need this capability can obtain it.

One approach is to develop a channel security exit for use on client channels. A security exit is able to set a user ID on the MCAUSER field. The exit on the client-side would need to gather the user's ID so that you can be sure that the user is the one you expect. It would then pass this user ID to the exit on the queue manager end, where it would be set in the MCAUSER field. The security exits at each end of the client channel can communicate with each other. Coding channel security exits is a nontrivial task. Also, if the client-side security exit is passing credentials, such as a user ID and password, to the server-side security exit, then use SSL to encrypt this communication.

Another approach can be to use TLS and client certificates. You can define a CLNTCONN and SVRCONN definition that specifies the use of client certificates. By ensuring that only your IBM MQ administrators have access to these client certificates, you can control access. A typical use of a

server-side exit is to map the TLS certificate to a user ID and set the MCAUSER with the authenticated ID.

Without SSLPEER, any certificate that a trusted signer issues is accepted. Set SSLPEER on the channel and delete unneeded certificate authorities from the truststore. If you use IBM Key Management to create a keystore, it contains well-known CA certificates. Delete the well-known CA certificates if you are not planning to use them.

Authorizing groups and principals

- On UNIX and Linux
 - You can authorize access to groups
 - If enabled on the queue manager, you can authorize access to principals (users)
- On Windows
 - You can authorize access to groups and principals (users)
 - Fully qualify principals with the domain to avoid ambiguity

Example: `authorize -p user@domain`
- Decide what actions are necessary to apply the appropriate level of security for the groups or principals to the IBM MQ components
 - Who needs access to the queue manager?
 - Do these users or groups need full administrative access or partial administrative access on a subset of queue manager resources?
 - Do these users or groups need read-only access to all queue manager resources?

Figure 10-27. Authorizing groups and principals

You can authorize access to IBM MQ resources by groups or principals. Before you configure authorization, determine whether access should be limited to a user or a group.



Note

The ability to authorize principals on UNIX or Linux must be explicitly configured on the queue manager.

To enable principal authorization when you create the queue manager, add the `-oa user` option to the `crtmqm` command. If you do not set this parameter, group authorization is used.

You can change the authorization model later by setting the **SecurityPolicy** parameter in the **Service** stanza of the `mq.ini` file.

Interactive users with OAM

- User IDs and authorization policies that are associated with interactive users can be dynamic
 - Ongoing administration can be extensive and expensive
- Do not routinely grant CREATE, DELETE, CHANGE, CONTROL, and CONTROLX
 - These authorizations confer administrative authority to create, delete, or change objects and to start or stop channels or services

Figure 10-28. Interactive users with OAM

It might be better to enable authorizations for groups than by principals to reduce the amount of administration. It is important to ensure that all users in a group should have access to resources.

Use care when giving principals or groups access to create, delete, change, or control IBM MQ resources.

Considerations for applications with OAM

- Applications are generally fairly static and relatively small in number
- Do not make authorization too restrictive with OAM
 - Understand the access the application really requires
- Most applications need only PUT or GET/BROWSE and sometimes INQUIRE
- JMS applications always need INQUIRE
- Assign applications their own specific exception queue
 - For dead messages, for example
 - However, they are still allowed PUT access (but not GET access) to the dead-letter queue

Figure 10-29. Considerations for applications with OAM

This figure lists some considerations when configuring IBM MQ resource access for applications.

OAM and queue manager to queue manager

- Set MCAUSER on an inbound MCA channel (RCVR, RQSTR, or CLUSRCVR) to a low-privileged ID to restrict that channel so it can access only specific queues
- In addition to **+put** and **+inq**, the ID needs **+setall** authority
 - An administrative right, but the channel agent is trusted code
 - Do not allow other users in this group and disable the account
- Do not authorize access to the command queue
- Consider restricting access to the dead-letter queue and to any transmission queues
- Business requirements might determine whether you need to use TLS between queue managers
 - Consider many factors such as physical security, network interception points, and disaster recovery requirements

Authenticating channels and connections

© Copyright IBM Corporation 2020

Figure 10-30. OAM and queue manager to queue manager

This figure lists some tips for configuring access for queue manager to queue manager authorization.

If TLS is not used, some other method of authentication, such as using security exits, is required. Physical security and static IP addresses make IP filtering potentially useful for these connections.

A note about transmission queues

- Authorization to access a transmission queue gives the user PUT authority to any queue that is accessible from that transmission queue
- For the cluster transmission queue, the user can PUT to any queue in the cluster, whether it is advertised to the cluster or not
 - Restrict access to a transmission queue at the receiver side with a low-privileged MCAUSER user ID
 - Restrict access to a transmission queue at the sender side with aliases or by using remote queue definitions

Figure 10-31. A note about transmission queues

Remember to restrict access to the transmission queues. Authorization to a transmission queue gives the user PUT authority to any queue that is accessible from the transmission queue.

Allowing a PUT to a transmission queue can open the network to security attacks.

PUTAUT: Put Authority for channels

- When PUTAUT(CTX) is enabled, the alternative user ID is used from the context information that is associated with the message
- PUTAUT(CTX) requires all queue managers to be in the same authentication domain
 - Might not be practical to use PUT authority in a mixed operating system environment
 - Therefore seldom used
- Alternative to authorizing the transmission queue requires a remote queue or alias queue for every authorized remote destination
 - OAM privileges are then granted on the locally defined remote queue and alias queue objects

Figure 10-32. PUTAUT: Put Authority for channels

The administrator must also consider whether to allow PUT authority for channels.

Security guidelines

- When sending and receiving messages with an organization external to your company:
 - Use a dedicated gateway queue manager
 - Do not have channels that are connected directly to internal queue managers
- Do not allow an IBM MQ client to come through a firewall to an internal queue manager
- Do not allow IBM MQ clustering through a firewall
- Run all IBM MQ layered services such as dead-letter queue handler, trigger monitor, and HTTP bridge as low-privileged applications
 - Do not put the user accounts to run IBM MQ layered services in the “mqm” group
 - If you start services as IBM MQ defined service, use a mechanism that starts them with a low-privileged ID

Figure 10-33. Security guidelines

This figure lists some guidelines for helping to keep your IBM MQ network secure.

10.3. Connection authentication

Connection authentication

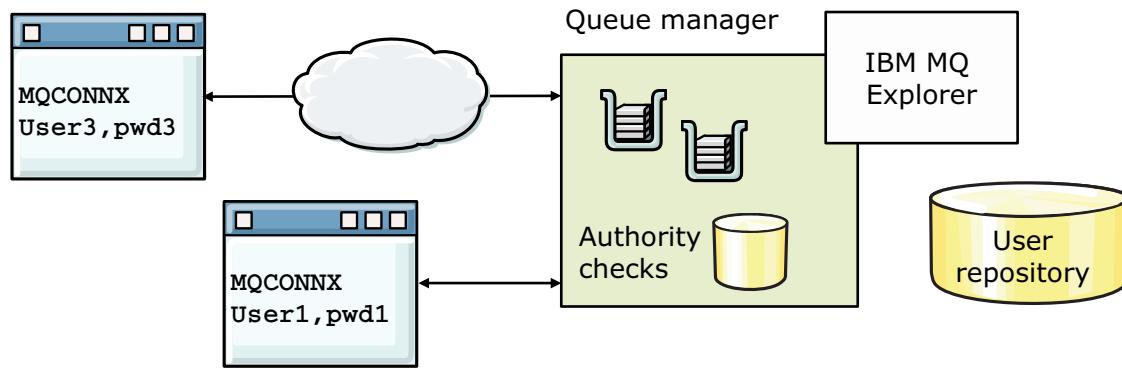
Authenticating channels and connections

© Copyright IBM Corporation 2020

Figure 10-34. Connection authentication

Connection authentication

- Ability for an application to provide a user ID and password
 - Client
 - Local bindings
- Enabled by default in new IBM MQ V9 queue managers
- Some configuration is required in the queue manager
- Requires a user repository that knows whether the user ID and password are a valid combination



Authenticating channels and connections

© Copyright IBM Corporation 2020

Figure 10-35. Connection authentication

This figure shows the general IBM MQ network that is referenced in this topic for connection authentication.

On the left side of the network are applications that are making connections. The top application is connecting as a client. The bottom application is connecting by using local bindings. These applications might be using different APIs to connect to the queue manager, but they all can provide a user ID and a password. The user ID that the application is running under might be different from the user ID provided by the application with its password. The user ID of each application is shown on the figure.

In the middle of the network is the queue manager. The queue manager is responsible for managing resources and the checking of authority to those resources. There are many resources in IBM MQ that an application might require authority to, but this example uses a queue that must be opened for output.

On the right of the figure is a representation of a user repository that contains the user IDs and passwords.

Enabling connection authentication on a queue manager

- Set the CONNAUTH attribute on the queue manager to the name of an authentication information object
- Define the authentication information object with the DEFINE AUTHINFO command
- Use the REFRESH SECURITY TYPE(CONNAUTH) command to refresh the cached view of the configuration for connection authentication

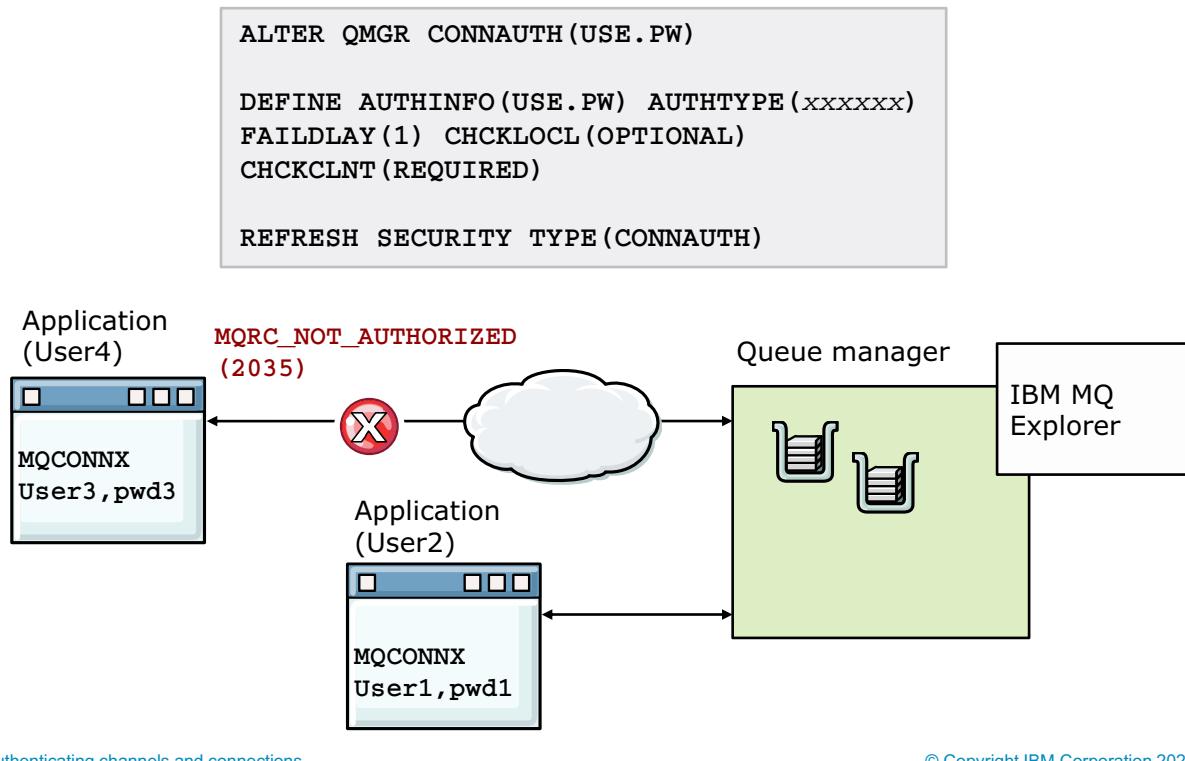
Figure 10-36. Enabling connection authentication on a queue manager

The first step is to enable connection authentication on the queue manager.

On the queue manager definition, the CONNAUTH (connection authentication) attribute points to an authentication information object. The authentication information object is defined by using the DEFINE AUTHINFO command.

After configuration is complete, the final step is to refresh the security cache for connection authentication.

Connection authentication configuration



Authenticating channels and connections

© Copyright IBM Corporation 2020

Figure 10-37. Connection authentication configuration

The figure shows the three main commands that configure connection authentication.

- The ALTER QMGR command modifies the queue manager to refer to a connection authentication information object that is named USE.PW.
- The DEFINE AUTHINFO command defines the USE.PW connection authentication information object.
- The REFRESH command updates the connection authentication and the cache so that the queue manager recognizes the changes.

The AUTHINFO command contains two parameters for user ID and password checking on a connection: CHCKLOCL (check local connections) and CHCKCLNT (check client connections).

Both of these parameters have the same set of attributes, which provides for a strictness of checking. You can set these fields to the following values:

- NONE disables connection authentication. When password checking is turned off with the NONE option, invalid passwords are not detected.
- OPTIONAL validates a user ID and password when the application provides them but a user ID and password are not mandatory.
- REQUIRED requires that all applications provide a user ID and password.

- REQDADM requires a valid user ID and password from a privileged user and is optional for nonprivileged users.

Any application that does not supply a user ID and password when required to, or supplies an incorrect combination even when it is optional receives an MQRC_NOT_AUTHORIZED error.

Any failed authentications are held for the number of seconds that are specified in FAILDLAY field before the error is returned to the application. This attribute provides some protection against a “busy loop” from an application that is repeatedly connecting.

In the example, the local application connection succeeds because CHCKLOCL is set to OPTIONAL.

Connection authentication error notification

- Application: MQRC_NOT_AUTHORIZED (2035)
- Administrator: Error message
- Monitoring tool
 - “Not Authorized” event message (Type 1 – Connect)
 - Connection not authorized: MQRQ_CONN_NOT_AUTHORIZED
 - User ID and password not authorized: MQRQ_CSP_NOT_AUTHORIZED
 - Field in the “connect” event: MQCACF_CSP_USER_IDENTIFIER
- To enable authorization events: **ALTER QMGR AUTHOREV (ENABLED)**

Figure 10-38. Connection authentication error notification

When an application provides a user ID and password that fails the authentication check, the application receives the standard IBM MQ security error, 2035 – MQRC_NOT_AUTHORIZED. The administrator can view this error in the error log and can see that the application was rejected because the user ID and password check failed.

A monitoring tool can also be notified of this failure by checking the SYSTEM.ADMIN.QMGR.EVENT queue if authority events are enabled. The generated event is a *Not Authorized* event, which is a Type 1, Connect event. This type of event includes the MQCSP field that indicates that a user ID is provided so two user IDs are provided in the message: the user ID that the application is running as and the user ID that the application presented for user ID and password checking. The event message does not provide the password.

To enable authority events, enter the MQSC command **ALTER QMGR AUTHOREV (ENABLED)**

Connection authentication and authorization

- When **ADOPTCTX = NO**, user ID provided by the application is authenticated but authorization uses the user ID that the application is running under
- When **ADOPTCTX = YES**, user ID that the application provides in the MQSCP structure of the MQCONN call is used for authentication
 - To adopt another security context, also set **ChauthEarlyAdopt** to Y in **Channels** stanza of **qm.ini**

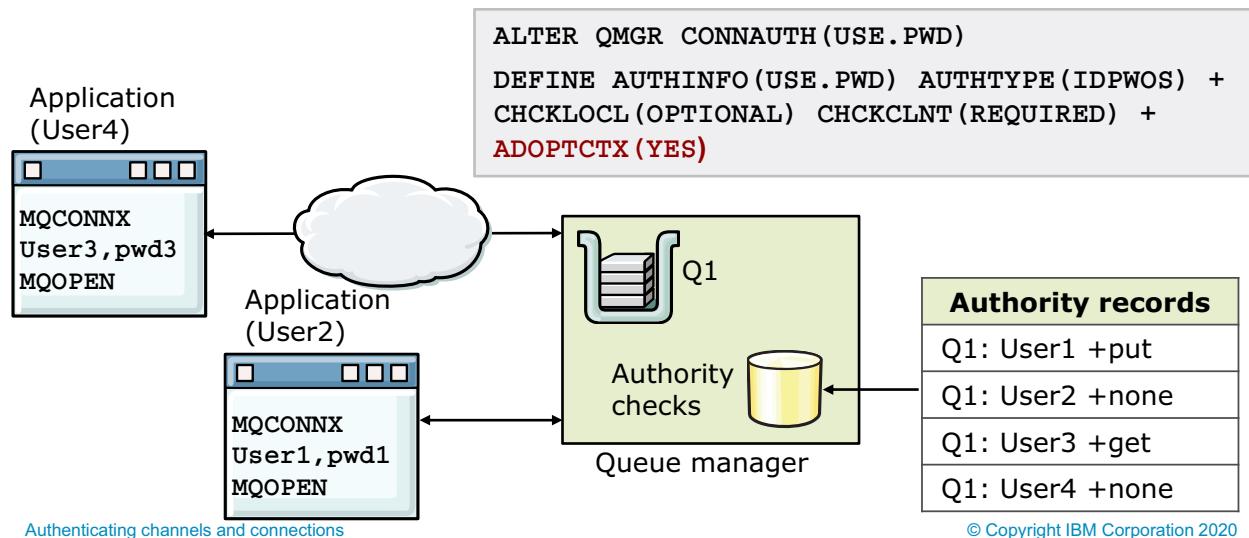


Figure 10-39. Connection authentication and authorization

On the previous figure, you saw how you can configure the queue manager to mandate user IDs and passwords from specific applications. You should also recognize that the user ID that the application is running under might not be the same user ID that the application presents.

So what is the relationship of these user IDs to the ones that are used for the authorization checks when the application, for example, opens a queue for output? The ADOPTCTX attribute on the authentication information object controls the relationship.

If you set ADOPTCTX(NO), the applications provide a user ID and password for the purposes of authenticating them at connection time, but continue to use the user ID that they are running under for authorization checks. The option might be useful when upgrading, or beneficial for client connections because authorization checks are done by using an assigned MCAUSER based on IP address or SSL/TLS certificate information.

The MQCSP structure on an MQCONN call enables the authorization service to authenticate a user ID and password. When you use the ADOPTCTX(YES) parameter on an authentication information object, the security context is set as the user ID that is presented in the MQCSP structure, when validated by a password. In this case, another security context cannot be adopted, unless you set the **ChauthEarlyAdopt** to Y in the **Channels** stanza of **qm.ini** file.

1+1=2 Example

The default authentication information object is set to ADOPTCTX(YES), and the user “fred” is logged in. The following two CHLAUTH rules are configured:

```
SET CHLAUTH('MY.CHLAUTH') TYPE(ADDRESSMAP) DESCRIPTOR('Block all access by default') ADDRESS('*') USERSRC(NOACCESS) ACTION(REPLACE)
SET CHLAUTH('MY.CHLAUTH') TYPE(USERMAP) DESCRIPTOR('Allow user bob and force CONNAUTH') CLNTUSER('bob') CHCKCLNT(REQUIRED) USERSRC(CHANNEL)
```

The following command is entered, with the intention of authenticating the command as the adopted security context of the user “bob”:

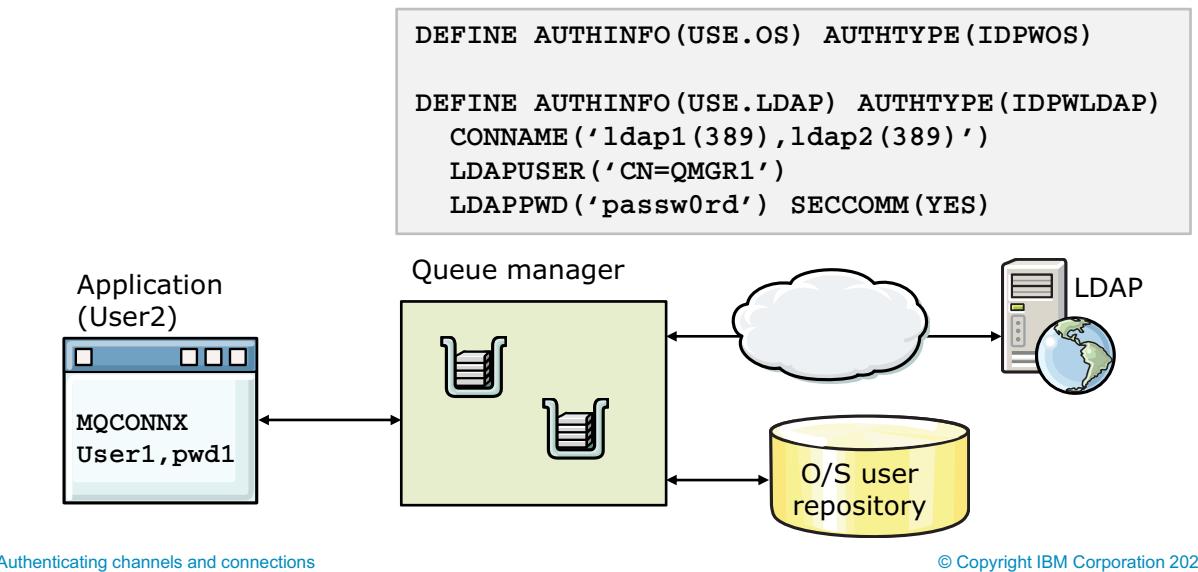
```
runmqsc -c -u bob QMGR
```

The queue manager uses the security context of “fred”, not “bob”, and the connection fails.

To use the security context of “bob”, the **ChlauthEarlyAdopt** parameter in the **Channels** stanza of the `qm.ini` file must be set to **Y**.

Connection authentication user repositories

- IDPWOS indicates that the queue manager uses the local operating system to authenticate the user ID and password
- IDPWLDAP indicates that the queue manager uses an LDAP server to authenticate the user ID and password



Authenticating channels and connections

© Copyright IBM Corporation 2020

Figure 10-40. Connection authentication user repositories

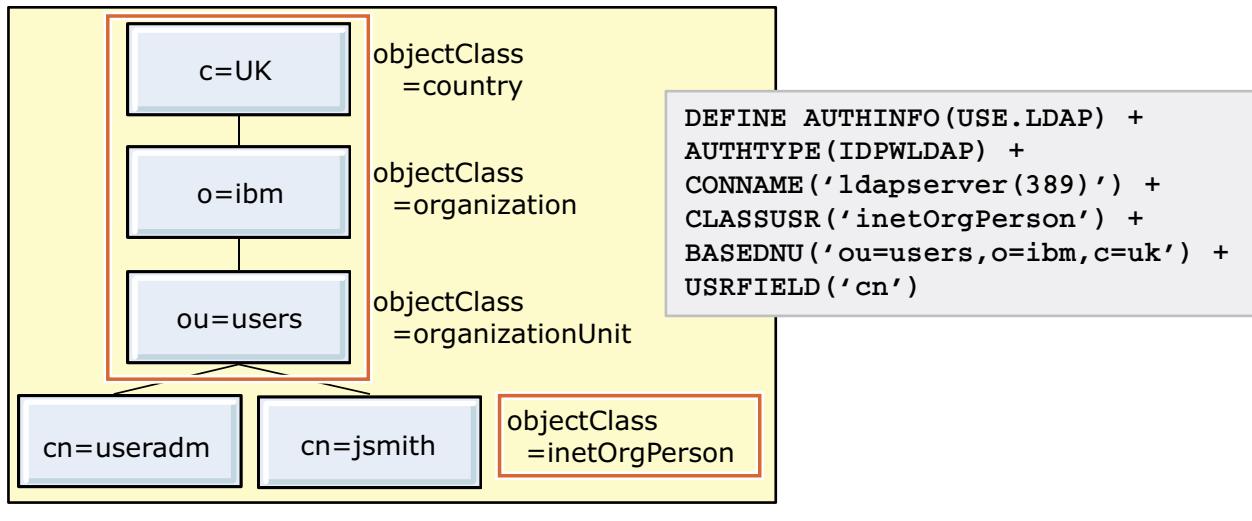
This figure describes the object types of the authentication information object.

The authentication information objects are defined on the AUTHTYPE field of the DEFINE AUTHINFO command.

- AUTHTYPE(IDPWOS) indicates that the queue manager uses the local operating system to authenticate the user ID and password.
- AUTHTYPE(IDPWLDAP) indicates that the queue manager uses an LDAP server to authenticate the user ID and password.

Only one type of authentication can be chosen for the queue manager.

LDAP user repository



Application provides	USRFLD	BASEDNU
cn=useradm,ou=users,o=ibm,c=uk		
cn=useradm		Adds ou=users,o=ibm,c=uk
useradm	Adds cn=	Adds ou=users,o=ibm,c=uk

Authenticating channels and connections

© Copyright IBM Corporation 2020

Figure 10-41. LDAP user repository

When using an LDAP user repository, the queue manager needs to know the location of the LDAP, and other information.

User ID records defined in an LDAP server use a hierarchical structure to uniquely identify them. An application can connect to the queue manager and present its user ID as the fully-qualified hierarchical user ID. However, this fully-qualified hierarchical user ID contains much information, and it would be simpler to configure the queue manager to a more generic information. For example, it would be simpler to have the queue manager assume that all user IDs that are presented are found in a specific area of the LDAP server and then add the qualification. In the DEFINE AUTHINFO command, the BASEDNU attribute identifies the area in the LDAP hierarchy where all the user IDs are found. So, the queue manager adds the BASEDNU value to the user ID presented by an application to fully qualify it before looking it up in the LDAP server.

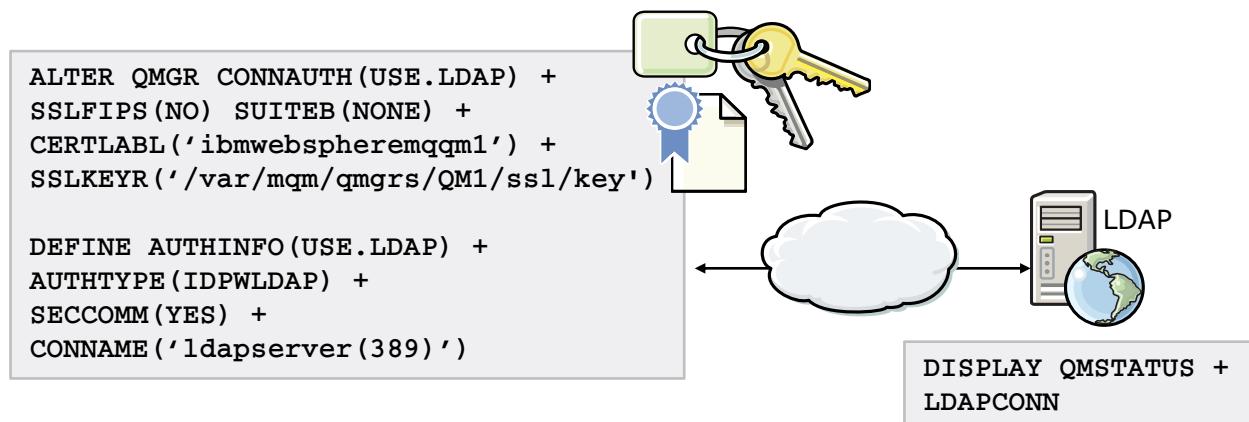
Also, an application might present the user ID but not the LDAP attribute name, such as "CN=". If the user ID provided by an application for authentication does not contain a qualifier for the field in the LDAP user record, the USRFIELD field in the DEFINE AUTHINFO command identifies the field in the LDAP user record that is used to interpret the provided user ID.

Any user ID that is presented to a queue manager without an equals sign (=) has the attribute and the equal sign added as a prefix and the BASEDNU value added as a suffix before looking it up in the LDAP server. This process can be beneficial when moving from operating system user IDs to

LDAP user IDs because the application might present the same string in both cases, which avoids any change to the application.

Secure connection to an LDAP server

- Enable SECCOMM on AUTHINFO command for TLS communication
- Queue manager attributes SSLFIPS and SUITEB restrict the set of cipher specs
- OCSP servers that are named in the AIA certificate extensions check certificate revocation
- Use DISPLAY QMSTATUS command to checks status of the connection from the queue manager to the LDAP server



Authenticating channels and connections

© Copyright IBM Corporation 2020

Figure 10-42. Secure connection to an LDAP server

You might want to secure your connection between the LDAP server and the queue manager. Unlike channels, there is no SSLCIPH parameter to enable the use of SSL/TLS for the communication with the LDAP server. So, in this case, IBM MQ acts as a client to the LDAP server. This topology requires that much of the configuration is done on the LDAP server.

Some parameters in IBM MQ (shown on the figure) define the connection between the queue manager and the LDAP server.

- The SECCOMM attribute on the DEFINE AUTHINFO command enables SSL/TLS communication.
- The queue manager attributes SSLFIPS and SUITEB restrict the set of cipher specifications that are chosen.

The NSA Suite B standard restricts the set of enabled cryptographic algorithms to provide an assured level of security. IBM MQ can be configured to operate in compliance with the NSA Suite B standard on Windows, UNIX, and Linux.

The Federal Information Processing Standards (FIPS) Cryptomodule Validation Program of the US National Institute of Standards and Technology and the cryptographic functions can be used in IBM MQ on TLS channels, for Windows, UNIX, Linux, and z/OS.

- The certificate that identifies the queue manager to the LDAP server is the queue manager certificate that the string `ibmwebspheremq<qmgr-name>` or the CERTLBL attribute identifies.
- Certificate revocation is checked by using the OCSP servers that are named in the AIA certificate extensions. This option can be enabled setting the SSL stanza attribute **OCSPCheckExtensions** in the `qm.ini` file.

The connection to an LDAP server is made as a network connection. The status of this connection from the queue manager to the LDAP server is shown by entering the DISPLAY QMSTATUS command.

Connection authentication and authorization: LDAP

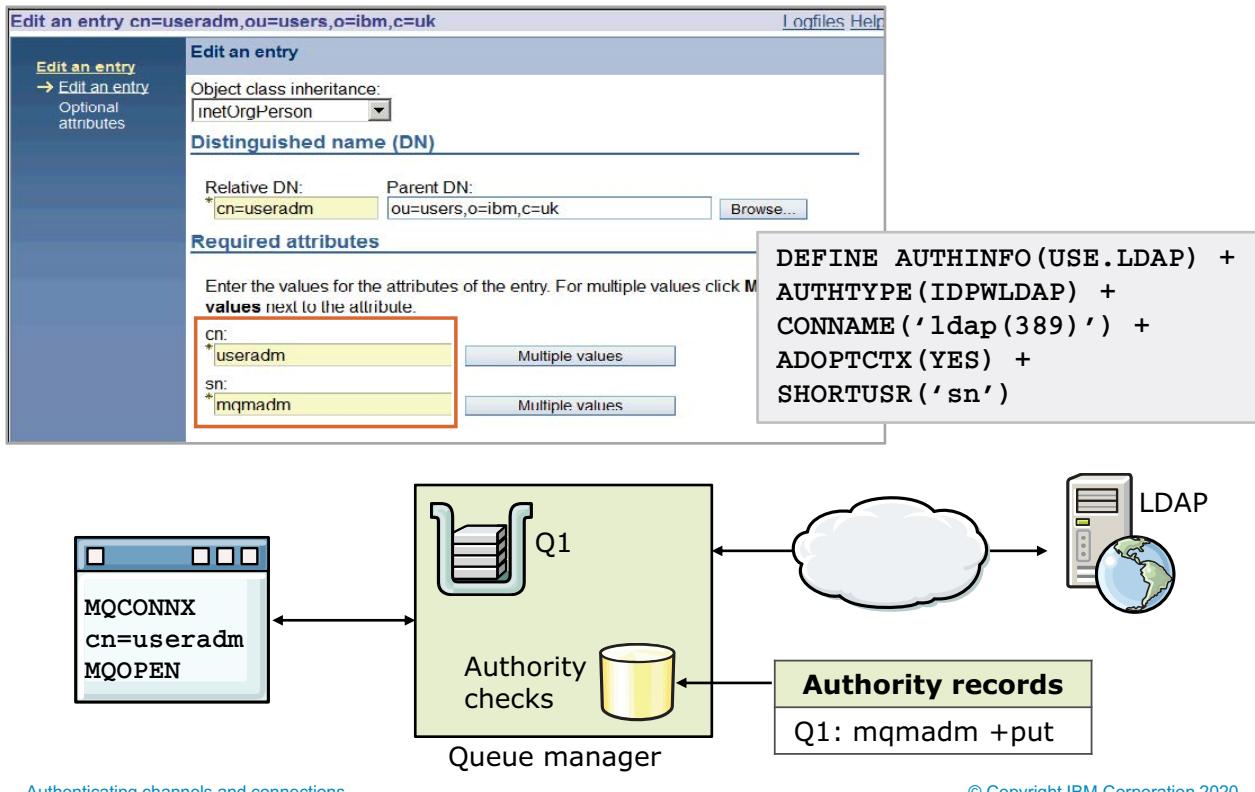


Figure 10-43. Connection authentication and authorization: LDAP

Previously in this topic, you learned about using the ADOPTCTX attribute to adopt the authenticated user ID as the context for the connection. So how does this work if you are using LDAP as the user repository but the operating system user ID is used for authorization?

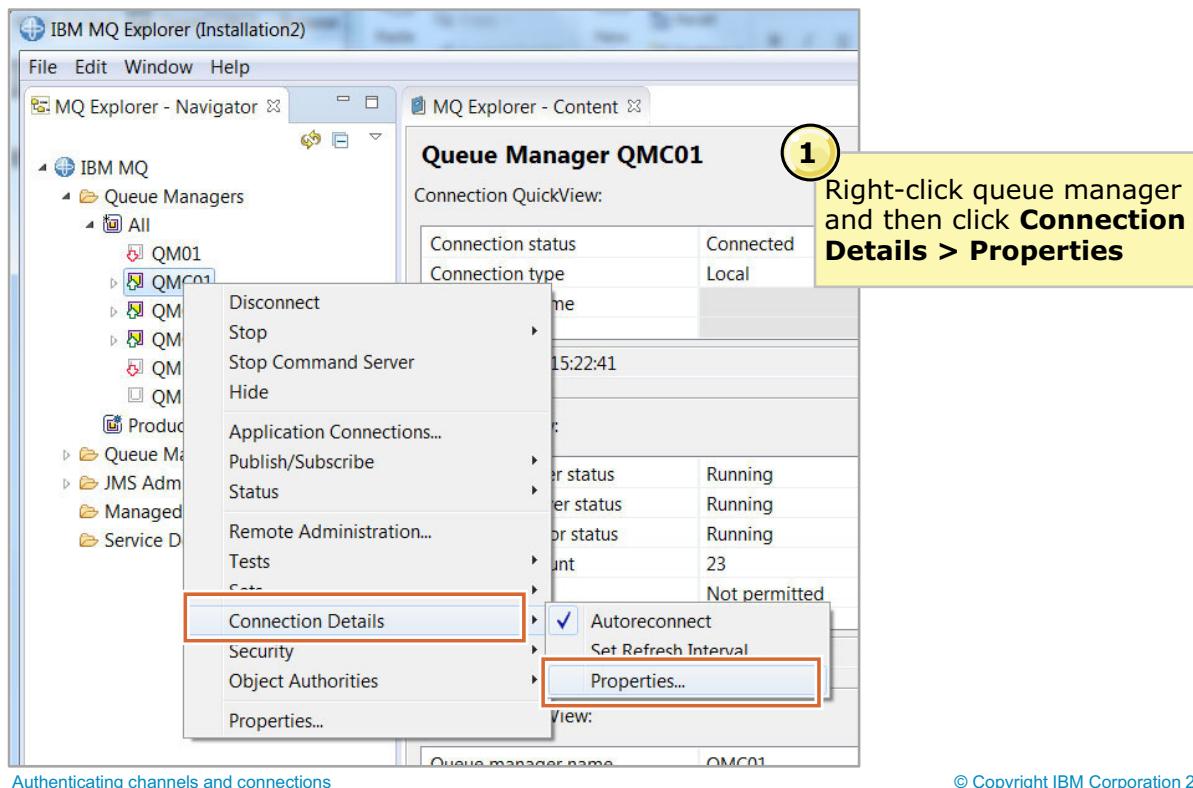
You need to get a user to represent the LDAP user that was presented, as an operating system user ID. This user can be found from the LDAP user record. The user can be any field that is defined in the user record, such as the short name field (**sn=**), which is a mandatory part of the definition of the **inetOrgPerson** class. Alternatively, the user might be in the user ID (**uid=**) field.

The queue manager then uses the LDAP user information to determine the operating system user ID to use as the context for this connection.

You configure the LDAP user by using the SHORTUSR option on the DEFINE AUTHINFO command.

In the example in the figure, the application connects with **cn=useradm**. The SHORTUSR option identifies **sn** as the location for the user name. The LDAP server looks up the record with **cn=useradm** and returns the short name of **mqmadm**. This user name is the user ID that is used for authority checks by the queue manager.

User ID and password for IBM MQ Explorer (1 of 2)



Authenticating channels and connections

© Copyright IBM Corporation 2020

Figure 10-44. User ID and password for IBM MQ Explorer (1 of 2)

IBM MQ Explorer can be configured with a user ID and password that it uses to connect to a local or client connection to a queue manager.

1. Right-click the queue manager in the **MQ Explorer - Navigator** view and then click **Connection Details > Properties**.

[<graphic>]

1. On the **Userid** page, click **Enable user identification**.
 2. Enter the user ID.
 3. IBM MQ Explorer has a password cache that must be enabled to use passwords. If saved passwords are enabled, enter the password.
- If you did enable saved passwords, a link appears on the **Userid** page to help you set up the cache.

User ID and password for IBM MQ Explorer (2 of 2)



Authenticating channels and connections

© Copyright IBM Corporation 2020

Figure 10-45. User ID and password for IBM MQ Explorer (2 of 2)

User IDs and passwords in IBM MQ programs

- Some commands allow a user ID and prompt for a password: `dmpmqcfg`, `dspmqrte`, `runmqtrm`, `runmqtmc`, `runmqsc`, and `runmqd1q`
- Some IBM MQ sample programs allow a user ID and prompt for a password
 - Use the environment variable `MQSAMP_USER_ID` for the user ID
Example: `set MQSAMP_USER_ID=j smith`

Sample name	Description
<code>amqspput / amqspputc</code>	Put one or more messages to a queue
<code>amqsget / amsgetc</code>	Get messages from a queue
<code>amqsbcg / amqsbcgc</code>	Read and output the message descriptor fields, any other message properties, and the message content of all the messages on a queue

Authenticating channels and connections

© Copyright IBM Corporation 2020

Figure 10-46. User IDs and passwords in IBM MQ programs

Some of the IBM MQ sample programs allow a user ID and password so that you can test your applications with authentication enabled. Set the sample program user ID by using the `MQSAMP_USER_ID` environment variable. When you run the sample program, it prompts you for a password.

For a complete list of the sample programs that support this feature, see the IBM Knowledge Center.

Connection authentication summary

- Application can provide a user ID and password in the MQCONN call or by using a security exit
- Queue manager checks password against operating system or LDAP

Examples:

```
ALTER QMGR CONNAUTH('CHECK.PWD')
DEFINE AUTHINFO('CHECK.PWD')
AUTHTYPE(IDPWOS|IDPWLDAP)
CHCKLOCL(NONE|OPTIONAL|REQUIRED|REQDADM)
CHCKCLNT(NONE|OPTIONAL|REQUIRED|REQDADM)
ADOPTCTX(YES)
plus LDAP attributes
```

- Refresh security to update connection authentication

```
REFRESH SECURITY TYPE(CONNAUTH)
```

Figure 10-47. Connection authentication summary

This figure summarizes connection authentication.

10.4. Channel authentication

Channel authentication

Authenticating channels and connections

© Copyright IBM Corporation 2020

Figure 10-48. Channel authentication

Channel authentication rules

- Set rules to control how inbound connections are treated
 - Inbound clients
 - Inbound queue manager to queue manager channels
 - Other rogue connections that are causing FDCs
- Rules can be set to:
 - Allow a connection
 - Allow a connection and assign an MCAUSER
 - Block a connection
 - Ban privileged access
 - Provide multiple positive or negative TLS peer name matching
- Rules can use any of the following identifying characteristics of the inbound connection
 - IP address or hostname
 - TLS subject's Distinguished Name
 - Client asserted user ID
 - Remote queue manager name

[Authenticating channels and connections](#)

© Copyright IBM Corporation 2020

Figure 10-49. Channel authentication rules

Channel authentication records allow you to define rules for handling inbound connections to the queue manager. Inbound connections include client channels and queue manager to queue manager channels.

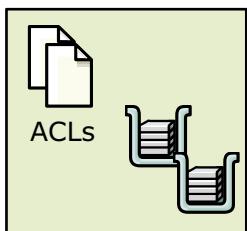
These rules can specify whether connections are allowed or blocked. If the connection is allowed, the rules can provide a user ID for the channel or use the channel user ID from the client or a user ID that was defined on the channel definition.

These rules can be used to do the following actions:

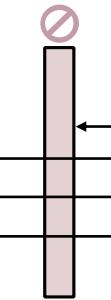
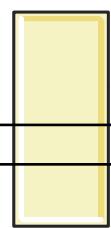
- Set up appropriate identities for channels to use when they run against the queue manager
- Block unwanted connections
- Ban privileged users

Channel access blocking points

Queue manager



Channel blocking/mapping



Channel blocking/mapping

- Rules to block channels, map channels to MCAUSER, and allow channels
- Runs before security exit
- Final check for user ID
 - After security exit runs and final MCAUSER is assigned
 - Ban privileged users with *MQADMIN

Listener blocking

- Not a replacement for an IP firewall
- Blocked before any data is read from the socket
- Simplistic avoidance of “denial of service” attack
- “Ping” attacks, if blocked, do not raise an alert

[Authenticating channels and connections](#)

© Copyright IBM Corporation 2020

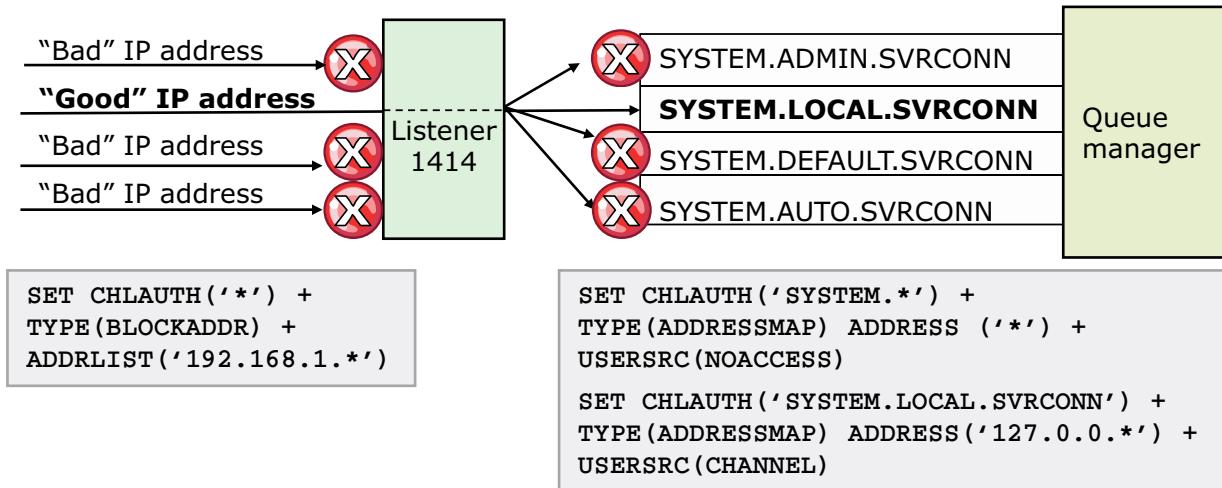
Figure 10-50. Channel access blocking points

This figure shows blocking points that an inbound connection must get through before it can access a queue: the IP firewall, the listener, and the channel.

The IP firewall is included in this set of blocking points. Do not ignore it as a blocking point. IBM MQ security features do not supersede the functions that the IP firewall provides.

The users that are considered as privileged users is different depending on the operating system on which the queue manager runs. There is a special value ‘*MQADMIN’ that is defined to mean “any user that is privileged on this operating system”. This special value can be used in the rules that check against the final user ID that the channel uses in TYPE(USERLIST) rules to ban any connection that is about to run as a privileged user. This application identifies any blank user IDs that come from a client, for example.

Channel authentication example



- Per-channel rules match the least-specific to most-specific
- Global blocking rules occur at the listener before the channel name is known and take precedence over per-channel rules

Authenticating channels and connections

© Copyright IBM Corporation 2020

Figure 10-51. Channel authentication example

The figure shows how you can filter connections by using channel authentication rules.

The SET CHLAUTH command on the left in the figure specifies TYPE(BLOCKADDR). It is a global blocking rule that occurs at the listener before the channel name is known. The channel name in the CHLAUTH attribute must be '*' when the TYPE is BLOCKADDR. The address list (ADDRLIST) is the address from which to refuse 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, the rule blocks all IP addresses that start with 192.168.1.

The per-channel filtering rules on the right have the TYPE attribute set to ADDRESSMAP. These rules map IP addresses to MCAUSER values. An ADDRESS must accompany the ADDRESSMAP parameter.

The first rule 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 rule do not have access to the queue manager and the channel ends immediately.

The second rule gives the local host (127.0.0.*) access to the channel that IBM MQ Explorer uses (SYSTEM.LOCAL.SVRCONN) because the USERSRC parameter is set to CHANNEL. The CHANNEL option means that inbound connections that match this rule use the flowed user ID or any user ID that is defined on the channel object in the MCAUSER field.

Precedence order

- Rules that use channel authentication records follow a precedence order so that it is clear which rule is used when an inbound connection matches multiple rules

Order	Identity mechanism	Notes
0	Channel name	
1	TLS Distinguished Name	
2=	Client asserted user ID	Several different user IDs can be running on the same IP address.
2=	Queue manager name	Several different queue managers can be running on the same IP address.
4	IP address	
5	Hostname	One IP address can have multiple hostnames.

Figure 10-52. Precedence order

Channel authentication rules follow a precedence order so that it is clear which rule is used when an inbound connection can match multiple rules.

A channel authentication rule that explicitly matches the channel name takes priority over a channel authentication record that matches the channel name by using a wildcard.

A channel authentication rule that uses a TLS DN takes priority over a rule that uses a user ID, queue manager name, IP address, or host name.

A channel authentication rule that uses a user ID or queue manager name takes priority over a rule that uses an IP address or host name.

Host names are less specific than an IP address because a single IP address can have multiple host names. If you have an IP address rule and a host name rule that can both match an inbound connection, then the IP address rule is used because it is considered to be more specific.

Precedence order example

```
DISPLAY CHLAUTH(APPL1.*)
CHLAUTH(APPL1.*)
TYPE(SSLPEERMAP)
SSLPEER('O="IBM UK"') MCAUSER(UKUSER)

CHLAUTH(APPL1.*)
TYPE(USERMAP)
CLNTUSER('jsmith') MCAUSER(SMITH)

CHLAUTH(APPL1.*)
TYPE(ADDRESSMAP)
ADDRESS('9.180.165.163') MCAUSER(SMITH)

CHLAUTH(APPL1.*)
TYPE(ADDRESSMAP)
ADDRESS('*.ibm.com') MCAUSER(IBMUSER)
```

Example input

Chl: APPL1.SVRCONN
 DN: CN=SMITH.O=IBM UK
 UID: jsmith
 IP: 9.180.165.163

Figure 10-53. Precedence order example

In this example, you can see that example input record matches more than one rule.

The example input matches all rules at the channel level (APPL1.*) so the next order of precedence is the DN, which takes precedence over the user ID and IP address. So in this example, the first rule applies and the MCAUSER is set to UKUSER.

Using IP addresses in channel authentication rules

- Initial listener blocking list
 - List, range, or pattern of IP addresses
 - Does not replace IP firewall
 - Should be used sparingly
- Channel-based blocking of single IP address, range, or pattern
- Channel allowed, based on single IP address, range, or pattern
- Further qualified rule that includes IP address on another rule type
 - Works with SSLPEER, QMNAME, and CLNTUSER

```
SET CHLAUTH('*') TYPE(BLOCKADDR) +
ADDRLIST('9.20.*', '192.168.2.10')
```

```
SET CHLAUTH('APPL1.*') +
TYPE(ADDRESSMAP) ADDRESS('9.20.*') +
USERSRC(NOACCESS)
```

```
SET CHLAUTH('.*.SVRCONN') +
TYPE(ADDRESSMAP) +
ADDRESS('9.20-21.*') MCAUSER(HUSER)
```

```
SET CHLAUTH('*') TYPE(SSLPEERMAP) +
SSLPEER('CN="Jon Smith"') +
ADDRESS('9.20.*') MCAUSER(JSMITH)
```

Figure 10-54. Using IP addresses in channel authentication rules

IP addresses can be used in channel authentication rules in four different ways.

The initial check that the listener makes for banned IP addresses is created with TYPE(BLOCKADDR) in the rule (the first example in the figure). Use this type of rule sparingly. It is intended as an IBM MQ administrator control to temporarily configure banned IP addresses until the IP firewall is updated to restrict access to the network.

After the initial listener check, the mapping rules are applied. You can ban a specific IP address from a channel by using USERSRC(NOACCESS) on a mapping rule (shown in the second example).

You can also map a channel to use a particular MCAUSER or to flow through its client-side credentials if it comes from a particular IP address (shown in the third example).

Finally, IP address restrictors such as SSLPEER, QMNAME, and CLNTUSER can be added to any of the other types of mapping rules (shown in the fourth example).

Using hostnames in channel authentication rules

- Initial listener blocking list
 - Hostnames are not allowed
- Channel-based blocking of hostnames with single hostname or pattern
- Channel allowed, based on hostnames with single hostname or pattern
- Further qualified rule that includes hostname on another rule type
 - Works with SSLPEER, QMNAME, and CLNTUSER

SET CHLAUTH('') TYPE(BLOCKADDR) +
ADDRLIST()

SET CHLAUTH('APPL1.*') +
TYPE(ADDRESSMAP) +
ADDRESS('*.**ibm.com**') +
USERSRC(NOACCESS)

SET CHLAUTH('*.**SVRCONN**') +
TYPE(ADDRESSMAP) +
ADDRESS('mach123.**ibm.com**') +
MCAUSER(HUSER)

SET CHLAUTH('*') TYPE(SSLPEERMAP) +
SSLPEER('CN="Jon Smith"') +
ADDRESS('s*.**ibm.***') MCAUSER(JSMITH)

Figure 10-55. Using hostnames in channel authentication rules

You can substitute host names for IP addresses in most channel authentication rules. The exception is on rules with TYPE(BLOCKADDR). This type of rules allows IP addresses only.

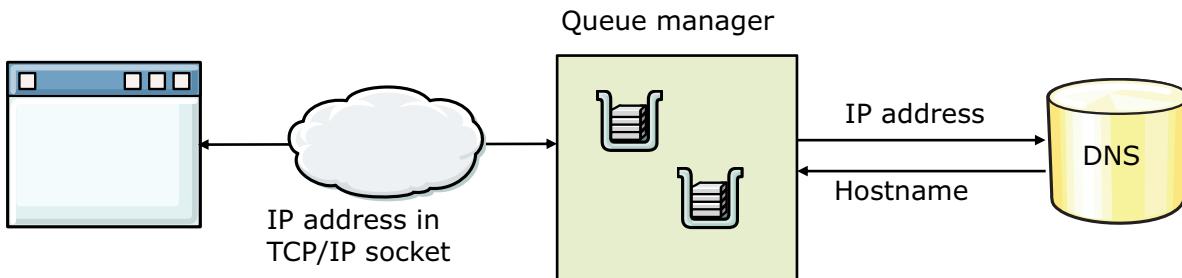
If you want to block specific hosts by using channel authentication, use a TYPE(ADDRESSMAP) rule with host names and USERSRC(NOACCESS), as shown in the second example.

As shown in the other examples, mapping rules, and address restrictors also allow host names instead of IP addresses.

Similar to IP addresses and channel names, host names also support pattern matching wildcard characters. Host names use wildcard string matching, such as ***.ibm.com**, which matches any host name that ends with **ibm.com**.

Getting the hostname

- Reverse look up of hostname from Domain Name Server (DNS) based on IP address
- If hostname is used in channel authentication rules:
 - Queue manager must be able to contact DNS
 - DNS must be able to resolve the IP addresses of senders or clients
- To disable reverse lookup, specify: `ALTER QMGR REVDNS (DISABLED)`



Authenticating channels and connections

© Copyright IBM Corporation 2020

Figure 10-56. Getting the hostname

To process channel authentication rules that contain host names, IBM MQ must be able to get the host name that represents the IP address of the socket. The host name is not sent to IBM MQ by the channel or by TCP/IP. IBM MQ gets the IP address from the socket. IBM MQ gets the other attributes that channel authentication records use from the various internal flows across the socket.

To get the host name, IBM MQ interrogates the Domain Name Server (DNS) to determine which host name matches the IP address in the socket. For this action to succeed, the queue manager must be able to use the DNS. The queue name already uses DNS if host names are specified in CONNAME attributes. Also, the DNS must be able to reverse look-up the IP address and find a host name for IBM MQ, which might not be the case.



Important

If host names are used in channel authentication rules, verify that all the sender channel or client application IP addresses are currently available in the DNS so that the IP address in the socket can be matched to a host name.

Some administrators are concerned about the potential security hazards of using host names than others. When REVDNS(ENABLED) is specified on the queue manager, the reverse look-up of the

IP address to retrieve the host name is done only when necessary. If you do not use host names in channel authentication rules, then the only time a reverse look-up is done is when writing an error message that contains that information.

Connection authentication configuration granularity

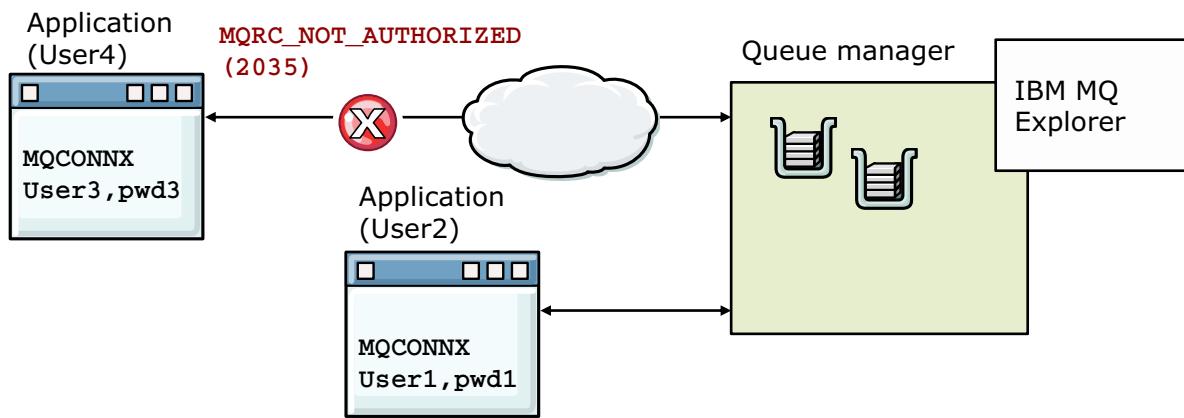
```

DEFINE AUTHINFO(USE.PW) AUTHTYPE(IDPWOS) +
CHCKCLNT(OPTIONAL)

SET CHLAUTH('*') TYPE(ADDRESSMAP) ADDRESS('*') +
USERSRC(CHANNEL) CHCKCLNT(REQUIRED)

SET CHLAUTH('*') TYPE(SSLPEERMAP) +
SSLPEER('CN=*') USERSRC(CHANNEL) +
CHCKCLNT(ASQMGR)

```



Authenticating channels and connections

© Copyright IBM Corporation 2020

Figure 10-57. Connection authentication configuration granularity

In addition to the two fields that enable connection authentication (CHCKCLNT and CHCKLOCL), you can use channel authentication rules (CHLAUTH) to enable authentication for all client and locally bound applications.

In a channel authentication rule, you set the CHCKCLNT field so that more specific configuration can be made for client connections. For example, you can set CHCKCLNT to OPTIONAL on the AUTHINFO command as the default. You can then make authentication more stringent for certain channels by setting CHCKCLNT(REQUIRED) or CHCKCLNT(REQDADM) on the CHLAUTH rule.

By default, channel authentication rules are run with CHCKCLNT(ASQMGR) so that it defaults to using the authorization set at the queue manager level.

Channel authentication is described in more detail later in this unit.

Diagnosing hostname lookup failures

- IBM MQ returns error message when a channel is blocked
 - Message contains the IP address and hostname (if available)
- Use the information in the error message in DISPLAY CHLAUTH MATCH(RUNCHECK) command to get the authentication record that matched the inbound channel at run time

Example message:

```
AMQ9777: Channel was blocked
EXPLANATION:
The inbound channel 'SYSTEM.DEF.SVRCONN' was blocked from address
'smith.ibm.com(9.1.1.1)' because the active values of the channel matched
a record configured with USERSRC(NOACCESS). The active values of the
channel were 'CLNTUSER(hughson) ADDRESS(smith.ibm.com,
smith.hursley.ibm.com)'.
```

Figure 10-58. Diagnosing hostname lookup failures

If it can be resolved, the error message contains the socket IP address and the host name when a channel is blocked and the channel authentication rule contains a host name. The figure includes an example an error message.

You can the DISPLAY CHLAUTH MATCH(RUNCHECK) command to find the authentication record that matched the inbound channel at run time.

Using MATCH(RUNCHECK) example

```
DISPLAY CHLAUTH(SYSTEM.ADMIN.SVRCONN) MATCH(RUNCHECK) +
SSLPEER('CN="John Smith", O="IBM UK"') +
CLNTUSER('jsmith') ADDRESS('9.1.1.1')

returns ===>

CHLAUTH(SYSTEM.ADMIN.SVRCONN)
TYPE(ADDRESSMAP)
ADDRESS('*.*.ibm.com') MCAUSER(JSMITH)
```

Example input

Chl: SYSTEM.ADMIN.SVRCONN
 DN: CN=John Smith.O=IBM UK
 UID: jsmith
 IP: 9.1.1.1

Figure 10-59. Using MATCH(RUNCHECK) example

The DISPLAY CHLAUTH MATCH(RUNCHECK) command returns the channel authentication rules that matched a specific inbound channel at run time. The specific inbound channel is described by providing values that are not generic, such as the channel name or IP address that appears in the error message.

You cannot use the host name in the MATCH(RUNCHECK) command because this command needs the IP address to find the host name. The queue manager then calls the DNS, as it would if the real inbound connection appeared, and finds the host name. The queue manager then runs the match against the rules.

If the queue manager is configured to use REVDNS(DISABLED) and you have channel authentication rules that use host names, a warning message appears. So, the DISPLAY CHLAUTH MATCH(RUNCHECK) command can help you to determine whether the reverse look-up for particular IP addresses is likely to work.

More security options

- Dynamic mapping of MCAUSER
- User ID blocking with CHLAUTH BLOCKUSER
- Filtering by IP address

Authenticating channels and connections

© Copyright IBM Corporation 2020

Figure 10-60. More security options

More security options that you might want to consider when creating your channel authentication rules include the following options:

- Dynamic mapping of the MCAUSER user ID
- Using the CLAUTH BLOCKUSER command to block access based on the user ID
- Filtering by IP address

Dynamic mapping of MCAUSER

- Configuration that is based on various validation criteria
- Allows fewer channels to support same or finer granularity than a security exit
- Uses standard IBM MQ tools for configuration and management

Authenticating channels and connections

© Copyright IBM Corporation 2020

Figure 10-61. Dynamic mapping of MCAUSER

An alternative way of providing a user ID for a channel is to use channel authentication records. With channel authentication records, different connections can use the same channel while using different credentials.

If both the MCAUSER on the channel is set and channel authentication rules apply to the same channel, the channel authentication records take precedence. The MCAUSER on the channel definition is only used if the channel authentication record uses USERSRC(CHANNEL).

You can use a channel authentication record to change the MCAUSER attribute of a server-connection channel, according to the original user ID received from a client. You can also use a channel authentication record to set the MCAUSER attribute of a channel, according to the queue manager from which the channel is connecting.

Dynamic MCAUSER mapping example

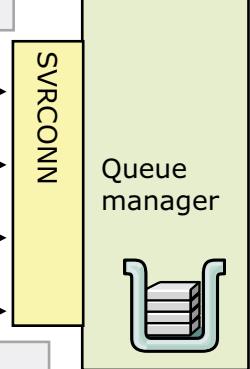
```
SET AUTHREC PROFILE(APP.QUEUE) OBJTYPE(QUEUE) +
... GROUP('appuser') AUTHADD(INQ, DSP, BROWSE, PUT, GET)
... GROUP('appsuppt') AUTHADD(INQ,DSP, BROWSE)
... GROUP('appdev') AUTHADD(INQ, DSP)
... GROUP('appmon') AUTHADD(INQ)
```

MCAUSER=appuser

MCAUSER=appsuppt

MCAUSER=appsdev

MCAUSER=appmon



```
DEF CHL(APP.SVRCONN) CHLTYPE(SVRCONN) +
TRPTYPE(TCP) MCAUSER('nobody')

SET CHLAUTH('APP.SVRCONN') TYPE(SSLPEERMAP) +
... SSLPEER('OU="App Users"') MCAUSER('appuser')
... SSLPEER('OU="App Support"') MCAUSER('appsuppt')
... SSLPEER('OU="App Dev"') MCAUSER('appdev')
... SSLPEER('OU="App Monitor"') MCAUSER('appmon')
```

Authenticating channels and connections

© Copyright IBM Corporation 2020

Figure 10-62. Dynamic MCAUSER mapping example

The figure provides an example of dynamic MCAUSER mapping.

The SET AUTHREC commands set authority records that are associated with a queue profile that is named APP.QUEUE. The authority profile assigns inquire, display, browse, put, or get authority for the groups that are named “appuser”, “appsuppt”, “appdev”, and “appmon” as required. For example, the group that is named “appuser” is given the authority to inquire, display, browse, put, and get. The group that is named “appmon” is given inquire authority only.

The DEFINE CHANNEL command sets the MCAUSER to a nonblank value (“nobody”) so that the MCA uses a user identifier for authorization to access IBM MQ resources.

The SET CHLAUTH commands set the channel authentication record for the APP.SVRCONN channel and creates a peer map. The peer map maps TLS DNs to MCAUSER values. A peer filter that compares the DN of the certificate from the peer queue manager or client at the other end of the channel must accompany the SSLPEERMAP parameter. In the example, the organizational unit (OU) determines the MCAUSER.

Any connection requests that do not match one of the CHLAUTH records are refused. The result is that a single channel definition serves four different security roles for the same application.

The DEF CHL, SET CHLAUTH, and SET AUTHREC definition are all managed by using standard IBM MQ administration tools.

User ID blocking with CHLAUTH BLOCKUSER

- Rules take effect after all other rules and exits are processed and the final value for MCAUSER is determined
- *MQADMIN represents administrative users as defined for the local operating system
 - Simplifies restricting access to administrators
- Blank user ID resolved to the ID of the MCA
- Rules are hierarchical and the most specific one matches
- Can implement a limited deny/allow policy by altering list of blocked names at different levels

Examples:

```
SET CHLAUTH(*) TYPE(BLOCKUSER) USERLIST('nobody, *MQADMIN')
SET CHLAUTH(SYSTEM.ADMIN.*) TYPE(BLOCKUSER) USERLIST('nobody')
```

Figure 10-63. User ID blocking with CHLAUTH BLOCKUSER

The BLOCKUSER parameter on the channel authentication rules prevents a specified user or users from connecting. A USERLIST must accompany the BLOCKUSER parameter.

In the user list, the string *MQADMIN represents administrative users as defined for the local operating system.

Rules in the user list are hierarchical; the most specific rule determines the MCAUSER value.

The figure shows two examples of the SET CHLAUTH TYPE(BLOCKUSER) command.

- The first rule blocks administrative users and the MCAUSER “nobody”, which prevents someone from creating a user ID “nobody” and putting it into an authorized group.
- The second rule restricts access for SYSTEM.ADMIN channels to administrators. It is assumed here that some other CHLAUTH rule such as an SSLPEERMAP or an exit validated the connection from administrator.

Filtering by IP address

- Ability to filter connection requests based on IP address of requester
- Per-channel rules match the least-specific to most-specific
- Global blocking rules
 - Occur at the listener before the channel name is known
 - Take precedence over per-channel rules

Authenticating channels and connections

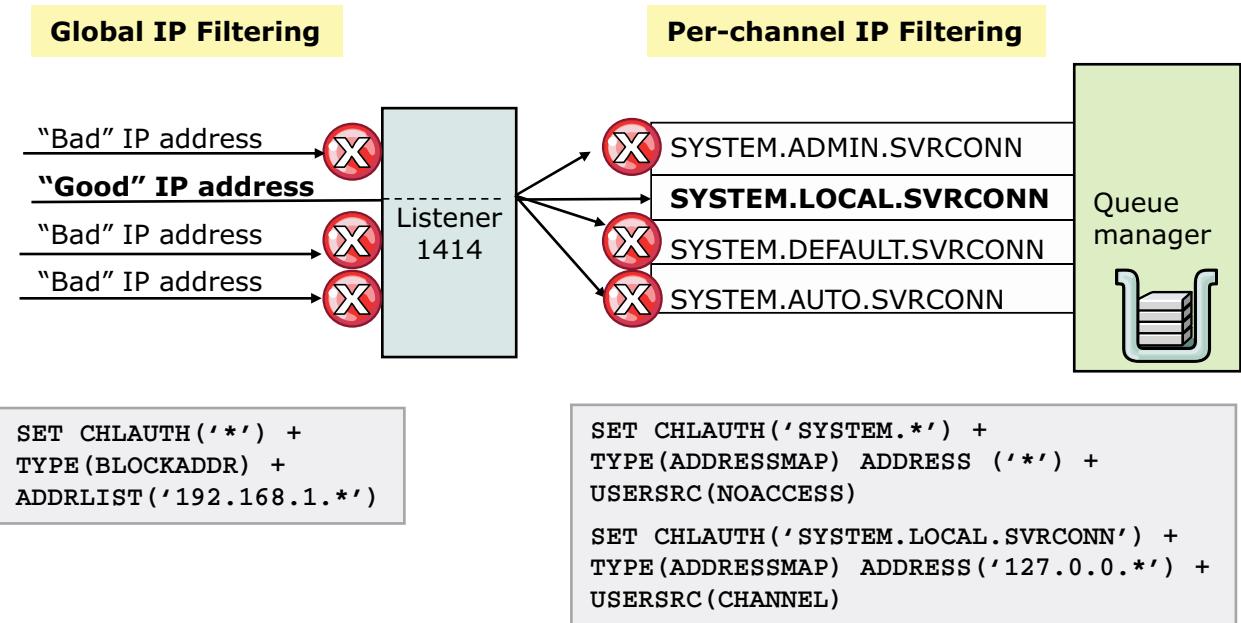
© Copyright IBM Corporation 2020

Figure 10-64. Filtering by IP address

You can filter connection requests based on the IP address of the requester.

Setting TYPE (ADDRESSMAP) in the SET CHLAUTH command maps IP addresses to MCAUSER values. The ADDRESS parameter must accompany the ADDRESSMAP parameter.

Filtering by IP address: Example



Authenticating channels and connections

© Copyright IBM Corporation 2020

Figure 10-65. Filtering by IP address: Example

In the SET CHLAUTH 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 on the left 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 have no access to the queue manager and the channel ends immediately.

The second command gives the local host (127.0.0) access to the channel IBM 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.

This example is the same example that was shown earlier in this topic to illustrate channel authentication rules.

Filtering by IP address: Guidelines

- Provide a list of authorized addresses instead of trying to identify all possible unauthorized addresses
- Use CHLAUTH TYPE(BLOCKADDR) to:
 - Temporarily handle transient problems such as a runaway client
 - Handle specific issues such as port scanners, which cause IBM MQ to generate FDC files
- CHLAUTH TYPE(ADDRESSMAP) rules are hierarchical
 - With all other parameters equal, the most specific matching profile name takes precedence
 - Start with “deny-all” policy rule that is followed by specific access rules

Figure 10-66. Filtering by IP address: Guidelines

The figure lists some guidelines for implementing channel authorization that is based on the IP address.

In general, it might be more concise to list the allowed IP addresses in the address list than trying to identify all the possible unauthorized addresses.

Step-by-step guide to a basic secure setup

- Steps to provide basic security hardening that locks down administrative access
 1. Set MCAUSER
 2. Provision user IDs and groups
 3. Set channels that are not SYSTEM channels
 4. Authorize groups
 5. Set up strong authentication
- Extra configuration is required to provide granularity between different user groups and roles

Figure 10-67. Step-by-step guide to a basic secure setup

The next five pages provide a step-by-step guide to implementing a basic secure setup for an IBM MQ environment. More security is required to provide granularity between different user groups and roles.

Step 1: Set MCAUSER

- Set MCAUSER to value of `nobody` in:
 - SYSTEM.ADMIN.SVRCCONN
 - SYSTEM.AUTO.* channels
 - Default channels
- The string `nobody` is important because it has a specific meaning to IBM MQ and to some operating systems

Authenticating channels and connections

© Copyright IBM Corporation 2020

Figure 10-68. Step 1: Set MCAUSER

Setting the MCAUSER to `nobody` is a method for using a string for which there is no matching account name. Because it was used by convention for many years, most administrators would not create an account that is called `nobody` on any operating system and they would notice if one appeared. In addition, some varieties of UNIX do not allow an account by that name.

IBM MQ verifies that the string does not match an actual account. It might match an account with no privileges, but then you must make sure that someone does not put the account into a privileged access group like “mqm”.

Step 2: Provision user IDs and groups

- Provision two low-privileged user IDs and their corresponding private groups
 - One is used for MCA channels
Example: `mqmmca`
 - One is used for MQI (SVRCONN) channels
Example: `mqmmqi`
- Configure user IDs so that they cannot be used for logon

Authenticating channels and connections

© Copyright IBM Corporation 2020

Figure 10-69. Step 2: Provision user IDs and groups

It is important to note here that the “mqmmca” user ID and its associated group are used to control access on MCA channels, which are channels between queue managers.

The “mqmmqi” user ID and its associated group are used to control access on the MCI channels, which are client channels.

Step 3: Set channels that are not SYSTEM channels

- Set any channels that are not SYSTEM channels of type RCVR, RQSTR, and CLUSRCVR to use `MCAUSER ('mqmmca')`
- Set any channels that are not SYSTEM channels of type SVRCONN to use `MCAUSER ('mqmmqi')`

Figure 10-70. Step 3: Set channels that are not SYSTEM channels

The SYSTEM channel definitions are created automatically when a queue manager is created. Recall that in step 1, these channels have an MCAUSER value that is set to “nobody.” In the channel definitions that are created later, the MCAUSER field must be set to one of the user IDs that were created according to the instructions for Step 2.

Step 4: Authorize groups

- Authorize MCA channels group to +put, +inq, and +setall on all the queues it needs
 - Typically all or most of the application queues
 - Does not include command, transmission, or initiation queues
- Authorize MQI channels group to +get, +browse, +put, +inq, and +disp as needed for application use
 - Do not authorize this group to create queues or have PUT access to transmission, initiation, and command queues

Figure 10-71. Step 4: Authorize groups

The next step is to use the `setmqaut` command or IBM MQ Explorer to set the group authorizations on the queues and the MQI channels.

Step 5: Set up strong authentication

- Expand the basic security configuration
 - Filter by IP address
 - Add user ID blocking
 - Configure dynamic mapping of MCAUSER
- Set up strong authentication to:
 - Allow for full remote administration by IBM MQ administrators
 - Control remote access for non-administrative interactive users and applications

Figure 10-72. Step 5: Set up strong authentication

The final step is to strengthen the security by using channel authentication rules to add IP address or host name filtering, user ID blocking, and dynamic of the MCAUSER.

10.5. Channel exits

Channel exits

Authenticating channels and connections

© Copyright IBM Corporation 2020

Figure 10-73. Channel exits

Channel exit programs

- Called at defined places in the processing sequence of an MCA
 - Users and vendors can write their own channel exit programs
 - Some channel exit programs are supplied by IBM
- Can be on sending MCA, receiving MCA, or both
- In most cases, programs are written in C
 - On Windows, the exit must be a .dll file
 - Can use exits that are written in Java, C, or C++ when IBM MQ classes for JMS applications use channel security and send and receive exits on the MQI channel that starts when the application connects to a queue manager
- Example application:
If channel authentication records are not suitable, you can use channel exits for added security

[Authenticating channels and connections](#)

© Copyright IBM Corporation 2020

Figure 10-74. Channel exit programs

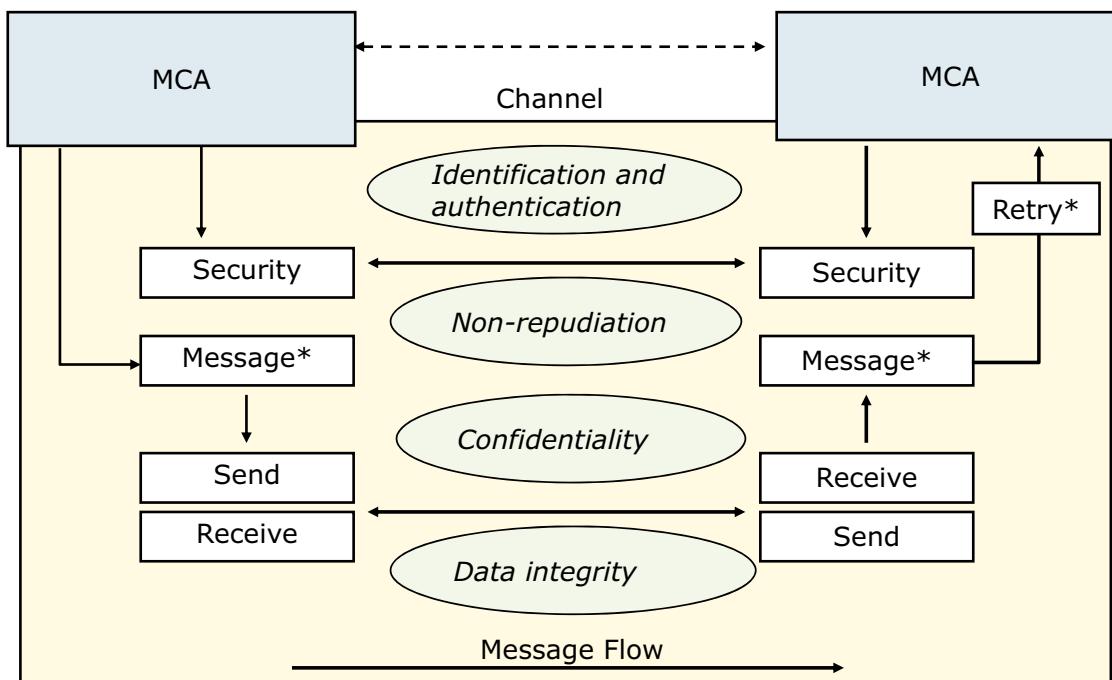
IBM MQ has many options for providing channel security. If the channel security that the IBM MQ provides is not adequate, there are several exit points where you can tailor the behavior of an MCA to your requirements.

All the channel exits normally must be supplied in pairs. That means that if the exit is activated on one side of the channel, it also must be activated on the other side.

Exits are written in C. In Windows, the exit must be a .dll file.

The IBM MQ classes for JMS applications can use channel security and send and receive exits on the MQI channel that starts when the application connects to a queue manager. The application in this case can use exits that are written in Java, C, or C++.

Channel exits for link level security



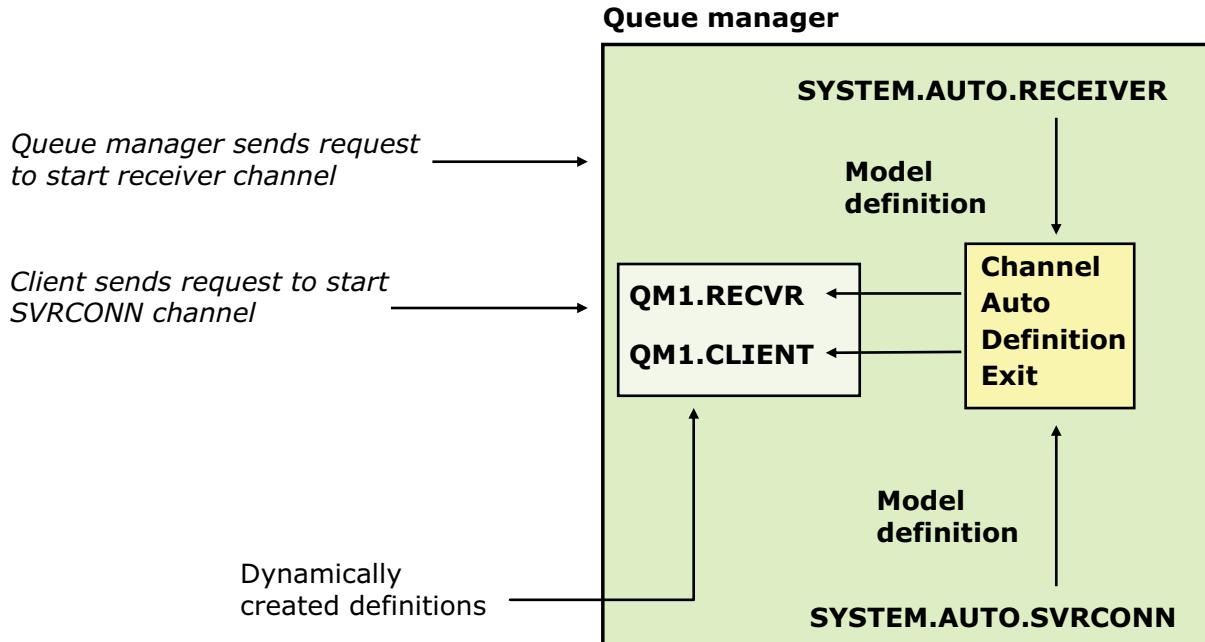
* Not available on client and server connection

Figure 10-75. Channel exits for link level security

There are several types of channel exit program, but only four have a role in providing link level security:

- **Security exit** is called during the channel setup. It allows for the addition of code that can check the security credentials of the partner MCA. This process is known as authentication. Normally, the receiving side starts the security exit data flow, thus forcing the requester side to act appropriately, or the channel is not allowed to start. Security data flows run before any messages are transmitted.
- **Message exit** is called on the sending side after a message is read from the transmission queue and on the receiving side before a message is put onto a destination queue. The message exit receives the entire message, including the IBM MQ message descriptor (MQMD) structure, and it can change the message and its descriptor, if required. The typical purpose of the message exit is the encryption and decryption of data or the logging of messages to ensure that a particular message was sent or received.
- **Send and receive exits** are called before a buffer of data is transmitted over the network or after one is received from the network. These exits can be called multiple times for a single message. Their typical purpose is data compression and decompression.
- **Retry exit** is called when a received message cannot be delivered to the target queue that the transmission queue header specifies, which includes reasons that are of a temporary nature.

Channel auto-definition exit



Authenticating channels and connections

© Copyright IBM Corporation 2020

Figure 10-76. Channel auto-definition exit

The channel auto-definition exit is called when a request is received to start a cluster-sender channel. It is also called when starting a cluster-receiver channel.

You can use the channel auto-definition exit to modify the supplied default definition for an automatically defined receiver or server-connection channel, SYSTEM.AUTO.RECEIVER, or SYSTEM.AUTO.SVRCONN. The exit might change most of these parameters.

The channel auto-definition exit is the property of the queue manager, not the individual channel. It must be named in the queue manager definition. The regular command is DEFINE or ALTER QMGR CHADEXIT(*programName*).

Channel exits on message channels

Channel type	Message exit	Message retry exit	Receive exit	Security exit	Send exit	Auto-definition exit
Sender	Yes		Yes	Yes	Yes	
Server	Yes		Yes	Yes	Yes	
Receiver	Yes	Yes	Yes	Yes	Yes	Yes
Requester	Yes	Yes	Yes	Yes	Yes	Yes
Cluster-sender	Yes		Yes	Yes	Yes	
Cluster-receiver	Yes	Yes	Yes	Yes	Yes	Yes
Client-connection			Yes	Yes	Yes	
Server-connection			Yes	Yes	Yes	Yes

Authenticating channels and connections

© Copyright IBM Corporation 2020

Figure 10-77. Channel exits on message channels

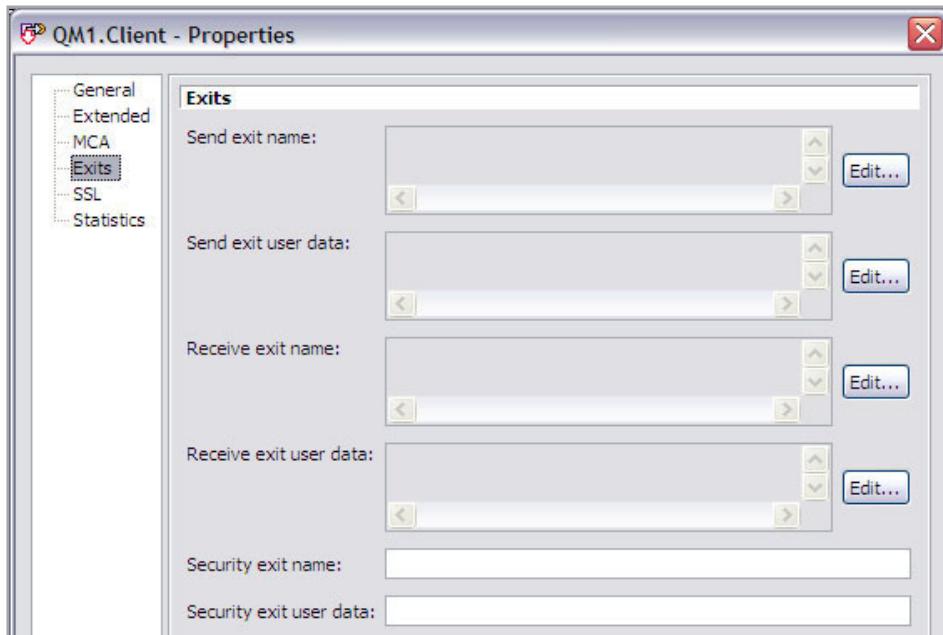
The figure identifies the IBM MQ exits that are available for each of the channel types.

Channel exits include message exits, message-retry exits, receive exits, security exits, send exits, and auto-definition exits.

Channel types include sender channels, server channels, receiver channels, requester channels, cluster-sender channels, cluster-receiver channels, client-connection channels, and server-connection channels.



Channel exit configuration



```
ALTER CHANNEL ('QM1.CLIENT') CHLTYPE(SVRCONN) +
SCYEXIT(MySecExit) SCYDATA(MAXIMUM)
```

Authenticating channels and connections

© Copyright IBM Corporation 2020

Figure 10-78. Channel exit configuration

For IBM MQ on UNIX, Linux, and Windows, you can specify the channel exit program names as properties of a channel definition that are set when you define or alter the channel.

The figure shows how IBM MQ Explorer or MQSC can be used to define or modify the channel to implement a security exit or message exit.

The security exit can specify a single program. The message exit and the send and receive exits can specify the name of more than one exit program. When multiple programs are specified, a comma separates each program name. However, the total number of characters that are specified must not exceed 999.

For the security exit, a text string of up to 32 characters in length can be specified. This string is passed to the exit program when it starts. For the types of exit programs, you can specify data for more than one exit program by separating multiple strings with a comma. The total length of the field must not exceed 999.

Location of exit modules

- Default exit paths
 - Windows: <install_location>\exits
 - UNIX and Linux: /var/mqm/exits (32-bit channel exits)
/var/mqm/exits64 (64-bit channel exits)
- If required, default exit paths can be changed
 - On the server, paths are in the **ExitPath** stanza the **qm.ini** file
 - On a client, paths are in the **ClientExitPath** stanza in the IBM MQ client configuration file **client.ini**

Figure 10-79. Location of exit modules

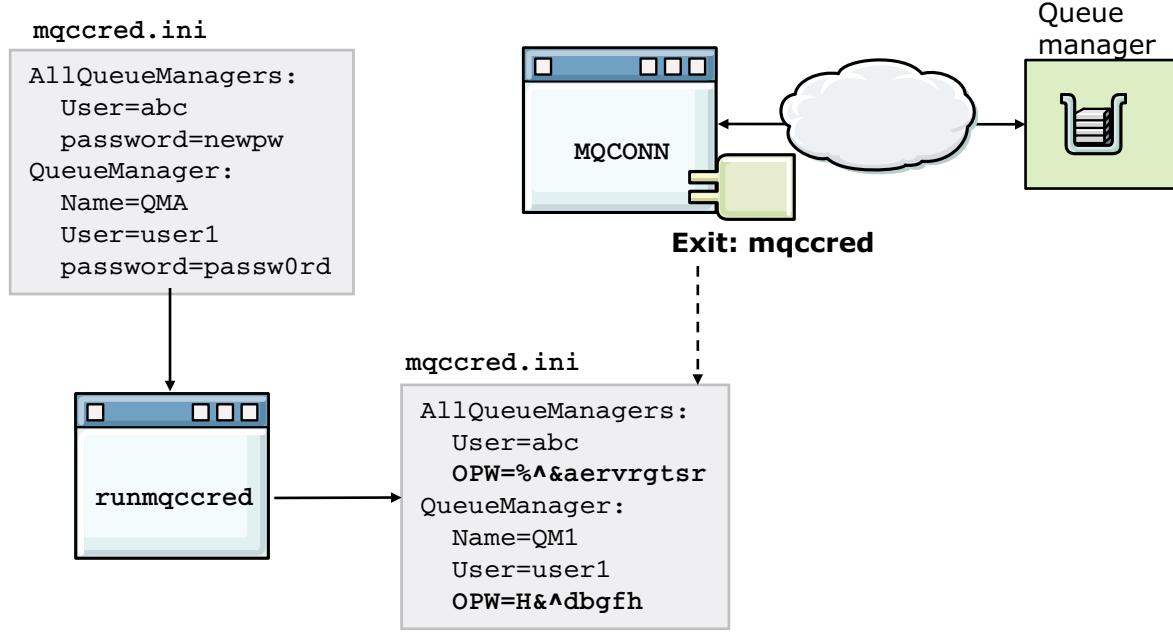
On UNIX, Linux and Windows systems, a client configuration file is added to the system during installation of the MQI client. A default path for location of the channel exits on the client is defined in this file, by using the stanza:

```
ClientExitPath:  
ExitsDefaultPath=string  
ExitsDefaultPath64=string
```

Where, **string** is a file location in a format appropriate to the operating system.

Client-side security exit

- IBM MQ provides a client-side security exit for setting the user ID and password for IBM MQ user authentication



[Authenticating channels and connections](#)

© Copyright IBM Corporation 2020

Figure 10-80. Client-side security exit

IBM MQ provides a client-side security exit to set the user ID and password. You might want to use this exit instead of changing an application.

The exit runs at the client-connection end of the channel and gets the user ID and the password from a file. The operating system controls the permissions to this file. If the exit discovers that the file permissions are too open, it causes a failure so that it is evident that this file is not protected.

IBM MQ provides a tool that obfuscates the passwords in this file from casual browsers. The algorithm for this obfuscation is not published, and neither is the source of the exit.

Unit summary

- Determine the current level of authentication that is enabled on a queue manager and a connection
- Add authentication to a channel
- Add authentication to a connection
- Identify and fix channel authentication and connection authentication problems
- Implement a channel exit program for securing messaging channels

Review questions

1. True or False: Authorization is the process of determining the user ID of the user.
2. True or False: If you have MCAUSER set to blank on any inbound channel definitions, then you have a serious security exposure.
3. After entering the following command, what access does the “payroll” group have to the queue:
`setmqaut -m DEMO -n DEMO.QUEUE -t queue -g payroll +put`
 - a. No access
 - b. PUT access
 - c. Full access
 - d. PUT access plus any other access that a previous `setmqaut` command granted



Authenticating channels and connections

© Copyright IBM Corporation 2020

Figure 10-82. Review questions

Write your answers here:

- 1.
- 2.
- 3.

Review answers

- True or False: Authorization is the process of determining the user ID of the user.

The answer is False. Authorization is the control access to resources on a per-ID, per-role, or per-profile basis



- True or False: If you have MCAUSER set to blank on any inbound channel definitions, then you have a serious security exposure.

The answer is True.

- After entering the following command, what access does the “payroll” group have to the queue:

`setmqaut -m DEMO -n DEMO.QUEUE -t queue -g payroll +put`

- a. No access
- b. PUT access
- c. Full access
- d. PUT access plus any other access that a previous `setmqaut` command granted

The answer is D.

Exercise: Implementing connection authentication

Authenticating channels and connections

© Copyright IBM Corporation 2020

Figure 10-84. Exercise: Implementing connection authentication

Exercise introduction

- Check locally bound connections
- Check client connections
- Configure the authentication failure delay



Authenticating channels and connections

© Copyright IBM Corporation 2020

Figure 10-85. Exercise introduction

Unit 11. Supporting JMS with IBM MQ

Estimated time

00:30

Overview

In this unit, you learn about IBM MQ support for Java Message Service (JMS).

How you will check your progress

- Review questions

Unit objectives

- Describe IBM MQ as a JMS provider
- Manage JMS resources in IBM MQ Explorer

Supporting JMS with IBM MQ

© Copyright IBM Corporation 2020

Figure 11-1. Unit objectives

Topics

- IBM MQ and JMS
- Managing JMS resources with IBM MQ Explorer

Supporting JMS with IBM MQ

© Copyright IBM Corporation 2020

Figure 11-2. Topics

11.1. IBM MQ and JMS

IBM MQ and JMS

Supporting JMS with IBM MQ

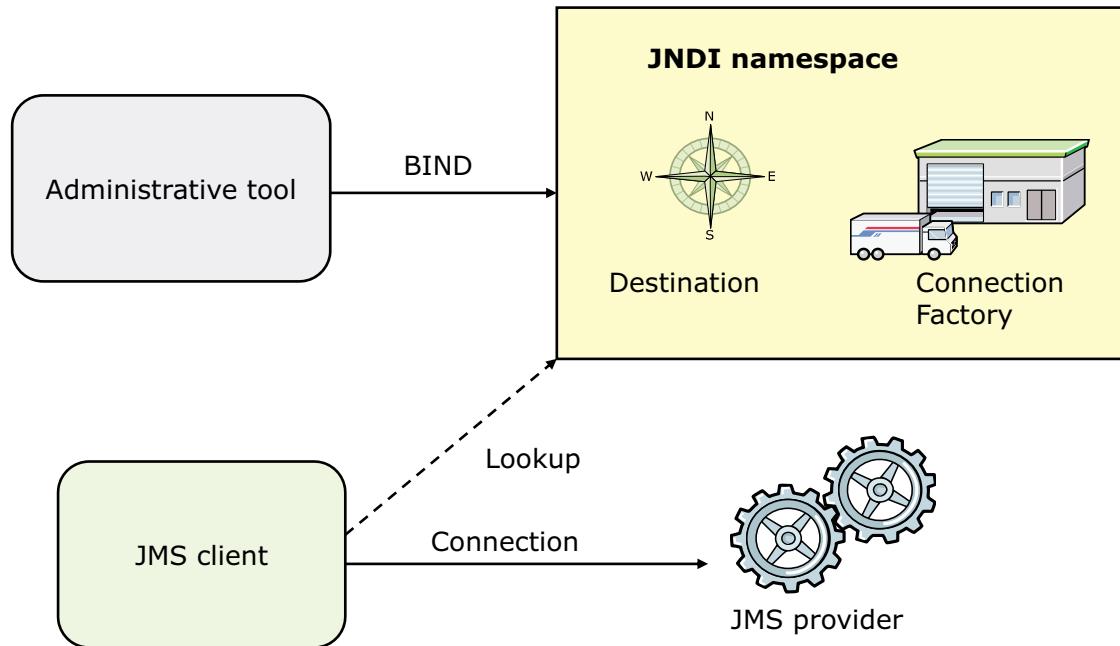
© Copyright IBM Corporation 2020

Figure 11-3. IBM MQ and JMS

What is JMS?

- JMS is a messaging API for Java applications
- JMS V2.0 is current standard
- Part of Java Platform, Enterprise Edition (Java EE)
- Allows the communication between different components of a distributed application to be loosely coupled, reliable, and asynchronous
- Supports point-to-point and publish/subscribe messaging styles

JMS elements



Supporting JMS with IBM MQ

© Copyright IBM Corporation 2020

Figure 11-5. JMS elements

JMS administered objects

- Connection factory
 - Set of connection configuration parameters that a client uses to create a connection with a JMS provider
 - On IBM MQ, contains queue manager connection information
- Destination
 - Object that a client uses to specify the destination of messages it is sending and the source of messages it receives
 - On IBM MQ, identifies queue or topic information

JMS providers

- IBM MQ
- IBM Application Server: Service Integration Bus
- Open source: Apache ActiveMQ
- Just about any other Java EE provider

Supporting JMS with IBM MQ

© Copyright IBM Corporation 2020

Figure 11-7. JMS providers

IBM MQ classes for Java or JMS

IBM MQ classes for Java	JMS
<ul style="list-style-type: none">• Encapsulate MQI• Easy to implement for anyone familiar with MQI in other languages• Can use the full features of IBM MQ• Similar object model to other object-oriented language interfaces like C++ and .NET• Built with Java 7	<ul style="list-style-type: none">• Implement Oracle JMS interfaces• Easy for anyone with JMS skills• Part of Java EE• Can use message-driven beans• Can use other JMS providers• Bridge applications between JMS providers

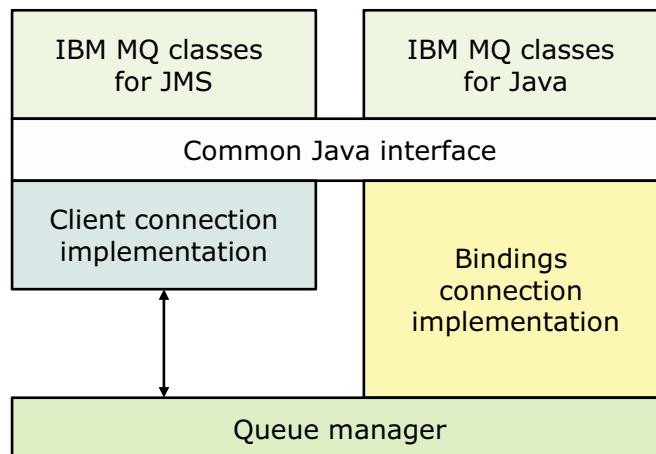
Supporting JMS with IBM MQ

© Copyright IBM Corporation 2020

Figure 11-8. IBM MQ classes for Java or JMS

IBM MQ classes for JMS and IBM MQ classes for Java

- IBM MQ classes for Java and IBM MQ classes for JMS are peers that use a common Java interface to the MQI
 - More scope for optimizing performance
 - Setting fields or calling methods in the MQEnvironment class does not affect runtime behavior of code that is written by using IBM MQ classes for JMS



Supporting JMS with IBM MQ

© Copyright IBM Corporation 2020

Figure 11-9. IBM MQ classes for JMS and IBM MQ classes for Java

JMS message properties

- When an IBM MQ classes for JMS application sends a JMS message, the JMS message is mapped to an IBM MQ message
 - Some JMS header fields and properties are mapped into fields in the MQMD
 - Some JMS header fields and properties are mapped into fields in the MQRFH2 header
- When an application calls MQGET to receive a message from an IBM MQ classes for JMS application, the application can choose to receive the message in one of the following ways:
 - Message contains message descriptor, an MQRFH2 header that contains data that is derived from JMS header fields and properties, and the application data
 - Message contains message descriptor, application data, and a set of message properties

Support for JMS 2.0 in IBM MQ

- Delayed delivery
 - Defer message delivery by specifying a delivery delay when a message is sent so that the JMS provider does not deliver the message until after the specified delivery delay elapses
- Shared subscriptions
 - Share messages from a topic subscription among multiple consumers
 - Each message from the subscription is delivered to only one of the consumers on that subscription
- Asynchronous send operation
 - Applications can send messages asynchronously
 - Control is immediately returned to the sending application without waiting for a reply from the server
- Any code that is written to be used with IBM MQ must use the `jms.jar` file that is supplied with IBM MQ

11.2. Managing JMS resources with IBM MQ Explorer

Managing JMS resources with IBM MQ Explorer

Supporting JMS with IBM MQ

© Copyright IBM Corporation 2020

Figure 11-12. Managing JMS resources with IBM MQ Explorer

Configuring JMS administered objects in IBM MQ

- Before an application can retrieve administered objects from a JNDI namespace, an administrator must first create the administered objects
 - Connect to the JNDI namespace and add an initial context
 - Create and configure connection factory and destination objects that are stored in the JNDI namespace
- Use IBM MQ Explorer to create and maintain administered objects in a JNDI namespace from IBM MQ

Supporting JMS with IBM MQ

© Copyright IBM Corporation 2020

Figure 11-13. Configuring JMS administered objects in IBM MQ



IBM MQ Explorer and JMS

- **JMS Administered Objects** folder contains initial contexts
 - An initial context must be added to IBM MQ Explorer before you can administer the JMS administered objects
 - You can add more initial contexts and you can remove initial contexts when you no longer want to use IBM MQ Explorer
- After you add the initial context to IBM MQ Explorer, you can create:
 - Connection factory objects
 - Destination objects
 - Subcontexts in the JNDI namespace

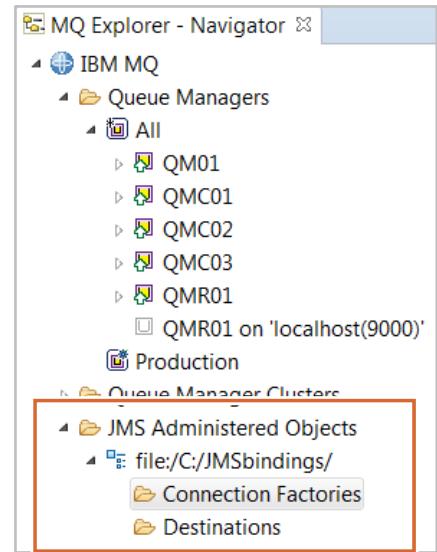
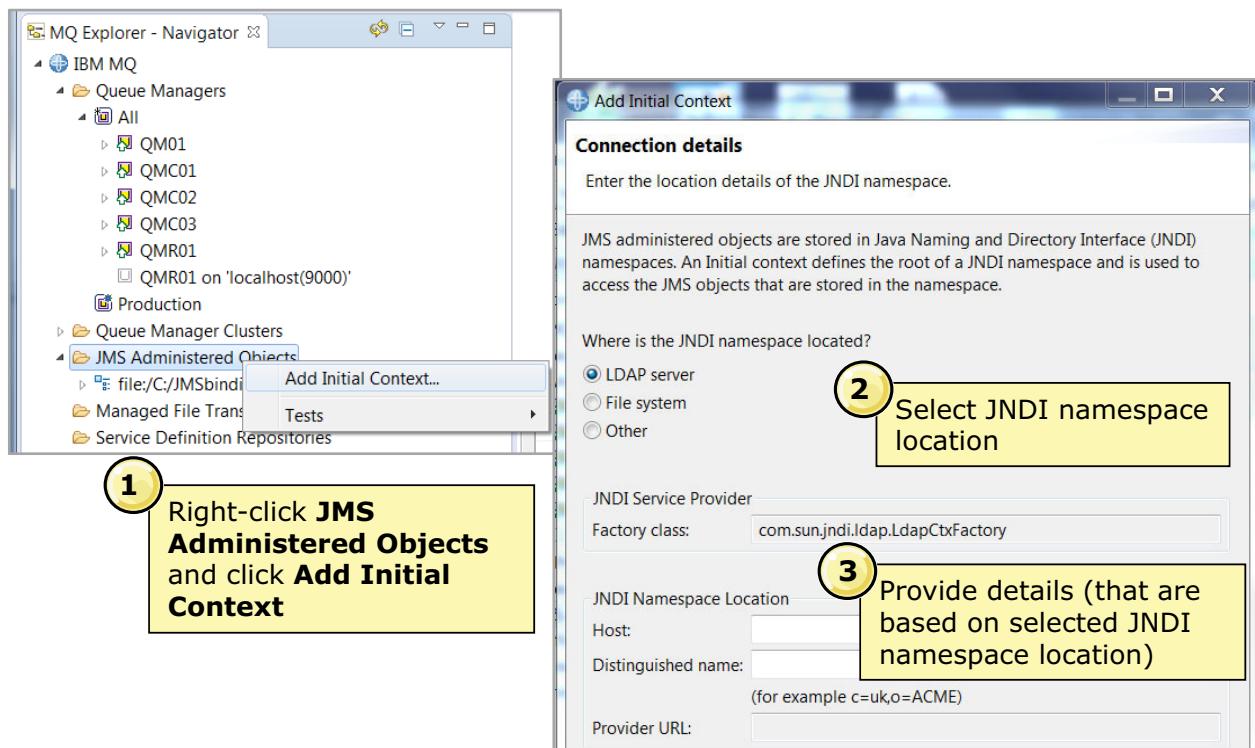


Figure 11-14. IBM MQ Explorer and JMS



Adding an initial context



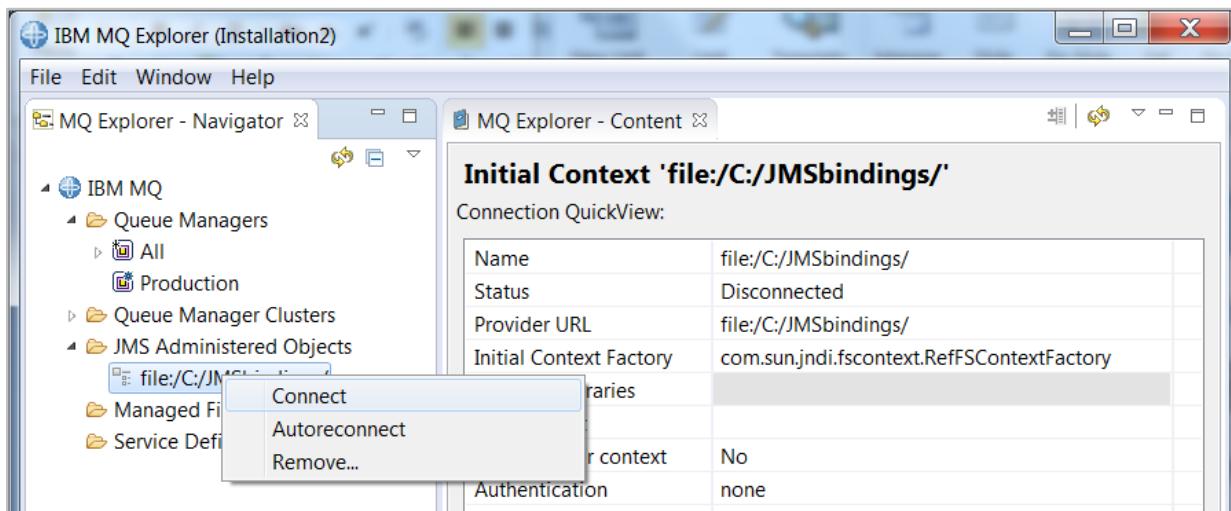
Supporting JMS with IBM MQ

© Copyright IBM Corporation 2020

Figure 11-15. Adding an initial context



Connecting the initial context

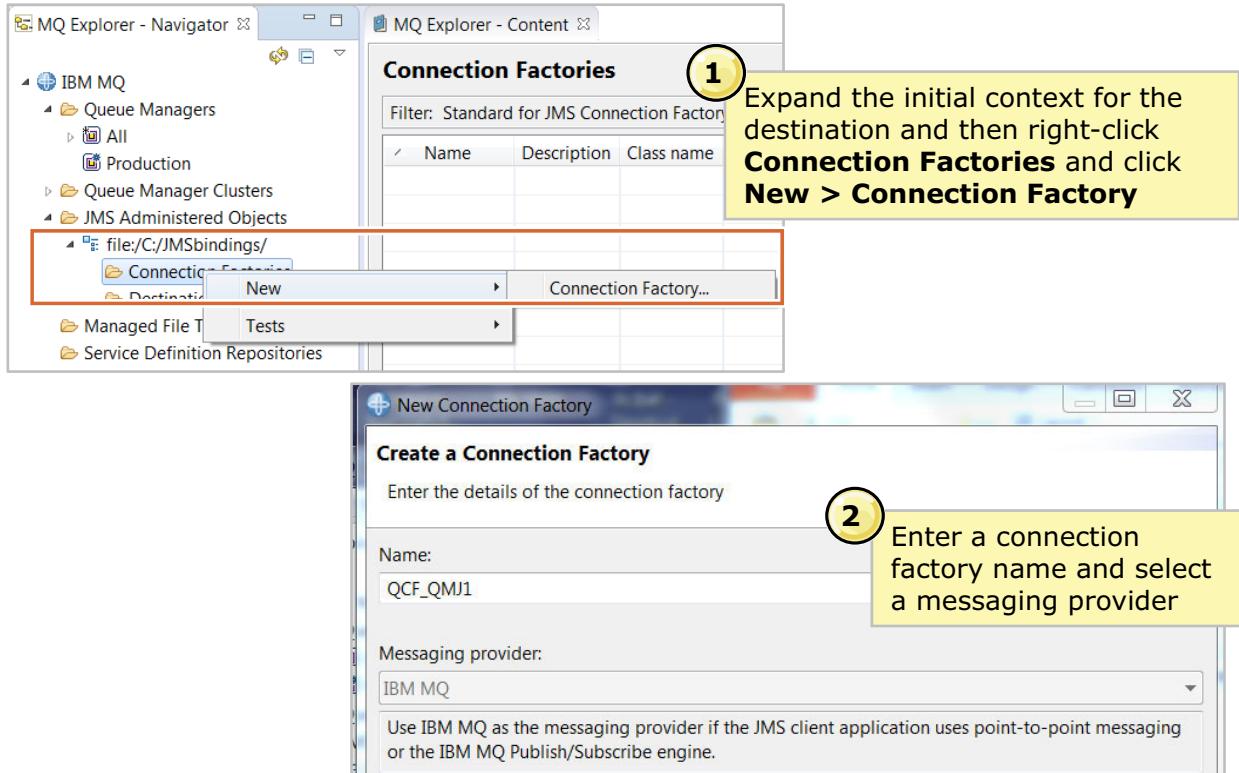


- Initial context must be connected to create and administer connection factories and destinations

Figure 11-16. Connecting the initial context



Creating a connection factory (1 of 3)

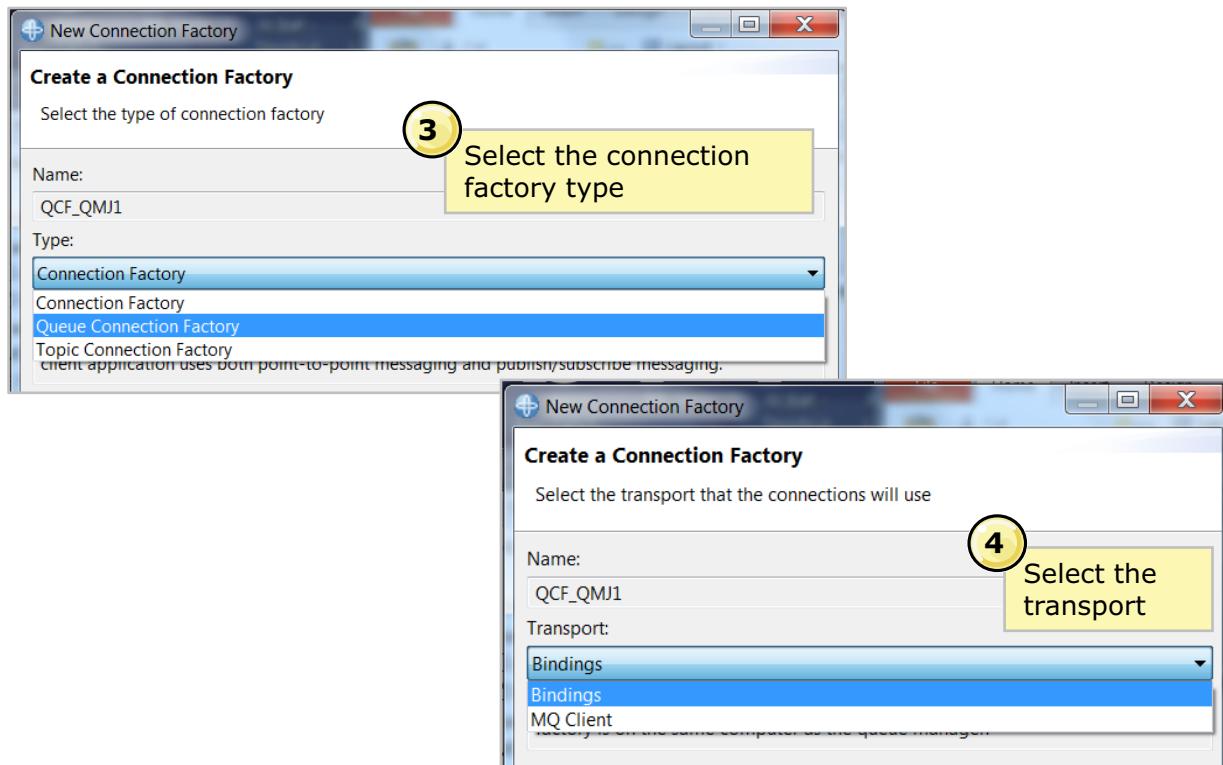


Supporting JMS with IBM MQ

© Copyright IBM Corporation 2020

Figure 11-17. Creating a connection factory (1 of 3)

Creating a connection factory (2 of 3)

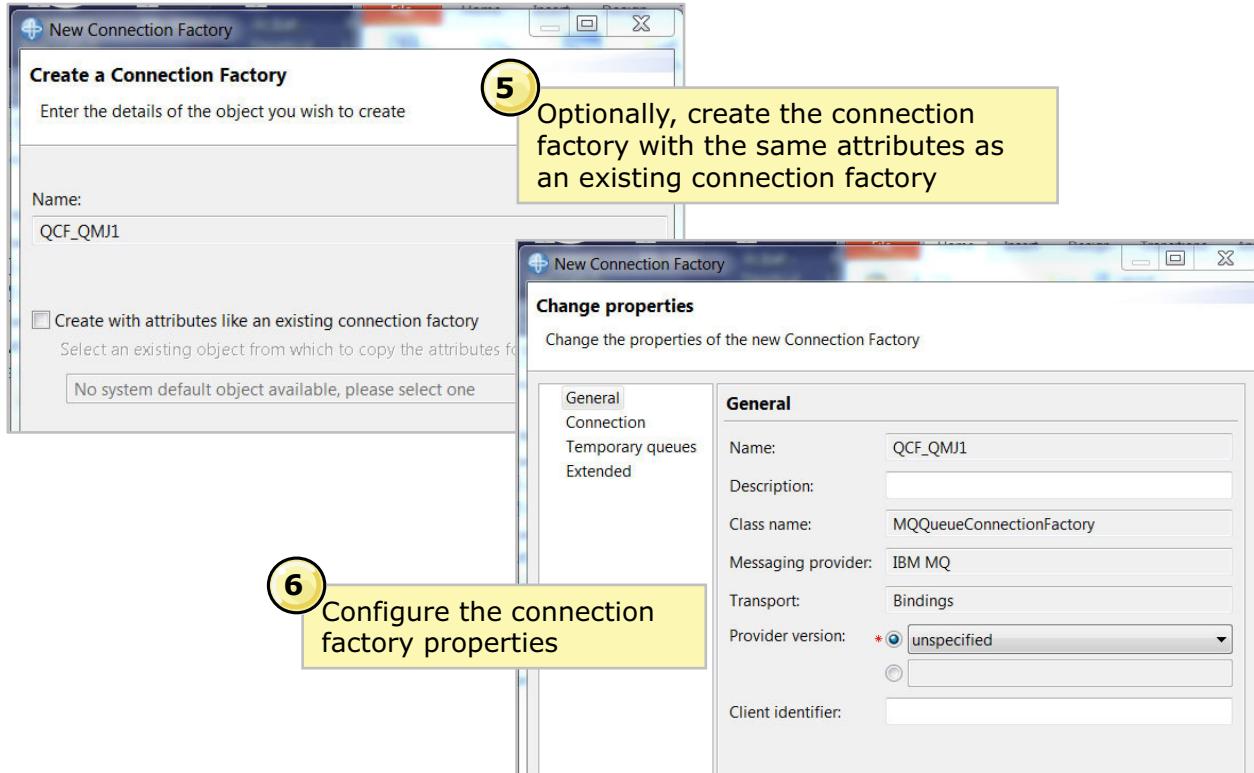


Supporting JMS with IBM MQ

© Copyright IBM Corporation 2020

Figure 11-18. Creating a connection factory (2 of 3)

Creating a connection factory (3 of 3)

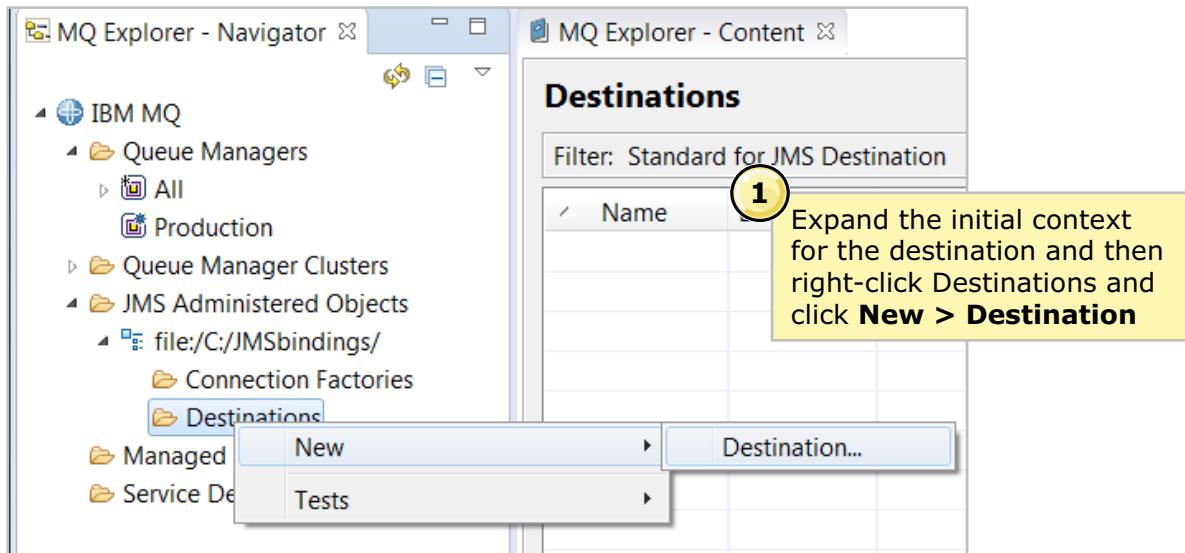


Supporting JMS with IBM MQ

© Copyright IBM Corporation 2020

Figure 11-19. Creating a connection factory (3 of 3)

Creating a JMS destination (1 of 3)



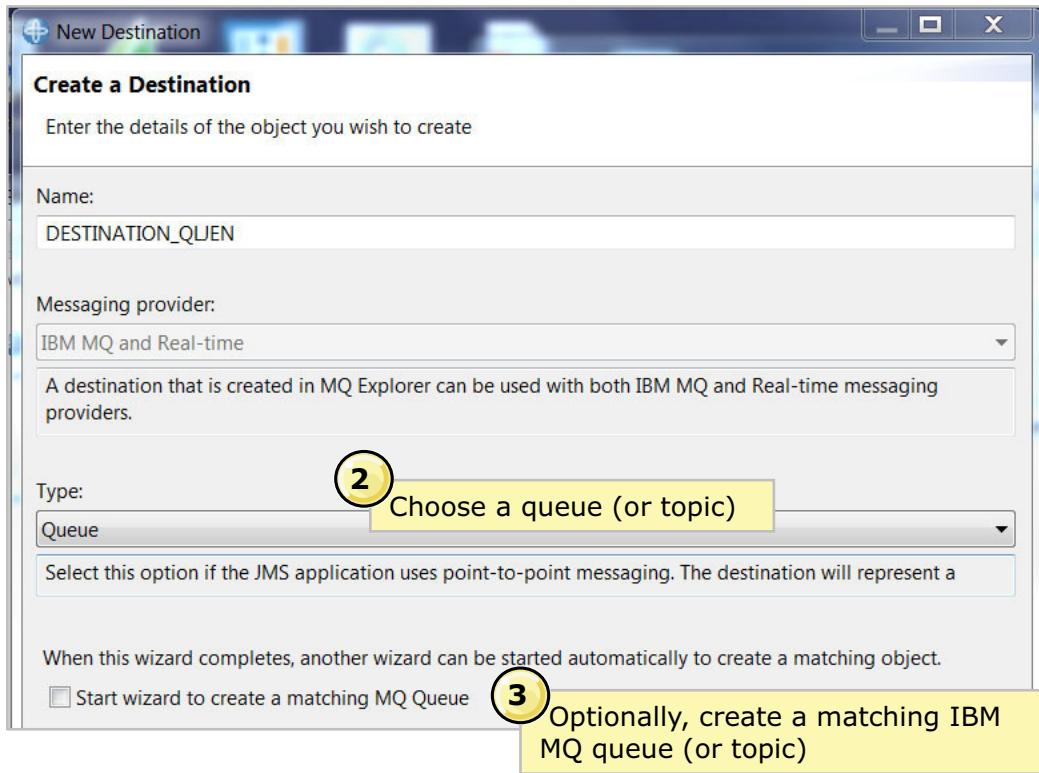
Supporting JMS with IBM MQ

© Copyright IBM Corporation 2020

Figure 11-20. Creating a JMS destination (1 of 3)

IBM Training

Creating a destination (2 of 3)

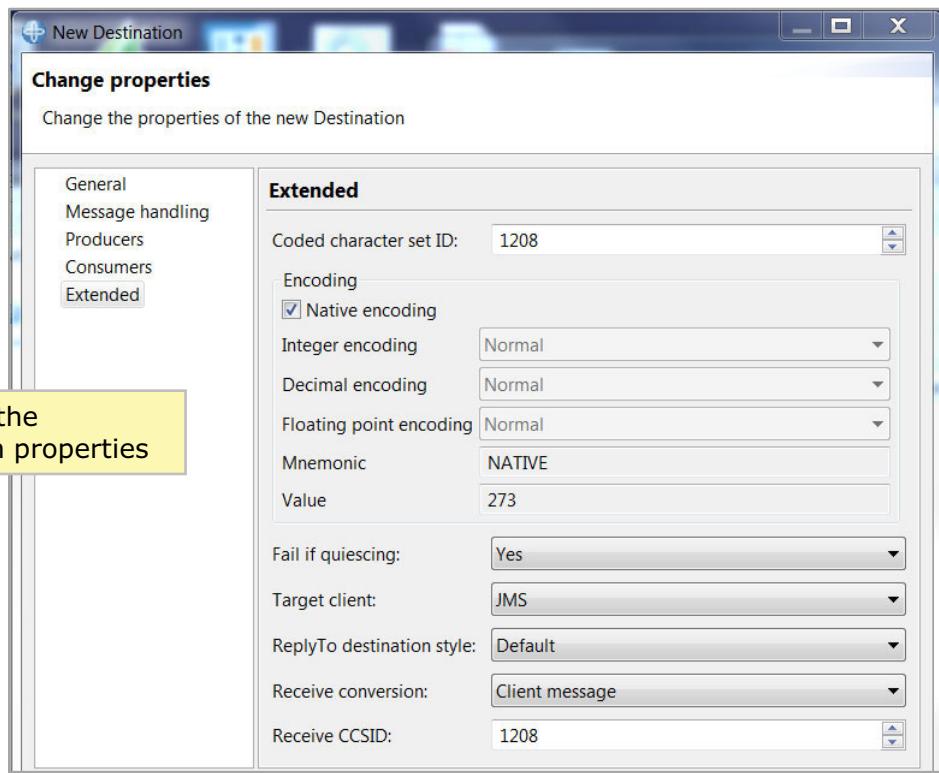


Supporting JMS with IBM MQ

© Copyright IBM Corporation 2020

Figure 11-21. Creating a destination (2 of 3)

Creating a destination (3 of 3)



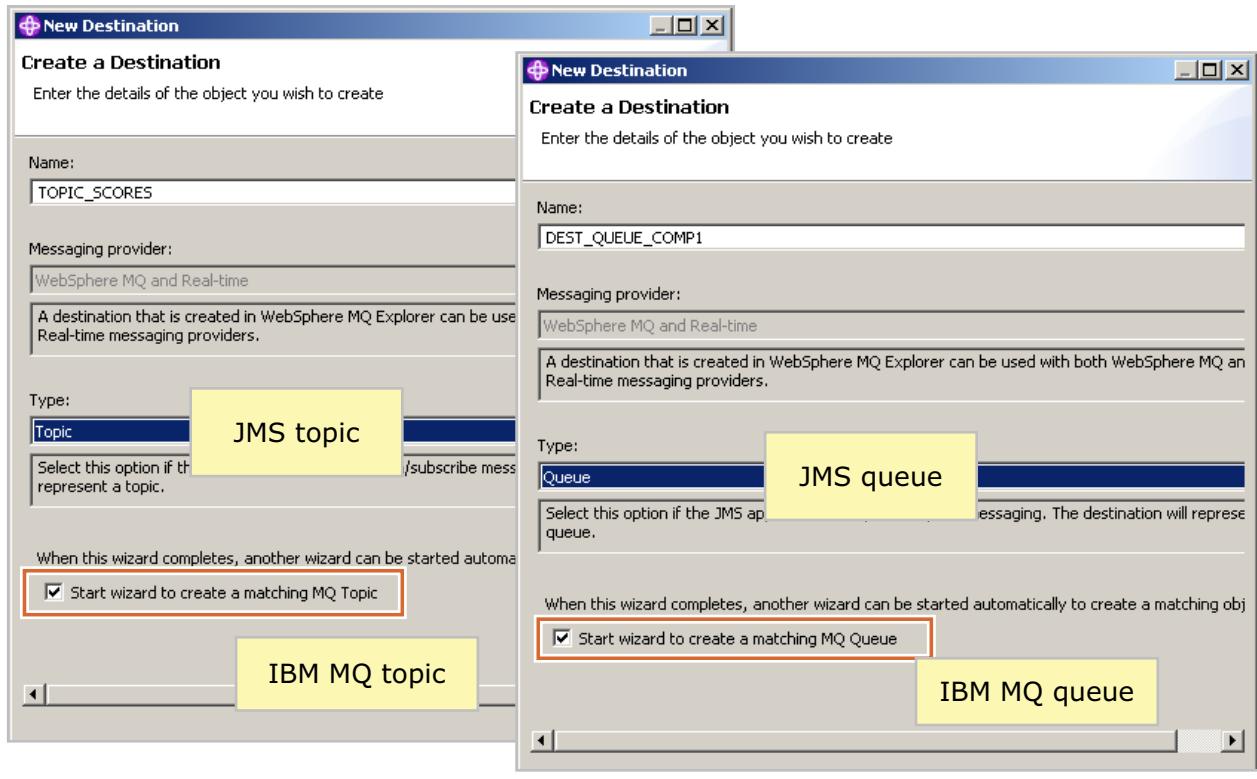
Supporting JMS with IBM MQ

© Copyright IBM Corporation 2020

Figure 11-22. Creating a destination (3 of 3)



Mapping between IBM MQ and JMS objects (1 of 2)

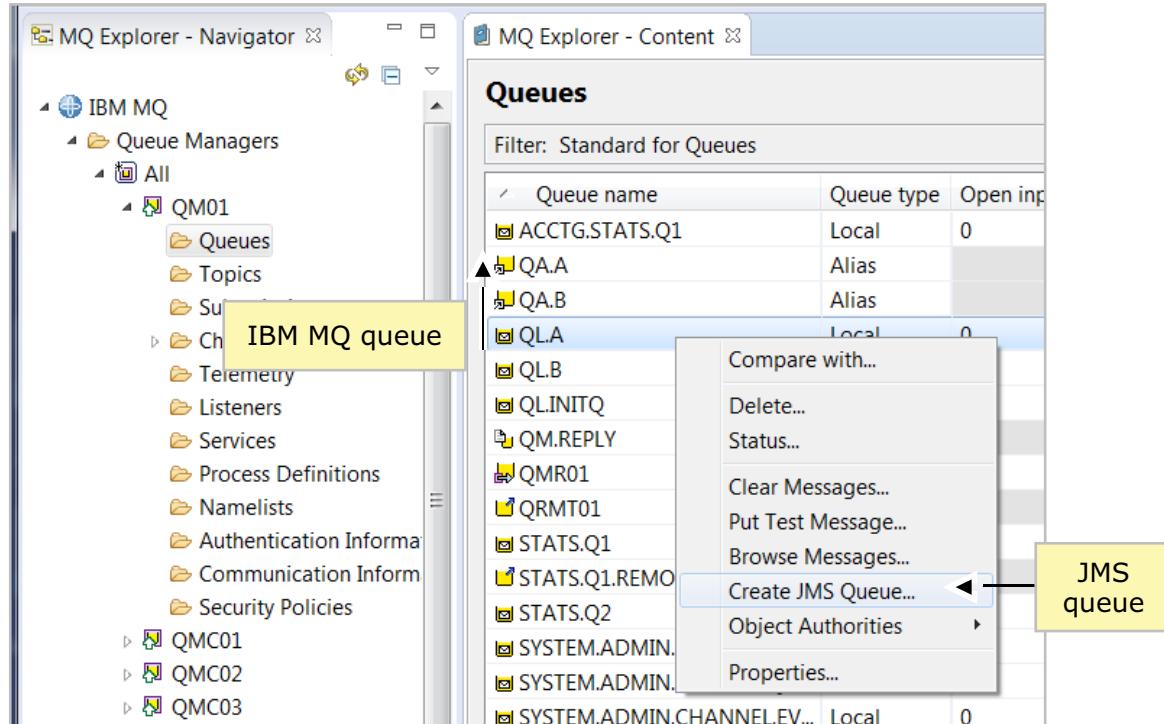


Supporting JMS with IBM MQ

© Copyright IBM Corporation 2020

Figure 11-23. Mapping between IBM MQ and JMS objects (1 of 2)

Mapping between IBM MQ and JMS objects (2 of 2)



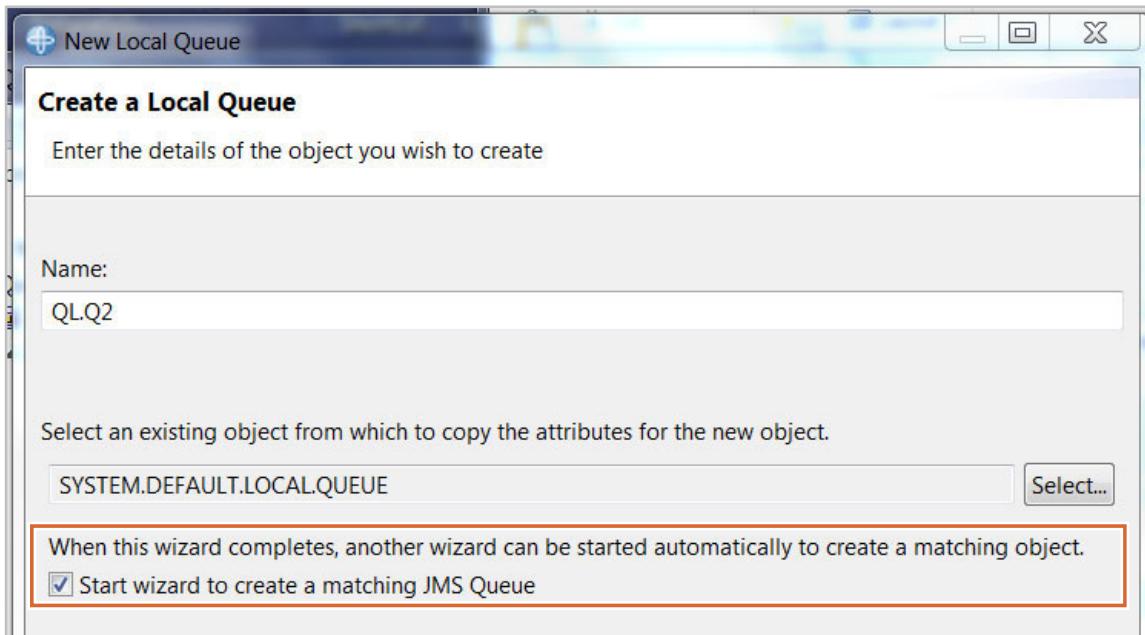
Supporting JMS with IBM MQ

© Copyright IBM Corporation 2020

Figure 11-24. Mapping between IBM MQ and JMS objects (2 of 2)

IBM Training 

JMS object creation wizards



Supporting JMS with IBM MQ

© Copyright IBM Corporation 2020

Figure 11-25. JMS object creation wizards

IBM MQ classes for JMS problem determination

- An application can use a provided class to:
 - Start and stop tracing
 - Specify the required level of detail in a trace
 - Customize trace output
- Logging
 - Log file contains messages about errors in plain text
 - An application can use a provided class to specify the location of the log file and its maximum size
- If a failure occurs, an FFST report that contains information that IBM Support can use to diagnose the problem is generated in an FDC file
- An application can use a provided class to query the version of IBM MQ classes for JMS
- Exception messages provide information about the causes of errors and the actions that are required to correct errors

Unit summary

- Describe IBM MQ as a JMS provider
- Manage JMS resources in IBM MQ Explorer

Supporting JMS with IBM MQ

© Copyright IBM Corporation 2020

Figure 11-27. Unit summary

Review questions

1. Which two objects are JMS administered objects:
 - A. Queue
 - B. Destination
 - C. Collection Factory
 - D. TARGCLIENT
2. True or False: The JNDI namespace can be a database.



Supporting JMS with IBM MQ

© Copyright IBM Corporation 2020

Figure 11-28. Review questions

Review answers

1. Which two objects are JMS administered objects:

- A. Queue
- B. Destination
- C. Collection Factory
- D. TARGCLIENT

The answer is A and B.



2. True or False: The JNDI namespace can be a database.

The answer is True.

Unit 12. 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
- Lab exercise

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

Topics

- Finding a missing message
- Cluster rerouting
- First Failure Support Technology (FFST)
- Error logs
- IBM MQ trace

[Diagnosing problems](#)

© Copyright IBM Corporation 2020

Figure 12-2. Topics

12.1. Finding a missing message

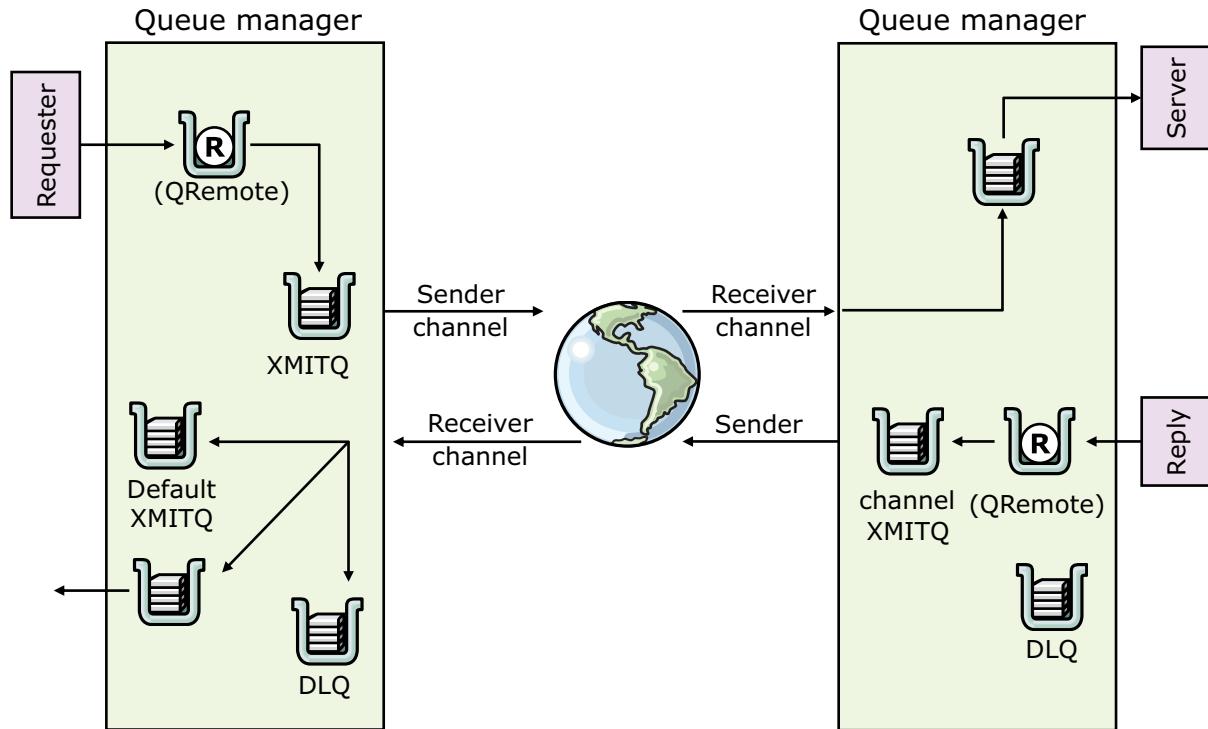
Finding a missing message

[Diagnosing problems](#)

© Copyright IBM Corporation 2020

Figure 12-3. Finding a missing message

Where is the message?



Diagnosing problems

© Copyright IBM Corporation 2020

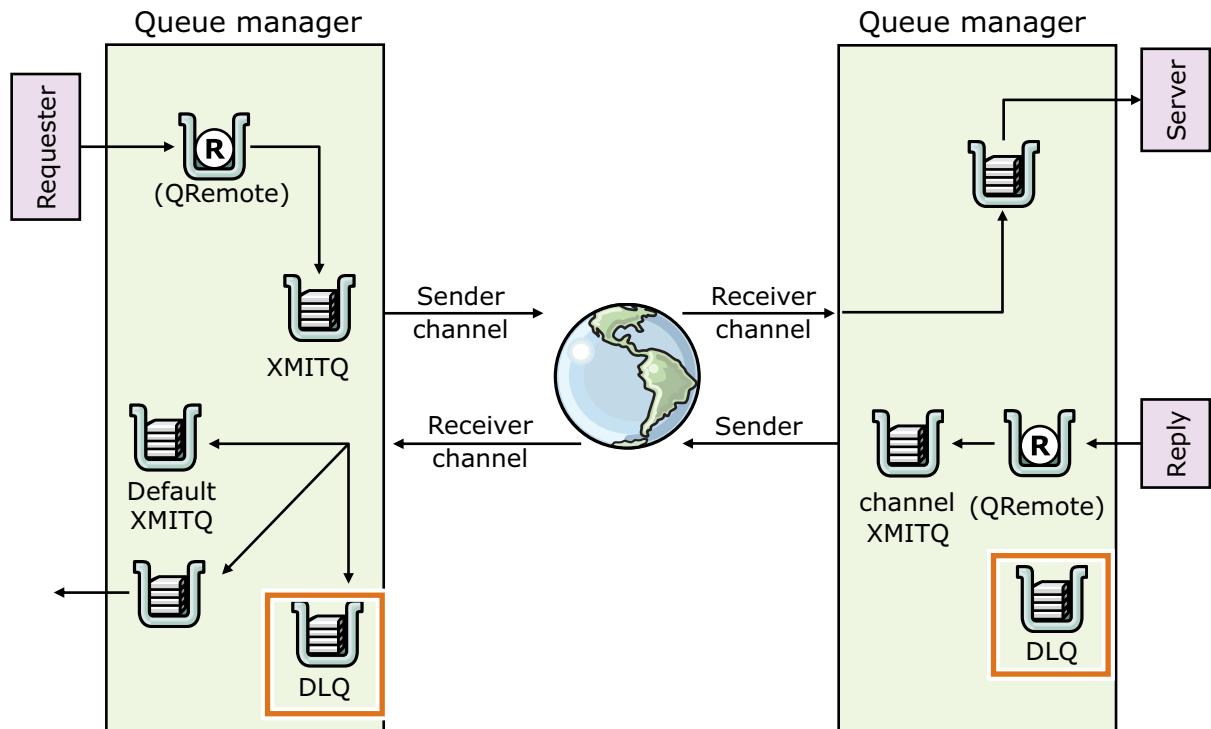
Figure 12-4. Where is the message?

The figure shows a typical IBM 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.

IBM 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.

Dead-letter queue



Diagnosing problems

© Copyright IBM Corporation 2020

Figure 12-5. Dead-letter queue

The first place to look for a message is the dead-letter queue.

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 2020

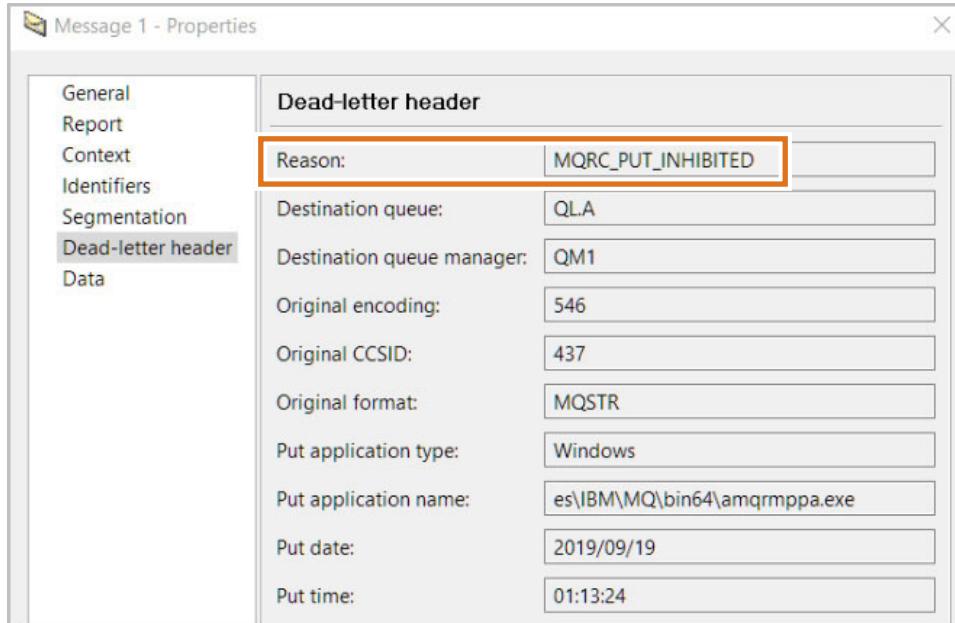
Figure 12-6. Missing message checklist

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 in IBM MQ Explorer

- Review the **Reason** property



Diagnosing problems

© Copyright IBM Corporation 2020

Figure 12-7. Finding the dead-letter reason in IBM MQ Explorer

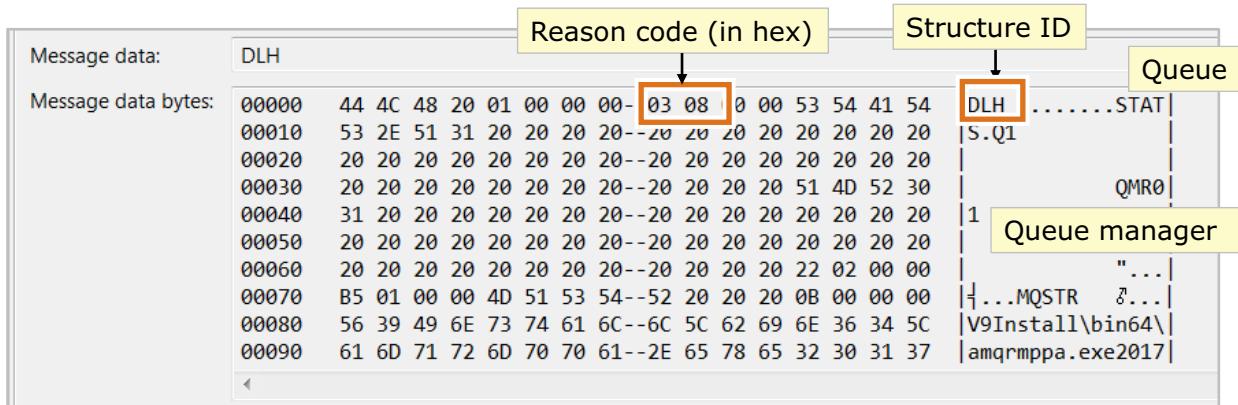
One of the first places to look for a message is the queue manager dead-letter queue.

The dead-letter reason is found by using the IBM MQ Explorer message properties.

Browse the message on the dead-letter queue in IBM MQ Explorer and view the reason in the **Dead-letter header** message properties

Finding the dead-letter reason with amqsbcg

- Reason code location in message data



Diagnosing problems

© Copyright IBM Corporation 2020

Figure 12-8. Finding the dead-letter reason with amqsbcg

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.

The amqsbcg sample program can browse the message on the dead-letter queue and provide the reason code. This reason code can be looked up with the mqrc command to determine 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
- Completion codes `MQCC_WARNING` and `MQCC_FAILED` include return code
- Use `mqrc` command to display information about return codes

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
```

[Diagnosing problems](#)

© Copyright IBM Corporation 2020

Figure 12-9. 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.

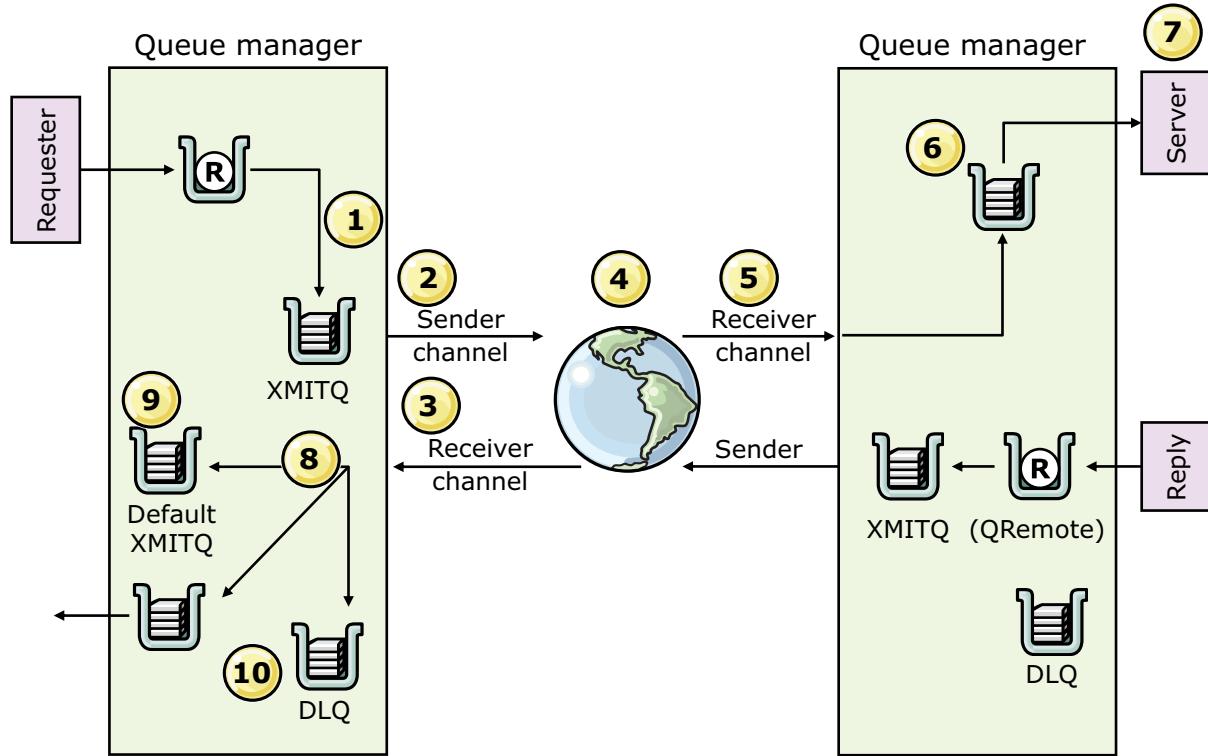
If the completion code is `MQCC_WARNING` or `MQCC_FAILED`, use the `mqrc` control command to check the reason code for more information about the problem.

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 2020

Figure 12-10. Checking for IBM MQ network problems

The next step for determining the reason for a missing message is to check the IBM 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 from IBM MQ Explorer. The window title is 'Message browser'. It displays information for a queue named 'ANDREW.TEST.PERSISTENT' under 'Queue Manager Name: Canberra1'. There are two messages listed in the table:

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 2020

Figure 12-11. Checking message persistence

Restarting a queue manager with nonpersistent messages on the queues is one of the most common reasons that messages are lost.

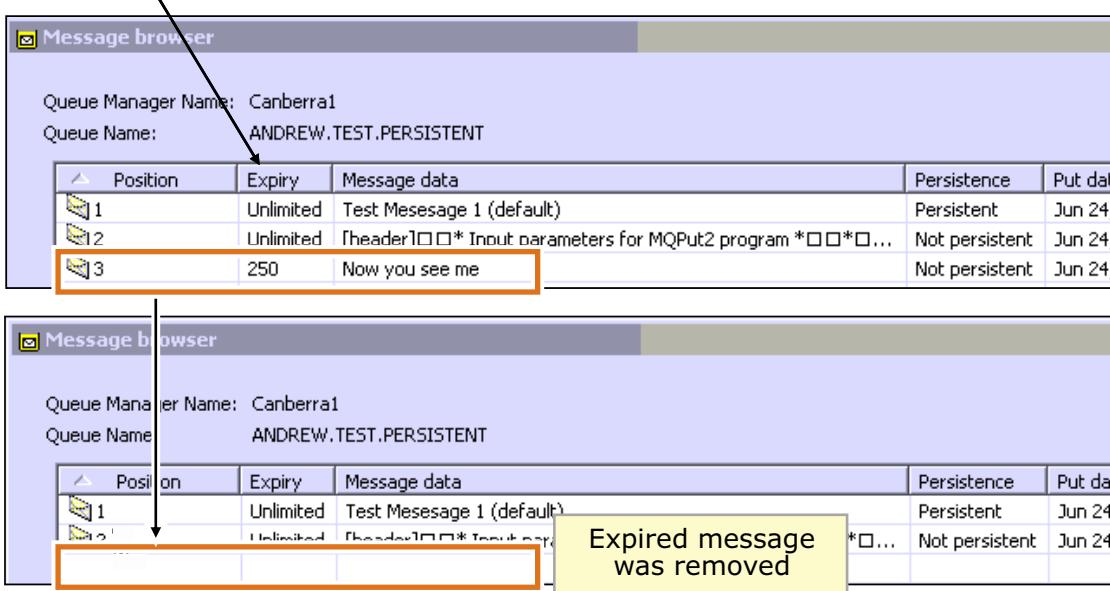
If the queue manager is restarted while a nonpersistent message is on the queue, the message is discarded. This behavior is standard for nonpersistent messages. They have much lower memory and storage requirements because they are not stored in a log.

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

- Expired messages are removed from the system
- Application that puts the message on the queue sets the expiration
- Check **Expiry** property on message



Diagnosing problems

© Copyright IBM Corporation 2020

Figure 12-12. 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

Figure 12-13. 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.

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)                               TYPE(QUEUE)
  CURDEPTH(4)                                IPPROCS(0)
  :
  OPPROCS(1)                                 QTIME( , )
  UNCOM(YES)
```

```
DISPLAY QSTATUS(TESTQ1) TYPE(HANDLE)
AMQ8450: Display queue status details
  QUEUE(TESTQ1)                               TYPE(HANDLE)
  APPLTAG(c:\WMQ\bin\amqspput.exe)           APPLTYPE(USER)
  :
  OUTPUT(YES)                                PID(10880)
  QMURID(0.0)                                SET(NO)
  TID(1)
  URID(XA_FORMATID[ ] XA_GTRID[ ] XA_BQUAL[ ]) USERID(slw@localhost)
  URTYPE(QMGR)
```

[Diagnosing problems](#)

© Copyright IBM Corporation 2020

Figure 12-14. *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.

Checking for channel problems

- 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 12-15. Checking for channel problems

In some cases, you might suspect that a problem exists with the IBM MQ channels. The figure lists some tips for locating problems with channels.

Resolving 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 2020

Figure 12-16. Resolving channel triggering problems

If problems are encountered with triggering, check the following:

- 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 can ease debugging.

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 2020

Figure 12-17. 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 IBM 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.

Checking configuration files

- 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 12-18. Checking configuration files

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 IBM MQ configuration file **mqm.ini** exists.
- Ensure that the IBM 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.

12.2. Cluster rerouting

Cluster rerouting

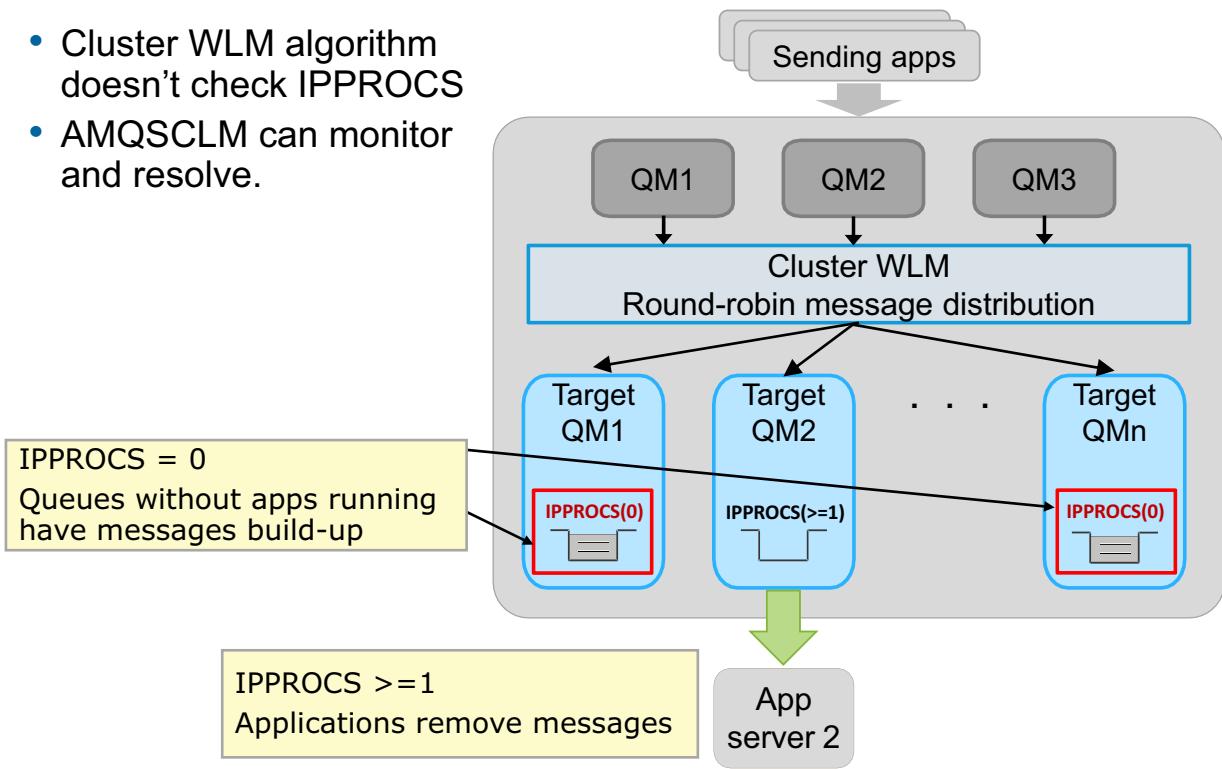
Diagnosing problems

© Copyright IBM Corporation 2020

Figure 12-19. Cluster rerouting

Cluster Workload Management (WLM)

- Cluster WLM algorithm doesn't check IPPROCS
- AMQSCLM can monitor and resolve.



Diagnosing problems

© Copyright IBM Corporation 2020

Figure 12-20. Cluster Workload Management (WLM)

Sending applications create messages to each of their queues in their queue managers. Those queues forward to a named queue in a cluster. The cluster WLM distributes the messages to the named queue in each of the queue managers defined in the cluster. Applications then retrieve and remove the messages from the named queue.

The Cluster WLM algorithm doesn't check whether consuming applications are connected to cluster queues. For instance, whether something is getting messages from them. You need to ensure that applications are consuming from all instances of a clustered queue to prevent messages building up. AMQSCLM can monitor and resolve.

Cluster Queue Monitoring sample program (AMQSCLM)

- Included with the product as a sample, including:
 - Precompiled executable
 - Source code
- Checks for queues without applications consuming messages
- Monitors queue property IPPROCS
- Reroutes messages within the cluster

[Diagnosing problems](#)

© Copyright IBM Corporation 2020

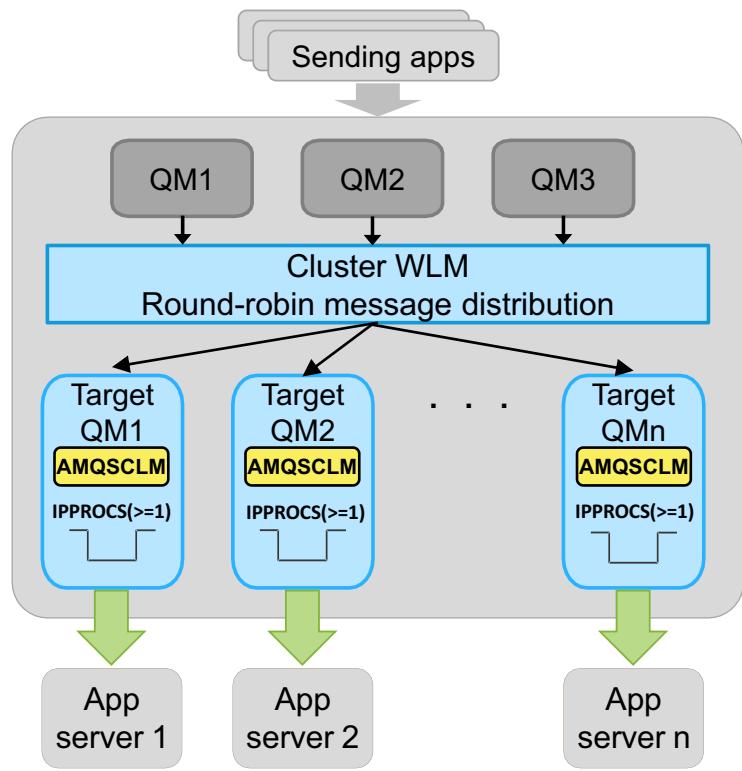
Figure 12-21. Cluster Queue Monitoring sample program (AMQSCLM)

More information at:

https://www.ibm.com/support/knowledgecenter/SSFKSJ_9.1.0/com.ibm.mq.dev.doc/q024620_.htm

Normal operation with AMQSCLM

- AMQSCLM running on each queue manager
- Applications working normally
- Messages are processed



Diagnosing problems

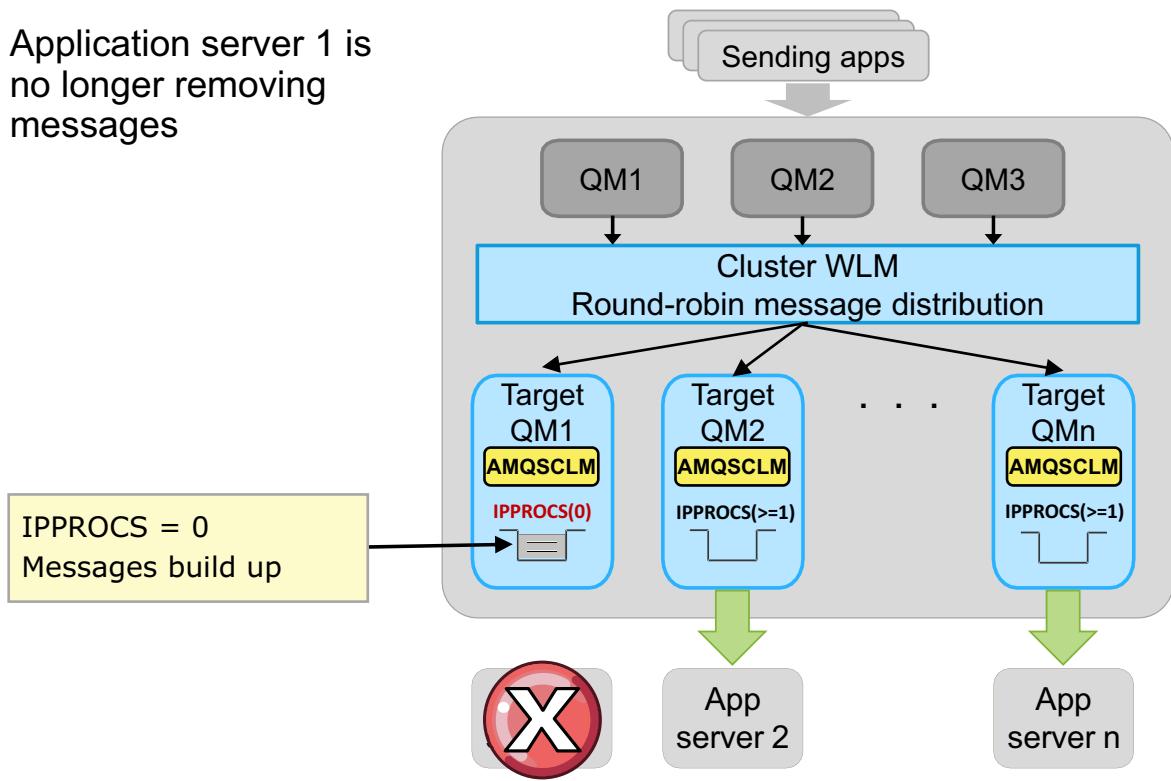
© Copyright IBM Corporation 2020

Figure 12-22. Normal operation with AMQSCLM

In normal operation, the messages are processed through all the queues in the cluster. There is no backup of messages, and each queue has at least one application pulling messages from it.

Application fails

- Application server 1 is no longer removing messages



Diagnosing problems

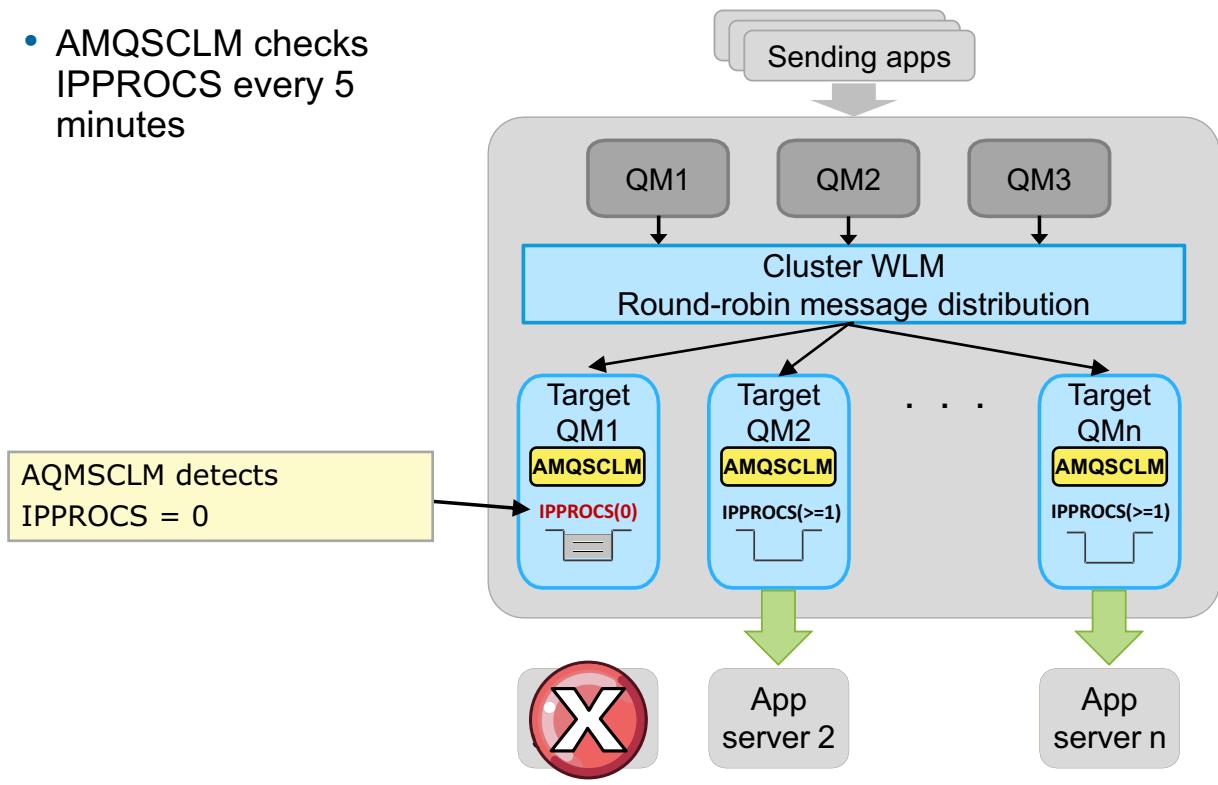
© Copyright IBM Corporation 2020

Figure 12-23. Application fails

When an application fails, it stops pulling messages from the queue. The IPPROCS value reduces, and if it was the only application running on that queue, the value becomes zero. Messages build up on that application as the cluster WLM does not notice this event on its own and continues to deliver messages to that queue.

AMQSCLM detects application failure

- AMQSCLM checks IPPROCS every 5 minutes



Diagnosing problems

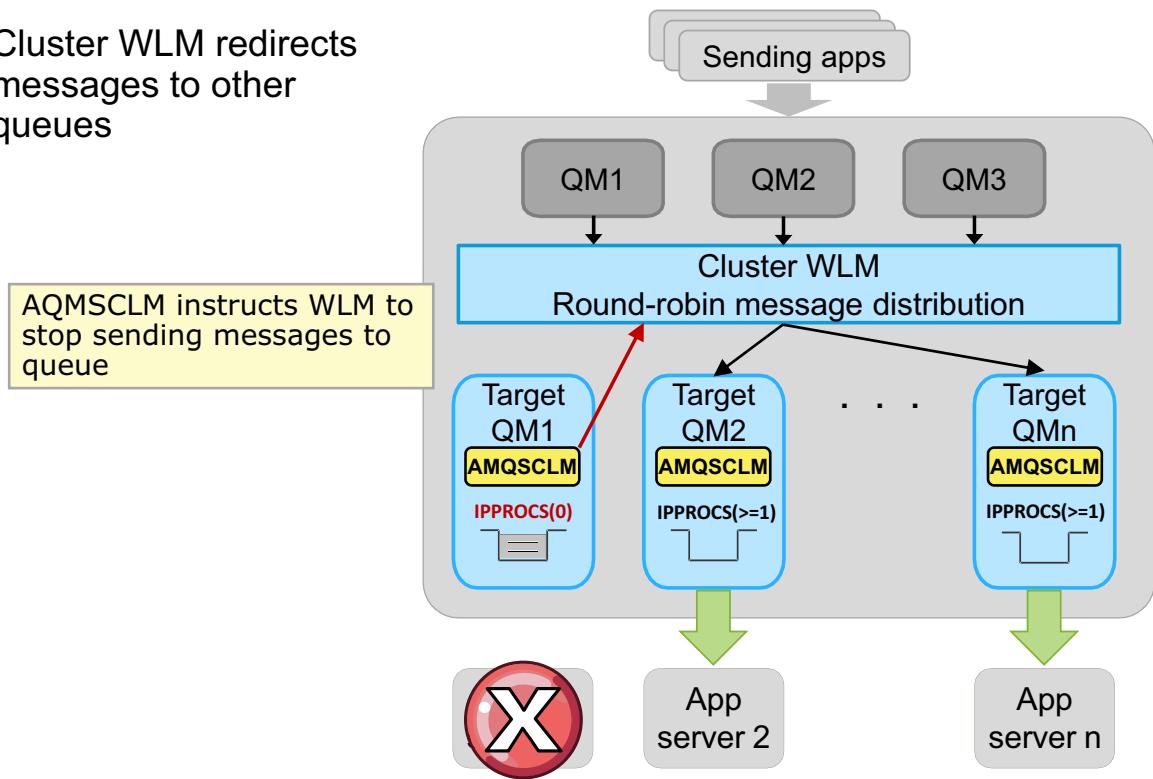
© Copyright IBM Corporation 2020

Figure 12-24. AMQSCLM detects application failure

The AMQSCLM program checks the IPPROCS value every 5 minutes and detects the failure.

AMQSCLM redirects messages

- Cluster WLM redirects messages to other queues



Diagnosing problems

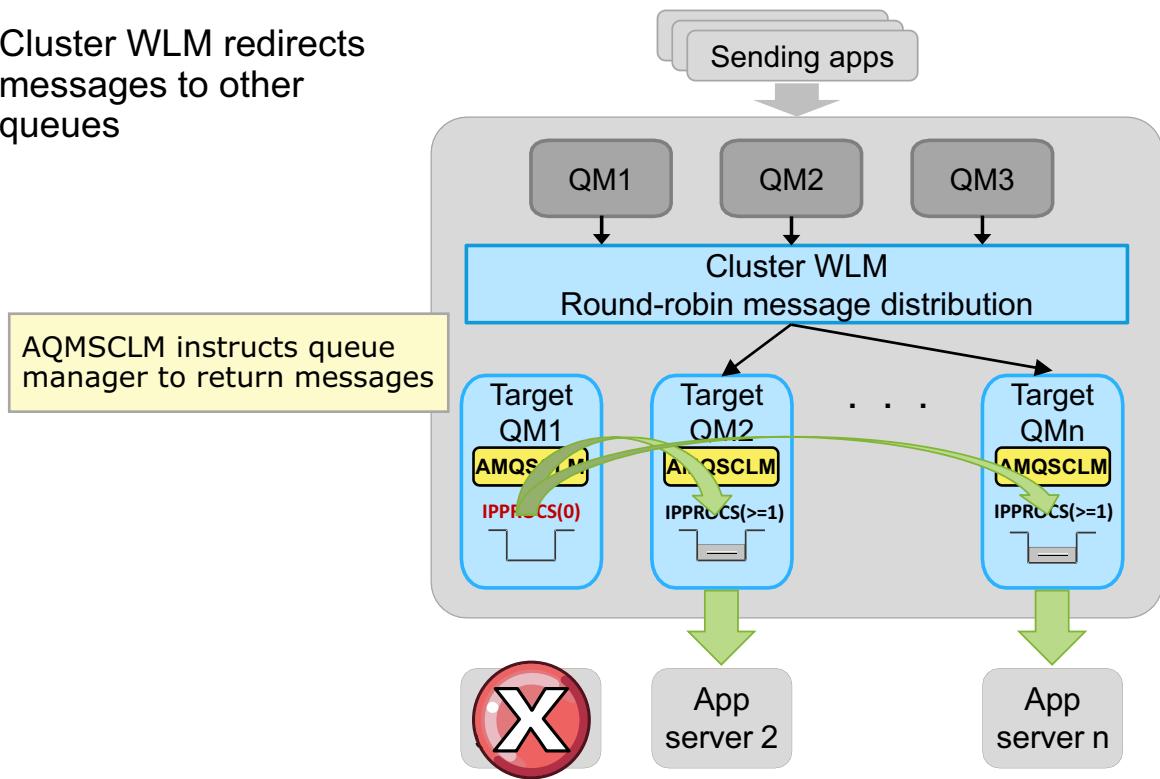
© Copyright IBM Corporation 2020

Figure 12-25. AMQSCLM redirects messages

The AMQSCLM instructs the cluster WLM to stop sending message to the queue manager. Messages are now only sent to the other queue managers.

AMQSCLM redistributes messages

- Cluster WLM redirects messages to other queues



Diagnosing problems

© Copyright IBM Corporation 2020

Figure 12-26. AMQSCLM redistributes messages

Optionally, the AMQSCLM instructs the cluster WLM to retrieve the messages from the queue manager with the failed application and redistribute them to the other queue managers.

12.3. First Failure Support Technology (FFST)

First Failure Support Technology (FFST)

© Copyright IBM Corporation 2020
Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Figure 12-27. First Failure Support Technology (FFST)

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 2020

Figure 12-28. First Failure Support Technology (FFST)

First Failure Support Technology (FFST) for IBM MQ provides information about events that, if an error occurs, can help IBM support personnel to diagnose the problem.

IBM 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, you might experience 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**

[Diagnosing problems](#)

© Copyright IBM Corporation 2020

Figure 12-29. FFST files

In IBM 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 IBM 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 platform)
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 2020

Figure 12-30. FFST report example

This figure shows an FFST report (an FDC file) that reports that an internal IBM 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.

12.4. Error logs

Error logs

© Copyright IBM Corporation 2020
Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Figure 12-31. Error logs

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 2020

Figure 12-32. IBM MQ error logs

IBM 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 12-33. 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 timestamp, 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 2020

Figure 12-34. IBM MQ AMQ messages

IBM 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 12-35. 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.

12.5. IBM MQ trace

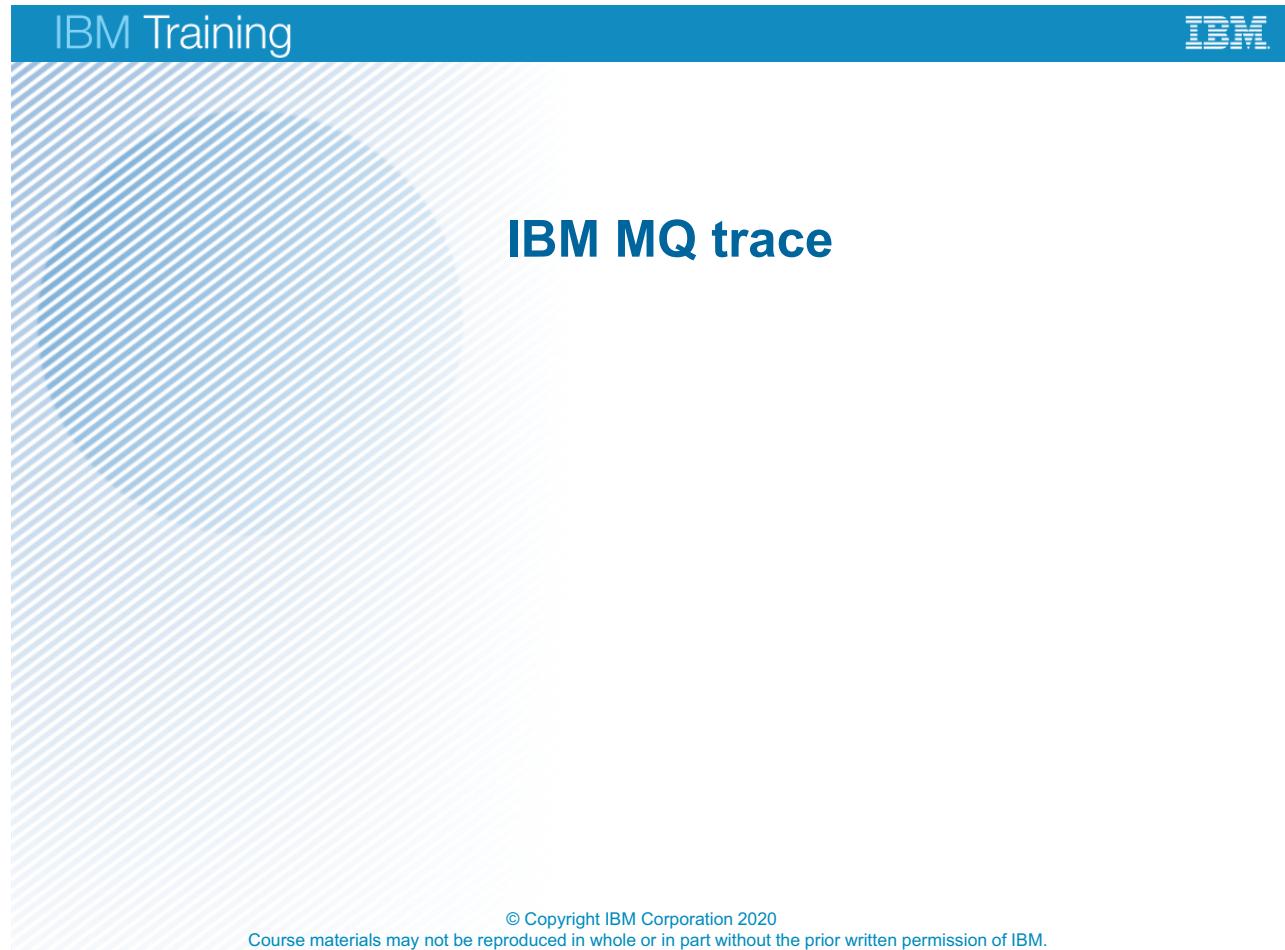


Figure 12-36. IBM MQ trace

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 2020

Figure 12-37. 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 12-38. 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 IBM 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
```

Diagnosing problems

© Copyright IBM Corporation 2020

Figure 12-39. The mqat.ini file

The **mqat.ini** file follows the same stanza key and parameter-value pair format as the **mqgs.ini** and **qm.ini** files.

The **AllActivityTrace** stanza specifies the level and frequency of reporting activity trace data by default for all activity trace.

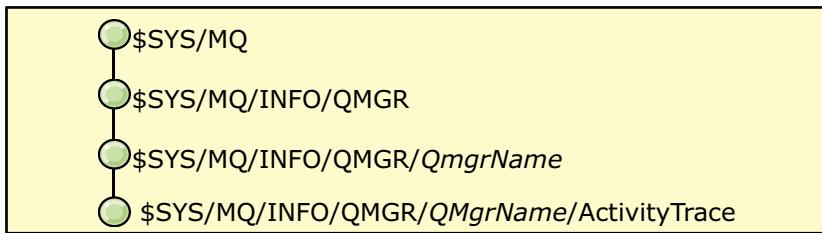
The **TraceLevel** identifies the amount of parameter detail to trace for each operation.

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 2020

Figure 12-40. 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.

IBM 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.

You can subscribe to an IBM MQ system topic to collect application activity trace information.

In IBM MQ, 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 12-41. Activity trace topics and subscriptions

You can subscribe to an IBM 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 **amqspput**:

```
$SYS/MQ/INFO/QMGR/QMGR1/ActivityTrace/AppName/amqspput.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 12-42. 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/App1Name/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 2020

Figure 12-43. Activity trace sample program

You can use the IBM 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 2020

Figure 12-44. 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 IBM 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 2020

Figure 12-45. Trace commands

The trace files are written to specific locations on the file system.

The figure shows some examples of starting an IBM 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.

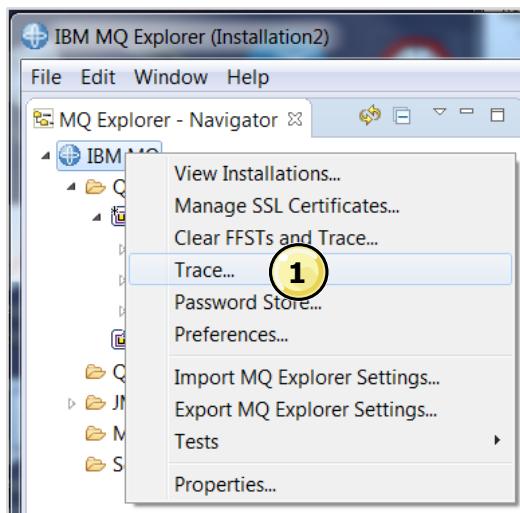
SSL trace files are named **AMQ.SSL.TRC** and **AMQ.SSL.TRC.1**. You cannot format SSL trace files; send them unchanged to IBM Support.

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.

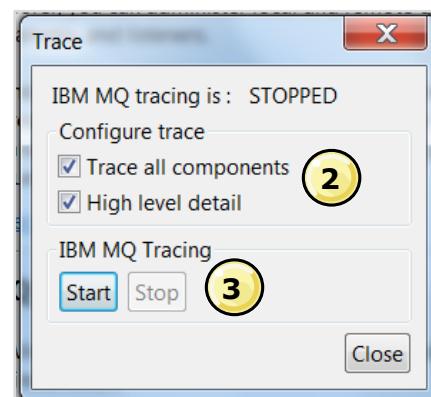
If you specify a wrapping trace, then each time a trace (**TRC**) file reaches the specified size limit, MQ renames it to use a **TRS** extension and starts a new trace file. The trace formatter can convert both files to a single formatted file, but only if you format the **TRC** and **TRS** files at 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 2020

Figure 12-46. 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\amqsgt.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 2020

Figure 12-47. Example trace on Windows

The figure shows an example of a trace on Windows

Use the `dspmqtrc` control command to format the trace report before viewing.

For example, to format all trace files in the current directory, type: `dspmqtrc *.TRC`

## 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

[Diagnosing problems](#)

© Copyright IBM Corporation 2020

*Figure 12-48. Unit summary*

## Exercise: Running an IBM MQ trace

Diagnosing problems

© Copyright IBM Corporation 2020

Figure 12-49. Exercise: Running an IBM MQ trace

## Exercise introduction

- Start and stop an IBM MQ trace
- Analyze the output from the IBM MQ trace
- Handle dead-letter messages



[Diagnosing problems](#)

© Copyright IBM Corporation 2020

*Figure 12-50. Exercise introduction*

---

# Unit 13. 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
- Lab exercise

## 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 2020

*Figure 13-1. Unit objectives*

## Unit topics

- Logs and recovery
- Backing up the IBM MQ file system
- Backing up object definitions

Backing up and restoring IBM MQ messages and object definitions

© Copyright IBM Corporation 2020

*Figure 13-2. Topics*

## 13.1. Logs and recovery

## Logs and recovery

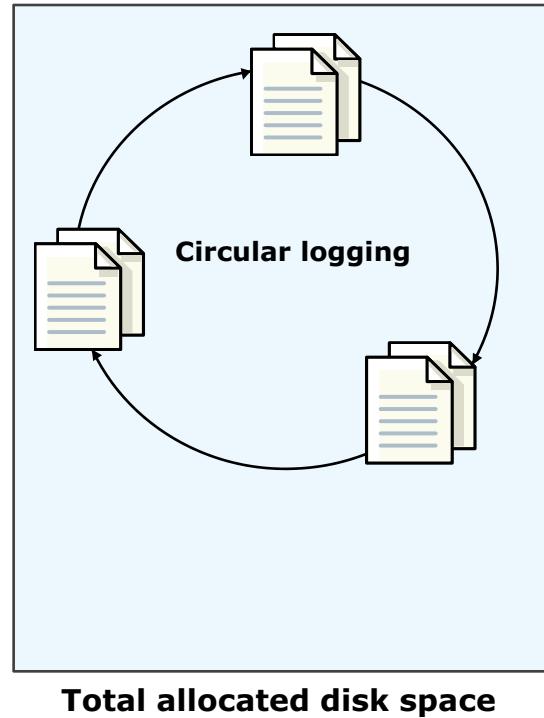
Backing up and restoring IBM MQ messages and object definitions

© Copyright IBM Corporation 2020

*Figure 13-3. Logs and recovery*

## Circular logs

- Primary log files continually reused in a ring
- Space is permanently allocated
- Restart recovery
- Amount of disk space that is required for the log does not increase with time



*Figure 13-4. 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, IBM 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.

**Restart recovery:** Enough information held in the log files to rebuild IBM MQ resources to the level that they were at before the queue manager stopped

When you specify the type of logging, you can also specify the size and location of the log files. Otherwise, the default values as specified in the IBM MQ configuration file are used.

## Linear logs

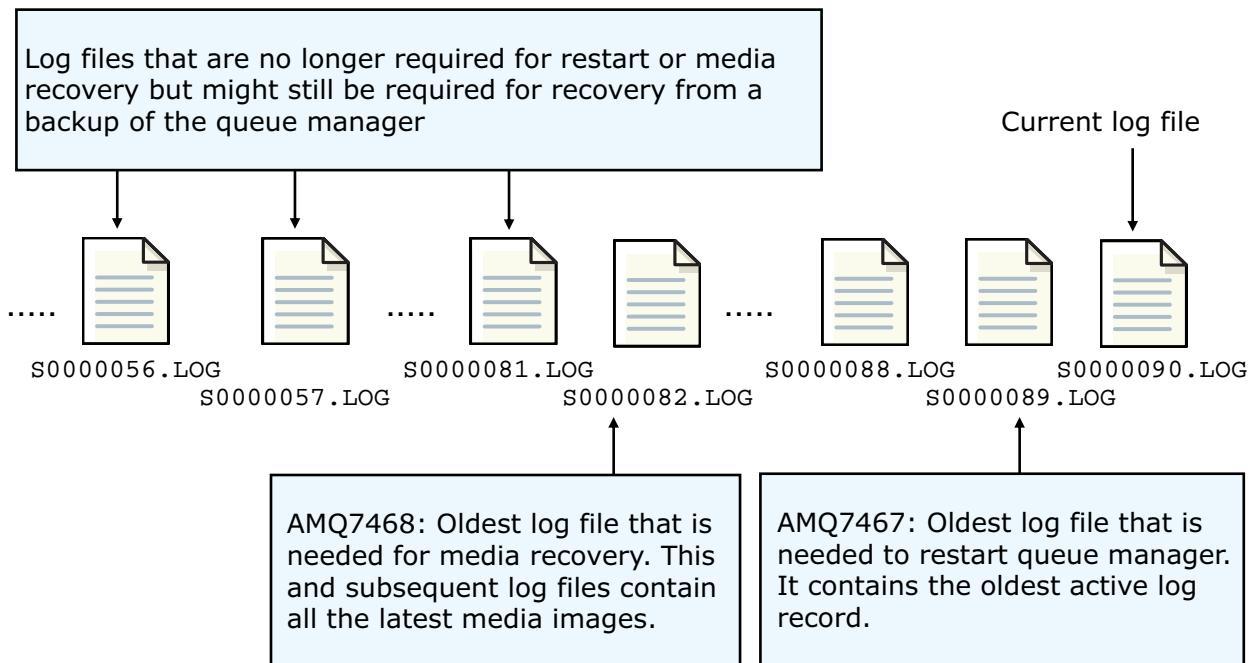


Figure 13-5. Linear logs

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.

## 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 2020

Figure 13-6. Types of logs

IBM 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.

IBM 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 versus linear

|                | <b>Circular</b>                                                       | <b>Linear</b>                                                                                                         |
|----------------|-----------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------|
| Recovery       | Can fix in-flight commands that were incomplete<br>Minimum overhead   | Same. And can also recover damaged queues                                                                             |
| Performance    | Logs allocated once and reused                                        | Logs must be allocated on an ongoing basis, causing performance issues                                                |
| Administration | Basically no administrative effort required during normal operations. | Administrators must ensure inactive log files are deleted or archived                                                 |
| Risk           | Loss of a queue file means loss of all messages on that queue.        | If log files are not managed regularly, a normally running queue manager will eventually exhaust available disk space |

Backing up and restoring IBM MQ messages and object definitions

© Copyright IBM Corporation 2020

Figure 13-7. Circular versus linear

## 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 IBM MQ objects

[Backing up and restoring IBM MQ messages and object definitions](#)

© Copyright IBM Corporation 2020

Figure 13-8. 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 `nnnnnnn` 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 IBM MQ objects

## Recovering persistent messages

- To restart, a queue manager requires:
  - Log records written since the last checkpoint
  - Log records written by transactions that were active when the 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 2020

*Figure 13-9. 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.

## 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 `rfrmqobj`

Figure 13-10. 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

**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 13-11. 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.

If the queues contain many messages, it can take a long time to run.

## 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 *`

*Figure 13-12. 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 IBM 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

|                         |                                                            |
|-------------------------|------------------------------------------------------------|
| <b>* or all</b>         | All object types                                           |
| <b>authinfo</b>         | Authentication information object for SSL channel security |
| <b>channel or chl</b>   | Channels                                                   |
| <b>clntconn or clcn</b> | Client connection channels                                 |
| <b>clchltab</b>         | Client channel table                                       |
| <b>listener or lstr</b> | Listener                                                   |
| <b>namelist or nl</b>   | Namelists                                                  |
| <b>process or prcs</b>  | Processes                                                  |
| <b>queue or q</b>       | All types of queue                                         |
| <b>qalias or qa</b>     | Alias queues                                               |
| <b>qlocal or ql</b>     | Local queues                                               |
| <b>qmodel or qm</b>     | Model queues                                               |
| <b>qremote or qr</b>    | Remote queues                                              |
| <b>service or srvc</b>  | Service                                                    |
| <b>syncfile</b>         | Channel synchronization file                               |
| <b>topic or top</b>     | Topics                                                     |

Backing up and restoring IBM MQ messages and object definitions

© Copyright IBM Corporation 2020

Figure 13-13. 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.

## 13.2. Backing up the IBM MQ file system

## Backing up the IBM MQ file system

Backing up and restoring IBM MQ messages and object definitions

© Copyright IBM Corporation 2020

*Figure 13-14. Backing up the IBM MQ file system*

## 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 2020

Figure 13-15. The need to back up

Periodically backing up IBM 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 2020

Figure 13-16. 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 IBM 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 IBM MQ on UNIX and Linux, you can preserve ownership by using the `tar` command.

Restart the queue manager after the backup is complete.

## 13.3. Backing up object definitions

## Backing up object definitions

Backing up and restoring IBM MQ messages and object definitions

© Copyright IBM Corporation 2020

*Figure 13-17. Backing up object definitions*

## 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 2020

Figure 13-18. Overview of IBM MQ objects

This figure is a review of the various IBM 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 2020

Figure 13-19. 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 IBM 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 IBM 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 2020

Figure 13-20. Capturing IBM MQ object definitions

You can capture object definitions by using MQSC and the `dmpmqcfg` command.

- Save the IBM 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 2020

Figure 13-21. 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 2020

Figure 13-22. 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
- IBM MQ set authority access (`setmqaut`) statements for Linux, UNIX, and Windows queue managers
- On Linux, grant IBM 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
```

Figure 13-23. 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 IBM MQ file system, the queue manager must be stopped.

When you back up an IBM 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 2020

Figure 13-24. 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) +
PLATFORM(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 2020

Figure 13-25. 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: JMSMITH
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 MYQMR -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 2020

Figure 13-26. 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 2020

*Figure 13-27. Overview of security definitions*

The `setmqaut` commands that the `dmpmqcfg -o setmqaut` command generates are IBM 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 IBM 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 2020

*Figure 13-28. 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
```

Figure 13-29. Backing up security definitions

IBM 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                                                                                                                                                                         |
|----------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>-n Profile</code>    | Name of the profile for which to dump authorizations                                                                                                                                |
| <code>-l</code>            | Dump only the profile name and type                                                                                                                                                 |
| <code>-a</code>            | Generate set authority commands                                                                                                                                                     |
| <code>-t ObjectType</code> | The type of object for which to dump authorizations<br>Possible values are: authinfo, channel, clntconn,<br>listener, namelist, process, queue, qmgr,<br>rqmname, service, or topic |
| <code>-e</code>            | Display all profiles that are used to calculate the cumulative<br>authority that the entity has for the object that is specified in<br><code>-n Profile</code>                      |
| <code>-x</code>            | Display all profiles with the same name as specified in<br><code>-n Profile</code>                                                                                                  |

Backing up and restoring IBM MQ messages and object definitions

© Copyright IBM Corporation 2020

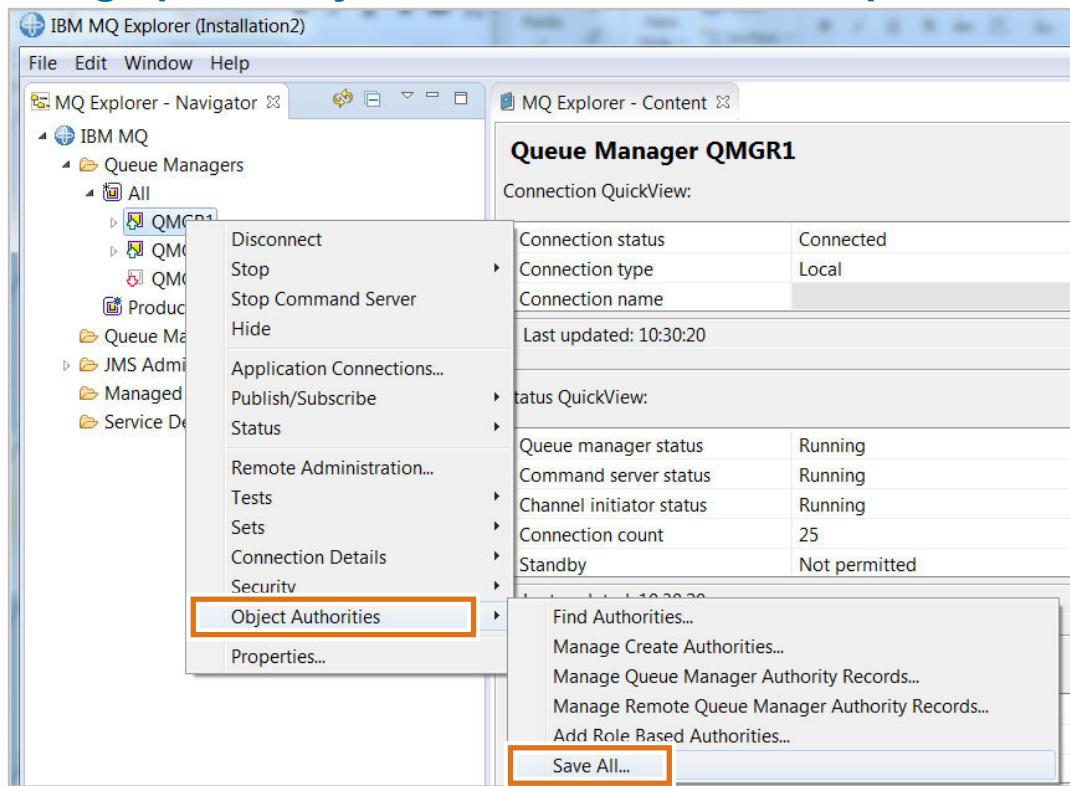
Figure 13-30. `dmpmqaut` command

You can use the IBM 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 2020

Figure 13-31. 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.

## 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 2020

*Figure 13-32. 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.
3. Which type of logging provides for media recovery?
  - A. Circular
  - B. Linear

Backing up and restoring IBM MQ messages and object definitions

© Copyright IBM Corporation 2020

Figure 13-33. Review questions

Write your answers here:

1.

## 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.
3. Which type of logging provides for media recovery?  
A. Circular  
B. Linear

Backing up and restoring IBM MQ messages and object definitions

© Copyright IBM Corporation 2020

Figure 13-34. Review answers



## Exercise: Using a media image to restore a queue

Backing up and restoring IBM MQ messages and object definitions

© Copyright IBM Corporation 2020

Figure 13-35. Exercise: Using a media image to restore a queue

## Exercise introduction

- Capture an object media image
- Recreate an IBM MQ object from an object media image



Backing up and restoring IBM MQ messages and object definitions

© Copyright IBM Corporation 2020

*Figure 13-36. Exercise introduction*

## Exercise: Backing up and restoring IBM MQ object definitions

Backing up and restoring IBM MQ messages and object definitions

© Copyright IBM Corporation 2020

*Figure 13-37. Exercise: Backing up and restoring IBM MQ object definitions*

## Exercise introduction

- Back up object definitions of a queue manager
- Upload object definitions to another queue manager



Backing up and restoring IBM MQ messages and object definitions

© Copyright IBM Corporation 2020

*Figure 13-38. Exercise introduction*

# Unit 14. High availability

## Estimated time

01:00

## Overview

In this unit, you learn about the IBM MQ high availability solutions.

## How you will check your progress

- Review questions
- Hands-on exercise

## References

IBM Knowledge Center for IBM MQ V9.1

## Unit objectives

- Plan for using high availability systems with IBM MQ
- Configure and manage a multi-instance queue manager

High availability

© Copyright IBM Corporation 2020

*Figure 14-1. Unit objectives*

In this unit, you explore some of the IBM MQ high availability options, and you see how to configure and manage multi-instance queue managers.

## Topics

- HA overview for IBM MQ
- Multi-instance queue managers

High availability

© Copyright IBM Corporation 2020

Figure 14-2. Topics

These are the topics in this unit.

## 14.1. HA overview for IBM MQ

## HA overview for IBM MQ

High availability

© Copyright IBM Corporation 2020

Figure 14-3. HA overview for IBM MQ

Your first topic is an overview of the high availability options in IBM MQ.

## Messaging system availability in IBM MQ

- Redundancy achieved through application client connectivity
  - Multiple active options available for applications to connect
- IBM MQ cluster routing
  - Ability to route messages around failures
- IBM MQ message high availability
  - Messages are not locked to a single runtime and quickly available elsewhere

High availability

© Copyright IBM Corporation 2020

Figure 14-4. Messaging system availability in IBM MQ

In a messaging system, you need to address three main areas to achieve high availability: redundancy, routing, and message availability.

Redundancy is having multiple options to connect your applications to the messaging system. Within the IBM MQ system, redundancy is achieved through application client connectivity where you decouple the application from a single queue manager and allow it to work with a set of queue managers.

In routing, you look at how messages can reach their destination even when failures occur. Increased routing availability is seen through IBM MQ cluster routing.

IBM MQ message high availability is achieved through avoiding a breakdown when a single queue manager fails. Message availability gives you multiple avenues to retrieve your messages so that you are not bound to a single runtime. Implementing all three of these areas produce a highly available IBM MQ system.

## Application client connectivity (redundancy)

- Decouple the applications from queue managers
- Applications locally bound to a queue manager limits the availability of the solution.
- Run applications remote from the queue managers
  - Always connect as IBM MQ clients
- Enables higher availability

High availability

© Copyright IBM Corporation 2020

Figure 14-5. Application client connectivity (redundancy)

In a basic IBM MQ setup, the application is locally bound to the queue manager. One part of creating a highly available system is to decouple each application from its queue manager and have the applications run remotely connecting as IBM MQ clients.

## Application client connectivity (redundancy)

- Process to decouple the application from the queue manager
  - Step 1: Connect the application as a client
    - Creates the ability to support solutions where a queue manager might failover between systems.
    - Splits the application and queue manager onto separate systems
    - Improved maintenance / fewer system conflicts
    - Reduction of restart times for each component
    - Use of client auto-reconnect for smooth queue manager restarts
  - Step 2: Allow the application to connect to a set of queue managers
    - Applications can continue to work with IBM MQ while a queue manager is down
    - Multiple options for connecting to the queue managers: network routing, connection name lists, Client Channel Definition Tables (CCDT)

High availability

© Copyright IBM Corporation 2020

Figure 14-6. Application client connectivity (redundancy)

To decouple the applications, you first need to connect the application as a client and preferably setup auto-reconnect when of a queue manager restarts. Then, you must implement a method for the application to have a set of queue managers to which it can connect. Creating a CCDT, a Client Channel Definition Tables, is a popular method to define the queue manager sets. Other options include defined network routing and constructing connection name lists.

## Client Channel Definition Tables (CCDT)

- Provide encapsulation and abstraction of connection information for applications, hiding the IBM MQ architecture and configuration from the application.
- Enable security, high availability, and workload balancing of clients.
- Applications can connect to an abstracted “queue manager” name (wildcards can be used instead of one specific name)
- CCDT defines whether the application connects to a single or a group of queue managers.
- Across a group, selection can be ordered or randomized and weighted.

High availability

© Copyright IBM Corporation 2020

Figure 14-7. Client Channel Definition Tables (CCDT)

The Client Channel Definition Tables, CCDT, provide the abstraction to the connection between a client and the queue managers. The client is only configured with the CCDT name, which can be a single queue manager or a group, either defined exactly or by using a naming pattern. When established as a group the selection of which queue manager is used from that group can be ordered or random. As multiple clients attach to a group, workload balancing is enacted to improve performance.

## IBM MQ Cluster Routing (routing)

- IBM MQ Clusters route messages based on availability
- Provides multiple potential targets for any message
- This solution improves the availability by always providing an option to process new messages.
- Routes new and reroutes old messages based on the availability of the channels and queue managers
- Route messages to active consuming applications
- Available on all supported IBM MQ platforms

High availability

© Copyright IBM Corporation 2020

Figure 14-8. IBM MQ Cluster Routing (routing)

The routing of messages is essential to maintaining system availability. By having an IBM MQ cluster, multiple routes are defined so that if one route fails, alternates exist. A queue manager in a cluster can route new and old messages based on the availability of the channels, routing messages to running queue managers.

This solution improves the availability by always providing an option to process new messages. This allows applications to always access the messages through an active and available queue manager.

## Message High Availability

- No single point of failure for messages
- Active / passive messages (IBM MQ Distributed HA solutions)
  - Messages are highly available, through replication
  - Only one runtime is the leader and can access the messages at a time
  - A failure results in a new leader assigned
- Active / active messages – *IBM MQ for z/OS Shared Queues*

High availability

© Copyright IBM Corporation 2020

Figure 14-9. Message High Availability

Your third pillar of high availability is to ensure that there is no single point of failure for the messages themselves. The messages are then highly available. This is accomplished through either replication in an active-passive queue solution, or with active-active shared queues in z/OS.

## IBM MQ Distributed HA solutions

- Externally managed
  - System-managed HA
  - Multi-instance queue managers
- IBM MQ Managed
  - IBM MQ Appliance
  - RDQM (Replicated Data Queue Managers) – *Linux only*

High availability

© Copyright IBM Corporation 2020

Figure 14-10. IBM MQ Distributed HA solutions

Distributed solutions for IBM MQ are grouped into two categories: externally managed and IBM MQ managed. Our MQ managed options presented are to install MQ appliance or to use RDQM, the newly available replicated data queue managers, however, the RDQM option is only available on Linux. The externally managed options are multi-instance queue managers, which you will see more detail of in the next topic, and system managed options.

## System-managed HA

- The HA manager monitors the IBM MQ system
- Requirements:
  - IBM MQ system in a container or VM
  - External, highly available, storage
- The queue manager is unaware of the HA system
- On detecting a failure it will:
  - Start a new system
  - Remount storage
  - Reroute network traffic
- Availability depends on speed to detect problems and to restart all layers of the system required (for example, VM and queue manager)
- Examples of HA Clusters:
  - IBM PowerHA cluster – *AIX*
  - MS Cluster Server (MSCS)
  - Veritas InfoScale
  - HP Serviceguard – *Linux*

High availability

© Copyright IBM Corporation 2020

Figure 14-11. System-managed HA

A system managed HA controls the queue managers from above and as such the queue managers are not aware of the HA system in use. The HA system is responsible to detect failures and responsible for such tasks as starting new systems as needed, reroute network traffic to avoid failures, and managing the connection to the external storage. To accomplish this the MQ system must be configured as a virtual machine or in a container and have highly available external storage. IBM, Microsoft, Veritas and others have HA clusters that can operate with MQ and provide this service.

## Microsoft Cluster Service (MSCS)

- Support integrated with IBM MQ
- Capabilities:
  - Active-active configuration
  - Unit of failure is queue manager (with one or more queue managers per MSCS group)
  - Automatic queue manager registration on standby node
  - Security and registration synchronization
  - Supports IBM MQ custom services
- Utility programs:
  - Register the resource type                    `haregtyp /r`
  - Unregister the resource type                `haregtyp /u`
  - Remove queue manager from node        `hadltmqm /m QmgrName`
  - Check and save setup details            `amqmsysn`
  - Move a queue manager to MSCS storage:  
`hamvmqm /m QmgrName /dd "DataDirectory" /ld "LogDirectory"`

High availability

© Copyright IBM Corporation 2020

Figure 14-12. Microsoft Cluster Service (MSCS)

The Microsoft Cluster Service supports IBM MQ and can have active-active queue managers along with many other capabilities.

## IBM MQ Appliance

- A pair of IBM MQ Appliances is connected together and configured as an HA group
- Queue managers that are created on one appliance can be automatically replicated, along with all the IBM MQ data
- Appliances monitor each other
- Automatic failover, plus manual failover for migration or maintenance
- Independent failover for queue managers so both appliances can run workload (active / active load)
- No persistent data loss on failure
- No external storage

High availability

© Copyright IBM Corporation 2020

Figure 14-13. IBM MQ Appliance

An MQ appliance is specialized hardware added to your system that houses queue managers. Having two of these connected together can form a highly available group. The appliances monitor each other and have several failover options.

## Replicated Data Queue Managers (RDQM)

- *Linux only*
- IBM MQ Advanced HA solution
- No need for a shared file system or HA cluster
- IBM MQ configures the underlying resources to make setup and operations natural to an IBM MQ user
- Three-way replication
  - Spread the workload across multiple queue managers and distribute them across all three nodes
- Active/passive queue managers with automatic takeover
- Supports active/active usage of nodes
- Synchronous data replication for once and once only transactional delivery of messages
- Simple application setup

High availability

© Copyright IBM Corporation 2020

Figure 14-14. Replicated Data Queue Managers (RDQM)

One of the most exciting improvements in MQ is replicated data queue managers, commonly referred to as RDQM. This is an MQ Advanced HA solution that also has disaster recovery support. In RDQM there is no need for a shared file system or HA cluster, although it can work with MQ clusters to provide additional routing which may help work around some problems. In RDQM, MQ configures the underlying resources to provide the MQ user with a natural feel with setup and operations. The three-way replication spreads the workload across multiple queue managers and will distribute them across all three nodes. Automatic takeover is possible with active/passive queue managers, and per queue manager control supports active/active utilization of nodes. The per queue manager IP addressing also provides simple application setup. This new RDQM feature is a huge improvement and available only to Linux deployments.

## 14.2. Multi-instance Queue Managers

## Multi-instance Queue Managers

High availability

© Copyright IBM Corporation 2020

*Figure 14-15. Multi-instance Queue Managers*

In this topic, you will discuss multi-instance queue managers and how to configure them.

## Multi-instance queue managers

- Basic failover support without HA cluster
- Two instances of the same queue manager on different servers
  - Active instance owns the queue manager files and accepts connections from applications
  - Standby instance monitors the active instance
  - If active instance fails, standby instance restarts queue manager and becomes active
- One set of queue manager data that is kept in networked storage

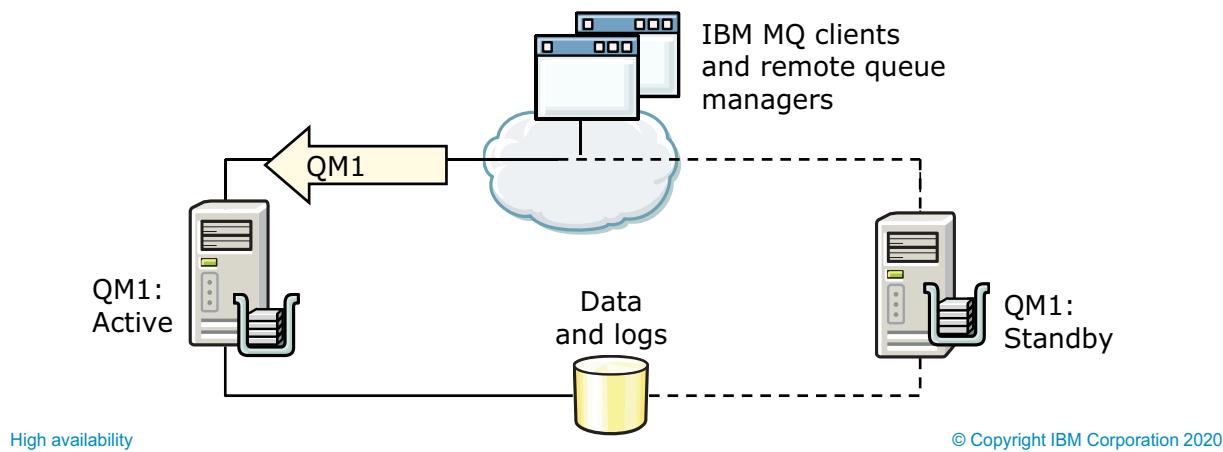


Figure 14-16. Multi-instance queue managers

Multi-instance queue managers provide basic failover support. In this instance, a duplicate queue manager is in standby mode and is restarted, activated, when the active queue manager fails. Both point to the same set of data and logs.

## Creating multi-instance queue managers

1. Set up shared file systems for queue manager data and logs
2. Create the queue manager on server 1 with pointers to shared stored for data and logs

Example:

```
crtmqm -md /shared/qmdata -ld /shared/qmlog QM1
```

3. Add configuration information for the queue manager to server 2

Example:

```
addmqinf -v Name=QM1 -v Directory=QM1 -v Prefix=/var/mqm
-v DataPath=/shared/qmdata/QM1
```

4. Start the (active) queue manager on server 1.

Example:

```
strmqm -x QM1
```

5. Start the (standby) queue manager on server 2.

Example:

```
strmqm -x QM1
```

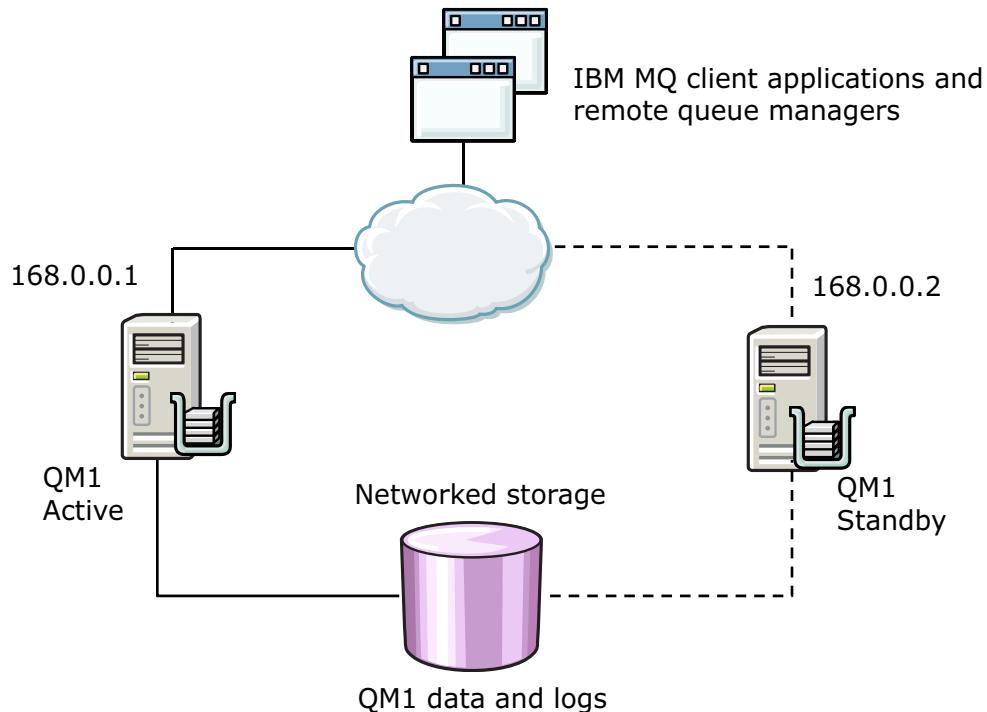
[High availability](#)

© Copyright IBM Corporation 2020

Figure 14-17. Creating multi-instance queue managers

To create the basic setup, one first creates the queue manager normally. Then on the second server, instead of the normal creation of the queue manager, the command addmqinf is used to create a reference to the queue manager data and logs on the network storage. Both are then started normally, and the second queue manager started is placed in standby mode.

## Standby configuration



High availability

© Copyright IBM Corporation 2020

Figure 14-18. Standby configuration

This diagram shows the normal running operation with QM1 server on the left, noted with IP address .1, being active, and the QM server on the right, on .2, with the queue manager in standby mode.

## Standby failover

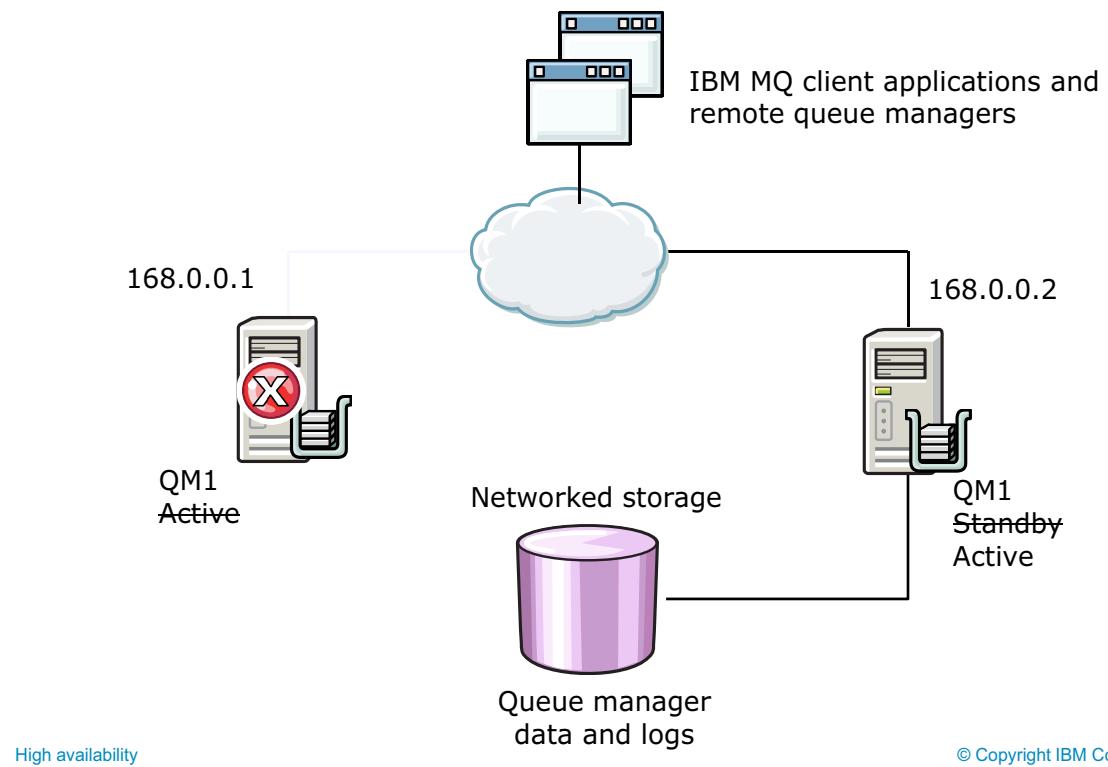


Figure 14-19. Standby failover

When a failure occurs in QM1 server on the left, the queue manager on the QM1 server on the right is restarted and becomes active.

## Handling multiple IP addresses in multi-instance

- IP address of the queue manager changes active queue manager moves during failover
- IBM MQ channel configuration needs way to select address
- Multiple IP address options:
  - Connection name syntax is extended to a comma-separated list  
Example: `CONNNAME ('168.0.0.1,168.0.0.2')`
  - Use external IP address translation or intelligent router

High availability

© Copyright IBM Corporation 2020

Figure 14-20. Handling multiple IP addresses in multi-instance

As you've seen, the servers on the multi-instance queue managers have different IP addresses. The clients and MQ channels need to be configured with the connection information for all the queue managers. When a failure occurs, and the standby is active, clients and channels are automatically reconnected to the newly activated queue manager.

## Multi-instance queue manager administration

- IBM MQ is *not* an HA cluster coordinator
  - Coordination of other resources requires an HA cluster
  - Queue manager services can be automatically started, but with limited control
- System administrator is responsible for restarting another standby instance when failover occurs
- All queue manager administration is on the active instance
  - Use `dspmq -x` command to identify active and standby instances

Example:

```
$ hostname
 server2
$ dspmq -x
QMNAME (QM1) STATUS (Running as standby)
INSTANCE (server1) MODE (Active)
INSTANCE (server2) MODE (Standby)
```

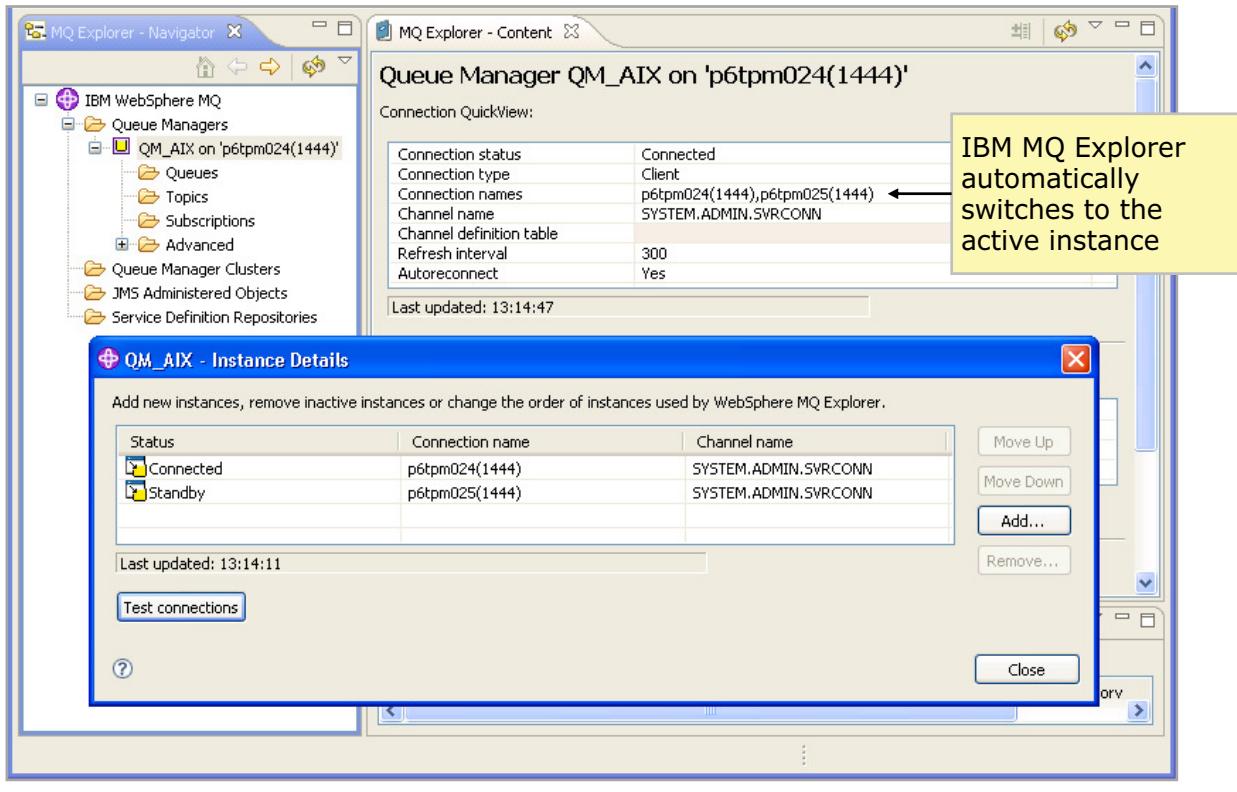
[High availability](#)

© Copyright IBM Corporation 2020

Figure 14-21. Multi-instance queue manager administration

Finding the current status of the queue managers in multi-instance can easily be shown by checking on the active server. Note that when the failure occurs, the administrator must place another server in standby mode.

## Managing multi-instance queue manager in IBM MQ Explorer



High availability

© Copyright IBM Corporation 2020

Figure 14-22. Managing multi-instance queue manager in IBM MQ Explorer

Here you see the details on the multi-instance queue manager shown from MQ Explorer.

## Unit summary

- Plan for using high availability systems with IBM MQ
- Configure and manage a multi-instance queue manager

High availability

© Copyright IBM Corporation 2020

Figure 14-23. Unit summary

In this unit, you explored the options to create a highly available system for IBM MQ, and showed the configuration of a multi-instance queue manager.

# Unit 15. 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. You also learn how to monitor performance with IBM MQ Console.

## How you will check your progress

- Review questions
- Lab exercise

## 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
- Monitor resources in the IBM MQ Console

## Topics

- IBM MQ statistics
- IBM MQ accounting data
- Displaying statistics and accounting data
- Real-time monitoring
- Configuring and tuning IBM MQ for performance
- Monitoring with IBM MQ Console



Monitoring and configuring IBM MQ for performance

© Copyright IBM Corporation 2020

*Figure 15-3. IBM MQ statistics*

## 15.1. IBM MQ statistics

## 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 IBM MQ object properties or by subscribing to IBM MQ statistics topics

*Figure 15-4. Collecting statistics and accounting data*

IBM MQ can generate statistics messages that record the information about the activities that occur in an IBM MQ system.

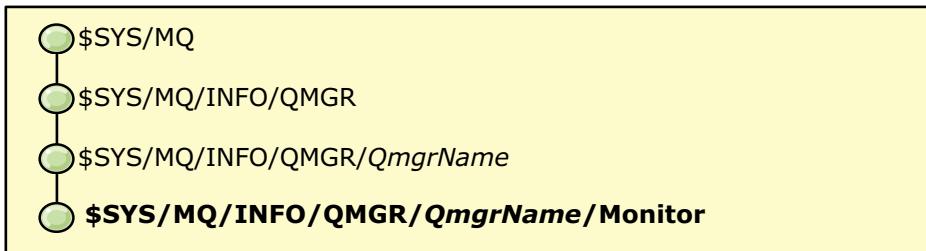
The information that is contained within statistics messages can be used for:

- 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 2020

Figure 15-5. Subscribing to statistics data (1 of 2)

IBM MQ Version 9.0 publishes statistics and activity trace information messages to IBM MQ system level topic strings.

In IBM 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`

Figure 15-6. Subscribing to statistics data (2 of 2)

## 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

Figure 15-7. 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

*Figure 15-8. IBM MQ statistics that are published to topics only*

This figure summarizes the IBM MQ statistics that are available only in published topics.

- **User CPU time** is the average percentage of time that is used by the CPU when it was in non-privileged code. Calculation is at 10-second intervals.
- **System CPU time** is the average percentage of time that is used by the CPU when it was in privileged code. Calculation is at 10-second intervals.
- **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. 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. 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.

## 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 15-9. Subscribing to statistics with the amqsrua program

The **amqsrua** command reports metadata 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

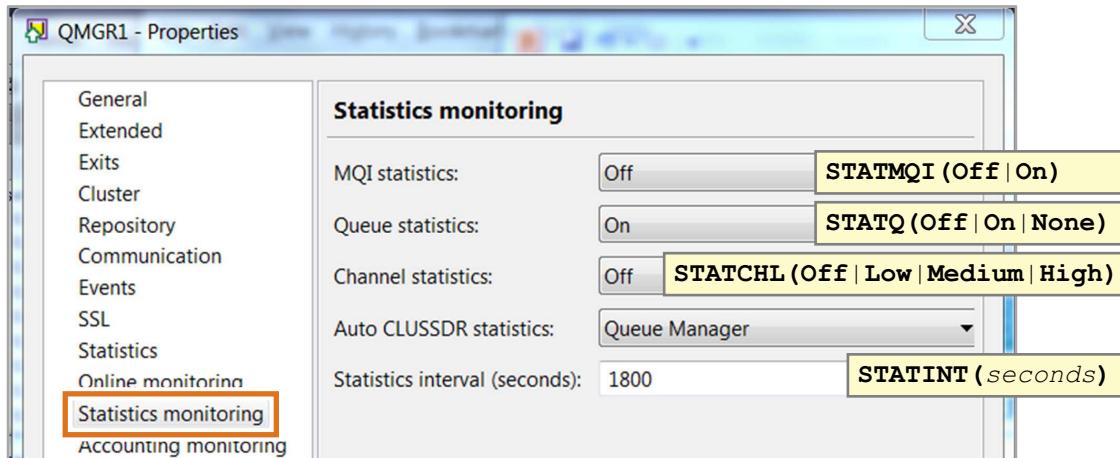
Publication received PutDate:20151014 PutTime:09180405
User CPU time - percentage estimate for queue manager 0.00%
System CPU time - percentage estimate for queue manager 0.00%
RAM total bytes - estimate for queue manager 200MB
```

Figure 15-10. 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.



## 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 2020

Figure 15-11. Queue manager statistics monitoring properties

You 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 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

*Figure 15-12. 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 IBM 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 15-13. Queue statistics

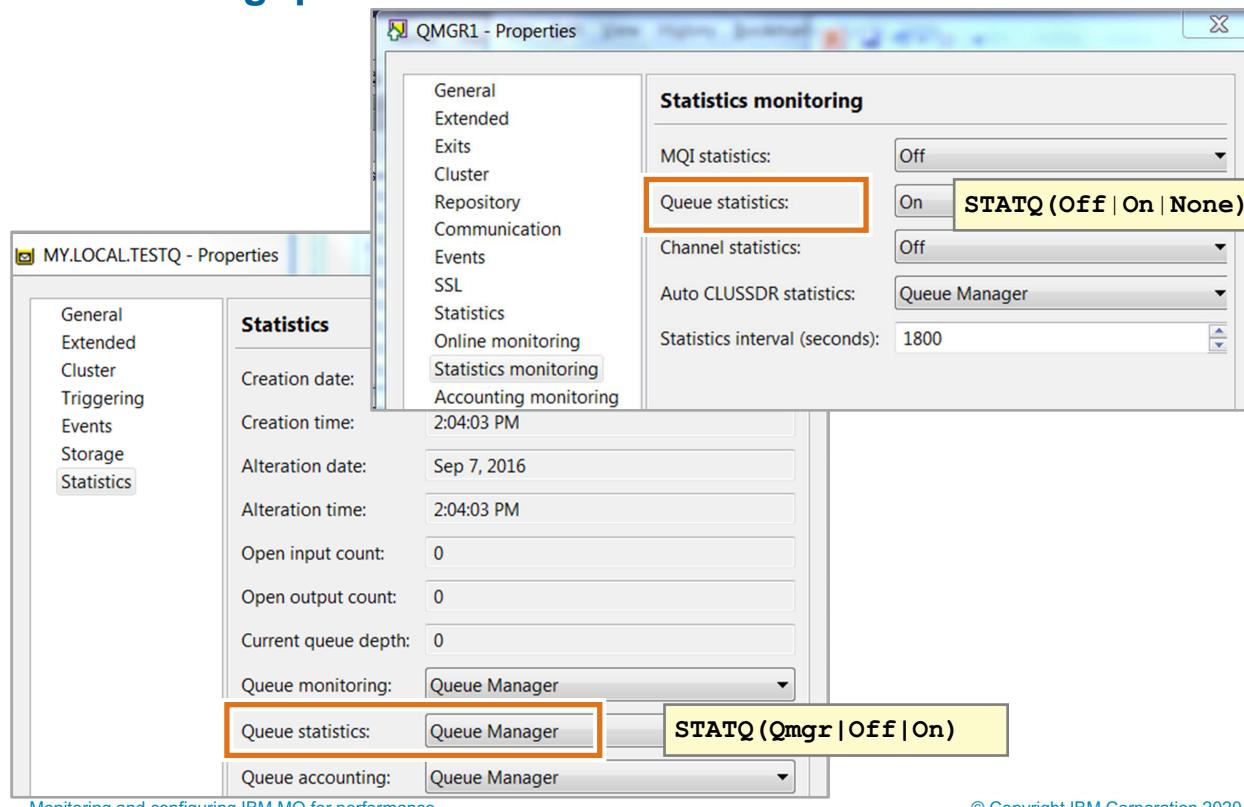
Queue statistics messages contain information about the activity of a queue during a configured interval. The information includes the number of messages that are put on and retrieved from the queue, and the total number of bytes that a queue processes.

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 IBM MQ STATQ topic, or by using IBM MQ Explorer.

# IBM Training

## Controlling queue statistics



Monitoring and configuring IBM MQ for performance

© Copyright IBM Corporation 2020

Figure 15-14. 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 QMGR** 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 **STATACL**S queue manager attribute to control automatically defined cluster-sender channels

*Figure 15-15. 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 IBM MQ objects, so they do not have attributes. To control automatically defined cluster-sender channels, use the queue manager attribute **STATACL**S. This attribute determines whether automatically defined cluster-sender channels within a queue manager are enabled or disabled for channel statistics information collection.

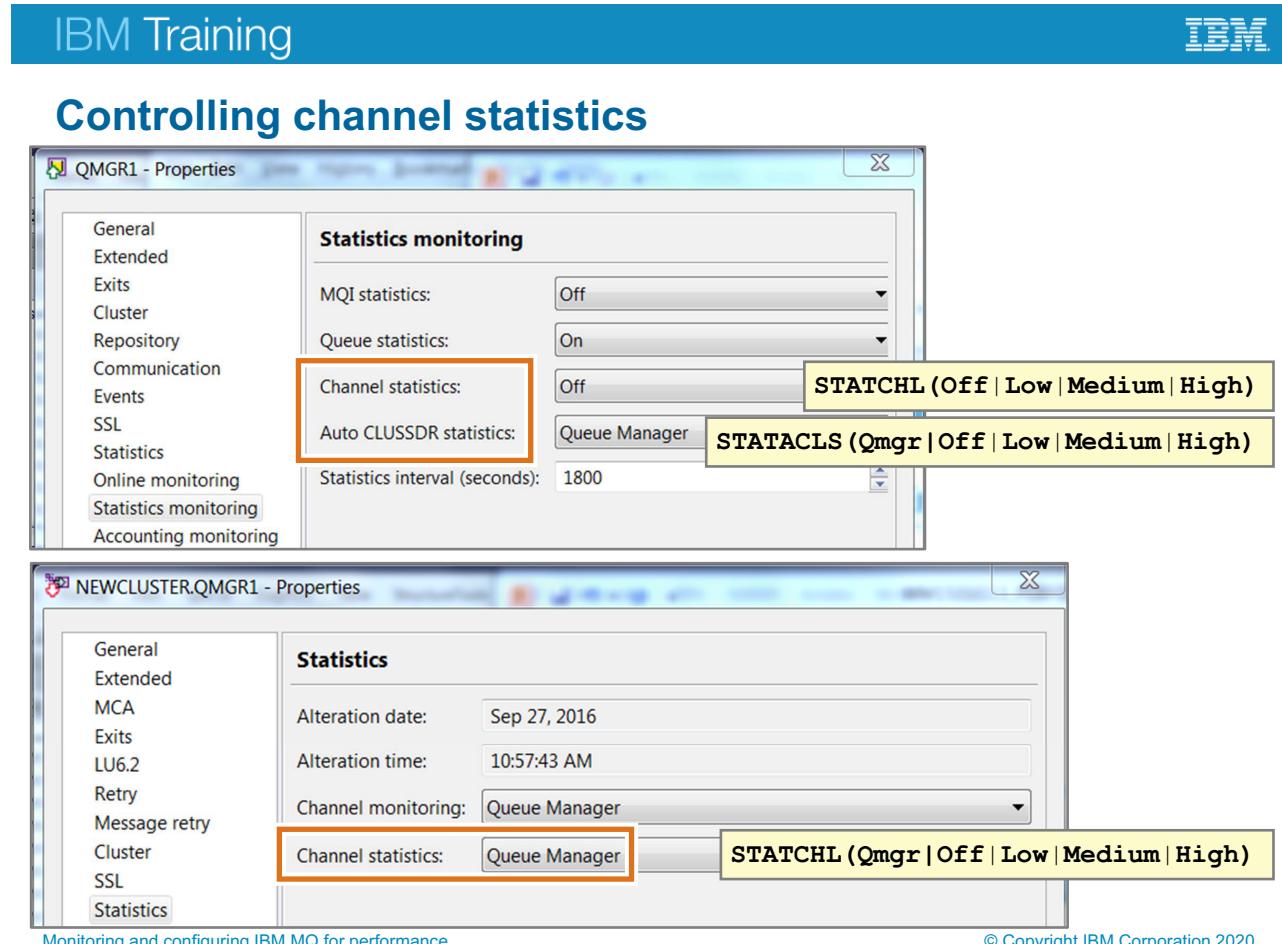


Figure 15-16. Controlling channel statistics

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.

## 15.2. IBM MQ accounting data

## IBM MQ accounting data

Monitoring and configuring IBM MQ for performance

© Copyright IBM Corporation 2020

*Figure 15-17. IBM MQ accounting data*

## IBM MQ accounting data collection

- Can collect connection-level and queue information for each connection
  - MQI accounting data controlled by a queue manager
  - Queue accounting controlled by a 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

*Figure 15-18. IBM MQ accounting data collection*

Accounting messages record information about the MQI operations that IBM 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 IBM 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

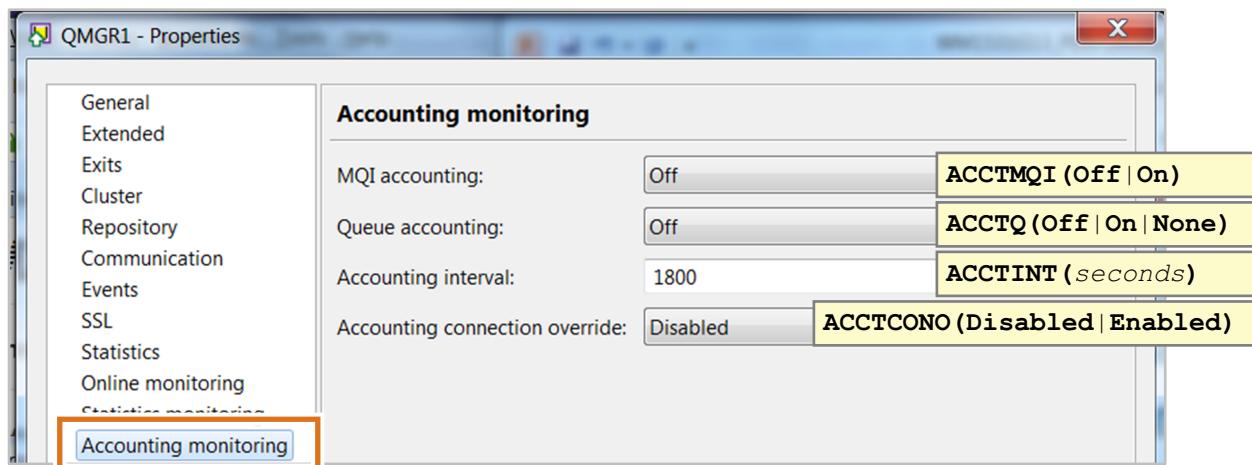
- Confirming that your queue manager network is running correctly

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.

# IBM Training

## Controlling queue manager accounting



- Can set properties for both local and remote queue managers

*Figure 15-19. 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

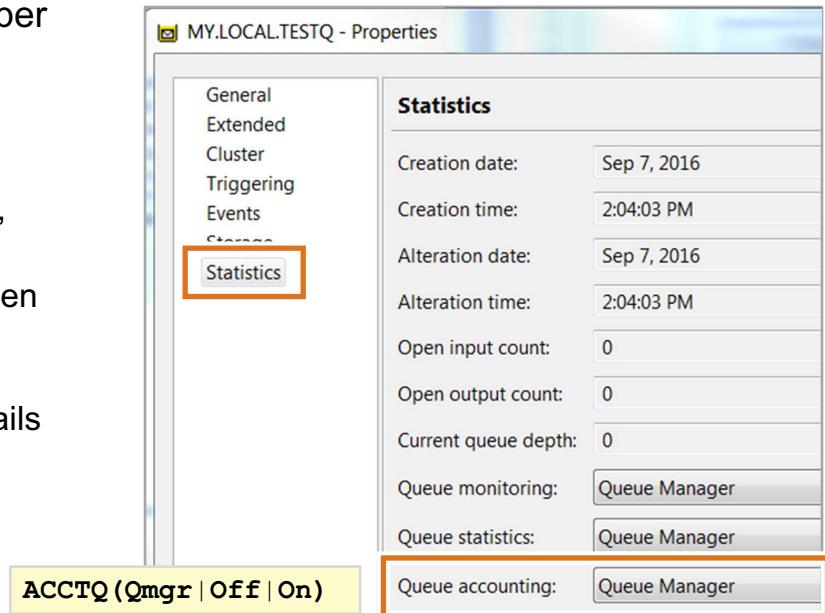


Figure 15-20. Controlling queue accounting

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.

## 15.3. Displaying statistics and accounting data

## Displaying statistics and accounting data

Monitoring and configuring IBM MQ for performance

© Copyright IBM Corporation 2020

*Figure 15-21. Displaying statistics and accounting data*

## 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.ACOUNTING.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
```

Figure 15-22. 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 suitable format.

The IBM 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.ACOUNTING.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 output

```
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 15-23. 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]
.
.
```

Figure 15-24. Example of statistics data

## 15.4. Real-time monitoring

## Real-time monitoring

Monitoring and configuring IBM MQ for performance

© Copyright IBM Corporation 2020

*Figure 15-25. Real-time monitoring*

## 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**

Figure 15-26. Real-time monitoring

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 IBM 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

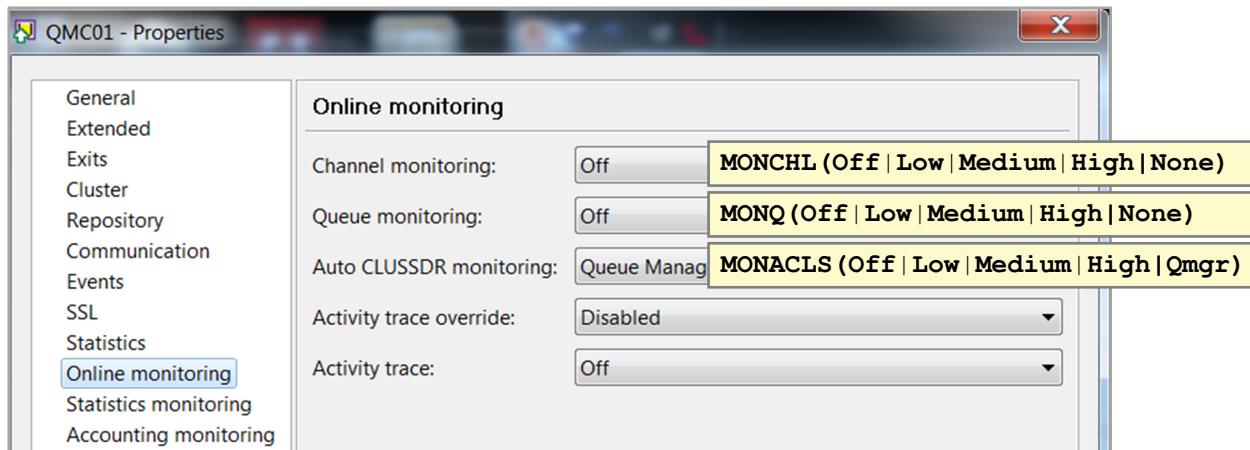


Figure 15-27. Controlling queue manager real-time monitoring

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**.

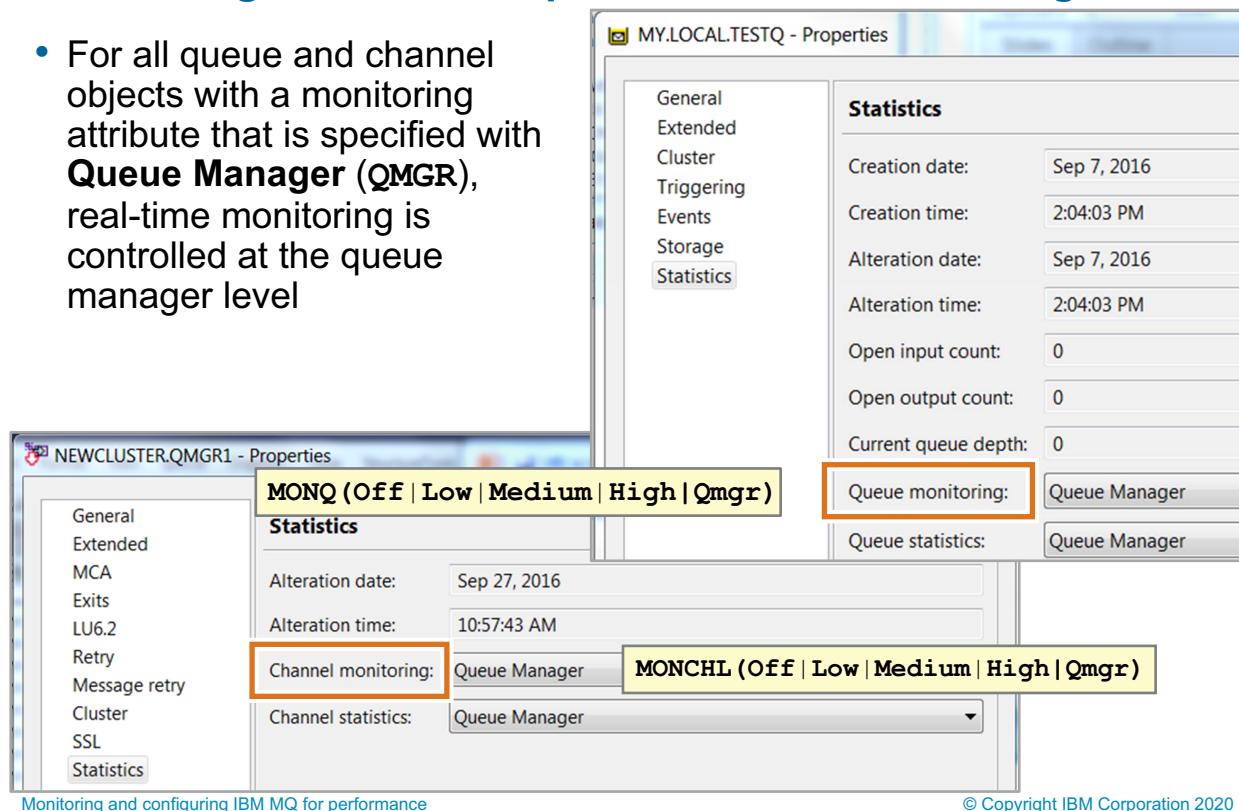
The **Queue monitoring** property specifies whether to collect online monitoring data about the current performance of queues that the queue manager hosts.

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 2020

Figure 15-28. 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 15-29. 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.

## 15.5. Configuring and tuning IBM MQ for performance

## Configuring and tuning IBM MQ for performance

Monitoring and configuring IBM MQ for performance

© Copyright IBM Corporation 2020

*Figure 15-30. Configuring and tuning IBM MQ for performance*

## 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 2020

Figure 15-31. Configuring and tuning IBM MQ for performance

You can make configuration changes to improve the performance of message processing with IBM MQ.



### Attention

It might not be necessary to implement the tuning guidelines described in this unit, especially if the message throughput and response time of the queue manager are acceptable. 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

Monitoring and configuring IBM MQ for performance

© Copyright IBM Corporation 2020

Figure 15-32. Queue manager logs

The location, size, and number of queue manager logs can affect performance.

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

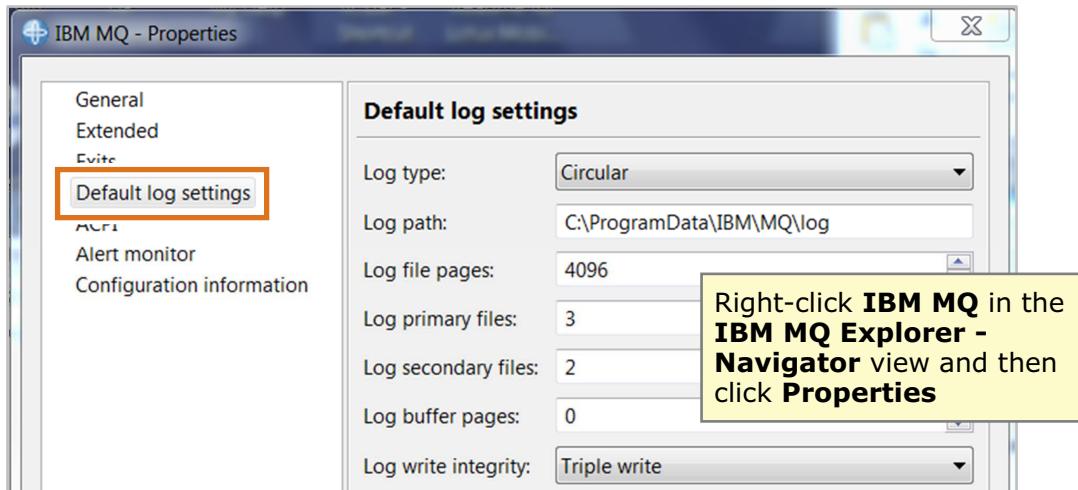


Figure 15-33. Default log settings in IBM MQ Explorer

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:

1. Right-click the queue manager in the **MQ Explorer - Navigator** view and then click **Properties**.
2. 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

Figure 15-34. Log type

The type of logging can affect performance.

Circular logging saves space because log files are reused when they no longer contain active log data.

## 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 15-35. 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

- 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

Figure 15-36. 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 2020

Figure 15-37. 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.

[https://www.ibm.com/support/knowledgecenter/en/SSFKSJ\\_9.1.0/com.ibm.mq.con.doc/q018860\\_.htm](https://www.ibm.com/support/knowledgecenter/en/SSFKSJ_9.1.0/com.ibm.mq.con.doc/q018860_.htm)

## 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

Monitoring and configuring IBM MQ for performance

© Copyright IBM Corporation 2020

Figure 15-38. 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. 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 IBM\MQSeries\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

Figure 15-39. Log write integrity level

IBM MQ provides full write integrity of logs where hardware that assures write integrity is not available.

The logger can be set to single write when 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 15-40. 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

Figure 15-41. Queue manager channels performance tuning

If your IBM 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 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

*Figure 15-42. 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.

## 15.6. Monitoring with IBM MQ Console



Monitoring and configuring IBM MQ for performance

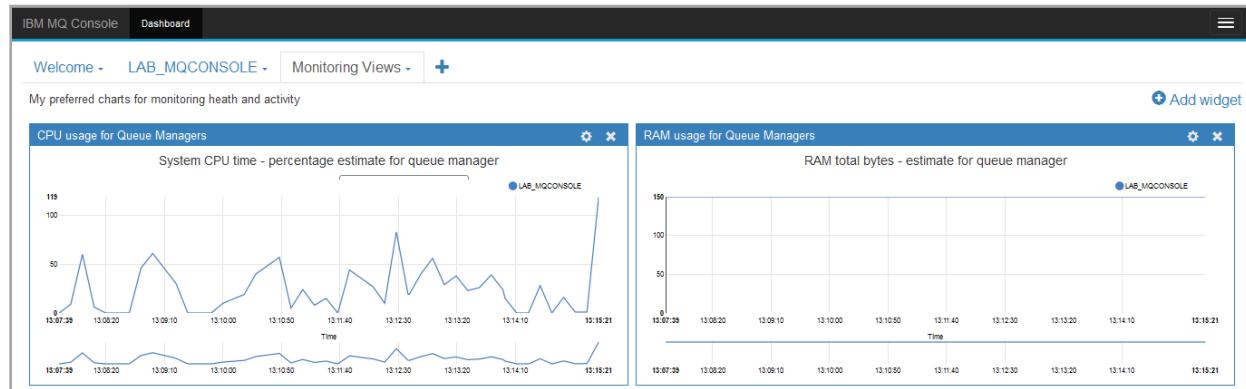
© Copyright IBM Corporation 2020

*Figure 15-43. Monitoring with IBM MQ Console*



## Configuring system resource monitoring

- Configure Charts widget to monitor a particular aspect of resource usage
  - Can create many instances of the Charts widget to display different data
  - Chart X-axis displays the timeline
  - Chart Y-axis displays units appropriate to the resource and is dynamically resized to accommodate the data that is returned



Monitoring and configuring IBM MQ for performance

© Copyright IBM Corporation 2020

Figure 15-44. Configuring system resource monitoring

You use the Charts widget in the IBM MQ Console to view monitoring data for queue managers.

You add a Charts widget to your dashboard and then configure it to monitor a particular aspect of resource usage. You can create many instances of the Charts widget to display different data.

The data is displayed in a chart format.

- The X-axis of the chart displays a timeline.
- The Y-axis displays units appropriate to the resource that you are viewing. The Y-axis is dynamically resized to accommodate the data that is returned.

Data is collected at 10-second intervals.

At least one queue manager must be running before you can configure a chart widget.

For complete list of resource monitoring classes and resource types, see "Monitoring system resource usage" in IBM Knowledge Center:

[www.ibm.com/support/knowledgecenter/SSFKSJ\\_9.1.0/com.ibm.mq.mqc.doc/q127750\\_.htm](http://www.ibm.com/support/knowledgecenter/SSFKSJ_9.1.0/com.ibm.mq.mqc.doc/q127750_.htm)

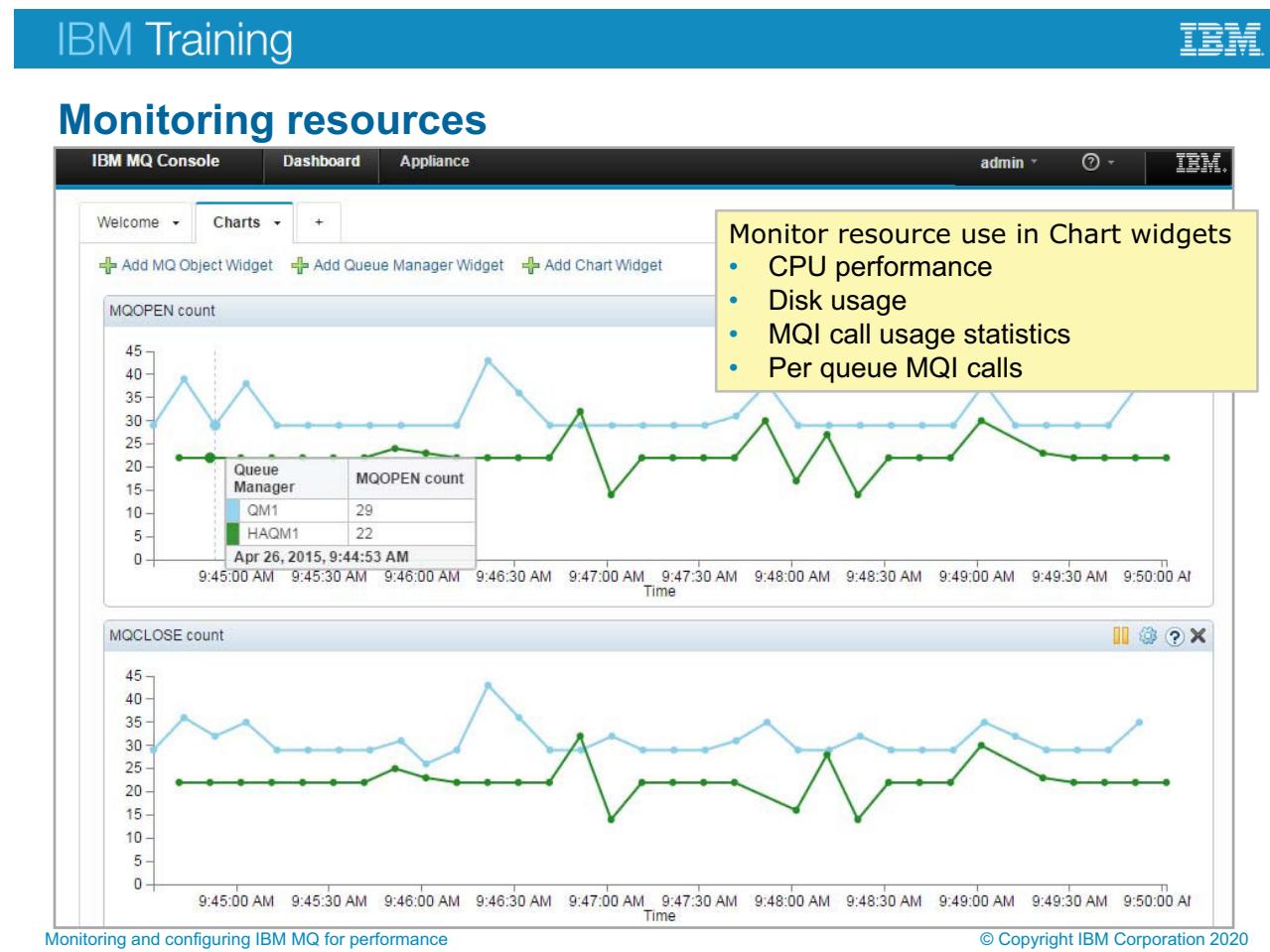


Figure 15-45. Monitoring resources

With the chart widget, you can monitor these resources:

- **Platform central processing:** Monitors CPU usage.
- **Platform persistent data stores:** Shows the use of disk resources.
- **API usage statistics:** Monitors MQI calls. It shows the same type of monitoring statistics as available with IBM MQ Explorer or by using MQSC options.

You can add chart widgets that can show multiple queue managers.

As shown in the example, you can hover the cursor over a collection point on the graph to display the values at that collection point.

## IBM MQ Console roles

- **MQWebAdmin**
  - Can do all operations
  - Operates under the security context of the operating system user ID that started the mqweb server
- **MQWebAdminRO**
  - Read-only access to display and inquire operations on IBM MQ objects, such as queues and channels, and browse messages on queues
- **MQWebUser**
  - Can start, stop, define, set, display, and inquire IBM MQ objects

Figure 15-46. IBM MQ Console roles

To use the IBM MQ Console, users need to authenticate against the mqweb server user registry.

To start monitoring system resources, the user or group must be part of the MQWebAdmin or MQWebAdminRO role.

Authenticated users must be a member of one of the groups that authorizes access to the capabilities of the IBM MQ Console.

Users with read-only access can access the IBM MQ Console to view IBM MQ objects and charts, but cannot change the configuration of IBM MQ objects.

During the exercise, you work with the MQWebAdmin or MQWebAdminRO roles.

## 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
- Monitor resources in the IBM MQ Console

## 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
  
3. True or False: Members of the MQWebAdminRO role can view performance chart data in IBM MQ Console.

Monitoring and configuring IBM MQ for performance

© Copyright IBM Corporation 2020

Figure 15-48. Review questions

Write your answers here:

- 1.
  
- 2.
  
- 3.

## Checkpoint 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](#).
3. True or False: Members of the MQWebAdminRO role can view performance chart data in IBM MQ Console.  
The answer is [True](#).

Monitoring and configuring IBM MQ for performance

© Copyright IBM Corporation 2020

Figure 15-49. Checkpoint answers

## Exercise: Monitoring IBM MQ for performance

Monitoring and configuring IBM MQ for performance

© Copyright IBM Corporation 2020

*Figure 15-50. Exercise: Monitoring IBM MQ for performance*

## Exercise introduction

- Enable accounting and statistics collection in IBM MQ
- View accounting and statistics data
- Configure a queue manager for online monitoring
- Monitor system resource usage



## Exercise: Monitoring resources with the IBM MQ Console

Monitoring and configuring IBM MQ for performance

© Copyright IBM Corporation 2020

*Figure 15-52. Exercise: Monitoring resources with the IBM MQ Console*

## Exercise introduction

- Monitor system resources
- Configure dashboard layouts
- Share an IBM MQ Console dashboard between user roles



---

# Unit 16. 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 2020

*Figure 16-1. Unit objectives*

## Course objectives

- Describe the IBM MQ deployment options
- Create and manage queue managers, queues, and channels
- Use the IBM MQ sample programs and utilities to test the IBM MQ network
- Configure distributed queuing
- Configure MQ 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

[Course summary](#)

© Copyright IBM Corporation 2020

*Figure 16-2. Course objectives*

## Course objectives

- Use IBM MQ troubleshooting tools to identify the cause of a problem in the IBM MQ network
- Plan for and implement basic 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 2020

*Figure 16-3. Course objectives*



## IBM badge

- Earn a Skills badge for this course by passing a quiz
- To earn the badge for this course:
  - [https://ibm-learning-skills-dev.github.io/ibm-learning-skills-dev.github.io/badges/IBM\\_MQ.html](https://ibm-learning-skills-dev.github.io/ibm-learning-skills-dev.github.io/badges/IBM_MQ.html)
- Other IBM Cloud badges:
  - <https://ibm-learning-skills-dev.github.io/ibm-learning-skills-dev.github.io/badges/badgemain.html>

Course summary

© Copyright IBM Corporation 2020

Figure 16-4. IBM badge

## IBM Professional Certifications

- By achieving an IBM Professional Certification, you can demonstrate your IBM Cloud product mastery to your employer or clients
- Certifications are a higher level of credential than a Skills badge for a single education course
- Product certifications demonstrate a strong knowledge of the product and typically require several months of work with the product
- IBM Cloud certifications are available for several roles, including developers, administrators, and business analysts
- For information on specific certifications and their requirements, see <http://www.ibm.com/certify>

Course summary

© Copyright IBM Corporation 2020

Figure 16-5. IBM Professional Certifications

## Other learning resources (1 of 4)

- **IBM Skills Gateway**

- Search the new IBM Training and Skills website (formerly IBM Authorized Training website) to find and access the content you want.
  - <https://www.ibm.com/training>

- **IBM Cloud Education Course Information Home**

- Go to the wiki to find course abstracts, course correction documents, and curriculum development plans for IBM Cloud offerings.
  - <https://ibm-learning-skills-dev.github.io/ibm-learning-skills-dev.github.io/education/courseinfo.html>

- **Role-based Learning Journeys**

- Learning Journeys describe the appropriate courses, in the recommended order, for specific products and roles.
  - [https://www.ibm.com/services/learning/journey\\_category?categoryId=o-itns-01-02](https://www.ibm.com/services/learning/journey_category?categoryId=o-itns-01-02)

Figure 16-6. Other learning resources (1 of 4)

## Other learning resources (2 of 4)

- **IBM Professional Certification Program**

- IBM Professional Certification enables skilled IT professionals to demonstrate their expertise to the world. It validates skills and proficiency in the latest IBM technology and solutions.
- <https://www.ibm.com/certify>

- **IBM Training blog, Twitter, and Facebook**

- These official IBM Training and Skills accounts provide information about IBM course offerings, industry information, conference events, and other education-related topics.
- <https://www.ibm.com/blogs/ibm-training>
- <https://twitter.com/IBMTTraining>
- <https://www.facebook.com/ibmtraining>

Course summary

© Copyright IBM Corporation 2020

Figure 16-7. Other learning resources (2 of 4)

## Other learning resources (3 of 4)

- **Business Partner Technical Enablement Portal**
  - <https://ibm.box.com/s/695khv9nyzekaorykqmsjrematz3v9xh>
  - This program provides technical training content modules to IBM software partners (via PartnerWorld) and IBM Business Partners.
- **IBM Developer**
  - IBM's official developer program offers access to software trials and downloads, how-to information, and expert practitioners.
  - <https://developer.ibm.com>
- **IBM Education Assistant**
  - These multimedia educational modules help users gain a better understanding of IBM Software products and use them more effectively to meet business requirements.
  - <https://www.ibm.com/products/software>

Course summary

© Copyright IBM Corporation 2020

Figure 16-8. Other learning resources (3 of 4)

## Other learning resources (4 of 4)

- **IBM Knowledge Center**

- The IBM Knowledge Center is the primary home for IBM product documentation.
- <https://www.ibm.com/support/knowledgecenter>

- **IBM Marketplace**

- IBM Marketplace is the landing page for all IBM Cloud products. Go to the Marketplace to learn about IBM offerings for Cloud, Cognitive, Data and Analytics, Mobile, Security, IT Infrastructure, and Enterprise and Business Solutions.
- <https://www.ibm.com/products>

- **IBM Redbooks**

- IBM Redbooks are developed and published by the IBM International Technical Support Organization (ITSO). Redbooks typically provide positioning and value guidance, installation and implementation experiences, typical solution scenarios, and step-by-step "how-to" guidelines.
- <http://www.redbooks.ibm.com>

Course summary

© Copyright IBM Corporation 2020

Figure 16-9. Other learning resources (4 of 4)

## 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 2020

*Figure 16-10. Unit summary*



## Course completion

**You have completed this course:**

*IBM MQ V9.1 System Administration*



**Do you have any questions?**

[Course summary](#)

© Copyright IBM Corporation 2020

*Figure 16-11. Course completion*

# Appendix 17.List of abbreviations

|              |                                             |
|--------------|---------------------------------------------|
| <b>ACL</b>   | access control list                         |
| <b>AIX</b>   | Advanced IBM UNIX                           |
| <b>API</b>   | application programming interface           |
| <b>CCDT</b>  | client channel definition table             |
| <b>CCSID</b> | coded character set identifier              |
| <b>CDR</b>   | Continuous Delivery Release                 |
| <b>COBOL</b> | Common Business Oriented Language           |
| <b>CPU</b>   | central processing unit                     |
| <b>DLH</b>   | dead-letter header                          |
| <b>DNS</b>   | Domain Name System                          |
| <b>EOF</b>   | end of file                                 |
| <b>FFDC</b>  | first failure data capture                  |
| <b>FFST</b>  | First Failure Support Technology            |
| <b>FIPS</b>  | Federal Information Processing Standard     |
| <b>FTP</b>   | File Transfer Protocol                      |
| <b>GPFS</b>  | General Parallel File System                |
| <b>HP-UX</b> | Hewlett-Packard UNIX                        |
| <b>HTTP</b>  | Hypertext Transfer Protocol                 |
| <b>IaaS</b>  | infrastructure as a service                 |
| <b>IBM</b>   | International Business Machines Corporation |
| <b>IP</b>    | Internet Protocol                           |
| <b>IPC</b>   | interprocess communication                  |
| <b>IPLA</b>  | International Program License Agreement     |
| <b>JMS</b>   | Java Message Service                        |
| <b>JNDI</b>  | Java Naming and Directory Interface         |
| <b>LSN</b>   | log sequence number                         |
| <b>LTSR</b>  | Long Term Support Release                   |
| <b>MCA</b>   | message channel agent                       |
| <b>MQ</b>    | message queue                               |
| <b>MQDLH</b> | MQ dead-letter queue header                 |

|                |                                                 |
|----------------|-------------------------------------------------|
| <b>MQFB</b>    | MQ feedback code                                |
| <b>MQI</b>     | Message Queue Interface                         |
| <b>MQMD</b>    | MQ message descriptor                           |
| <b>MQRC</b>    | MQ reason code                                  |
| <b>MQSC</b>    | MQ script commands                              |
| <b>MQTMC2</b>  | MQ trigger message 2 character format           |
| <b>MQTT</b>    | MQ Telemetry Transport                          |
| <b>MQXQH</b>   | MQ transmission queue header                    |
| <b>NetBIOS</b> | Network Basic Input/Output System               |
| <b>OAM</b>     | object authority manager                        |
| <b>QoS</b>     | quality of service                              |
| <b>PaaS</b>    | platform as a service                           |
| <b>PCF</b>     | programmable command format                     |
| <b>PI</b>      | program isolation                               |
| <b>PID</b>     | process ID                                      |
| <b>RAM</b>     | random access memory                            |
| <b>RCP</b>     | rich client platform                            |
| <b>SMIT</b>    | System Management Interface Tool                |
| <b>SNA</b>     | Systems Network Architecture                    |
| <b>SPX</b>     | Sequenced Packet Exchange                       |
| <b>SSL</b>     | Secure Sockets Layer                            |
| <b>TCP</b>     | Transmission Control Protocol                   |
| <b>TCP/IP</b>  | Transmission Control Protocol/Internet Protocol |
| <b>TLS</b>     | Transport Layer Security                        |
| <b>UNIX</b>    | Uniplexed Information and Computing System      |
| <b>URL</b>     | Uniform Resource Locator                        |
| <b>z/OS</b>    | zSeries operating system                        |



IBM Training



© Copyright International Business Machines Corporation 2020.