

IBM MQ



Product Overview

Version 9 Release 0

Note

Before using this information and the product it supports, read the information in "Notices" on page 459.

This edition applies to version 9 release 0 of IBM MQ and to all subsequent releases and modifications until otherwise indicated in new editions.

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

© **Copyright IBM Corporation 2007, 2018.**

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

Contents

Figures v

Tables vii

About IBM MQ 1

Introduction to IBM MQ	1
IBM MQ license information	4
IBM MQ for z/OS product identifiers	8
Pricing metric for Virtual Processor Cores	9
What's new and changed in IBM MQ Version 9.0.0	10
What's new in Version 9.0.0	10
What's changed in Version 9.0.0	15
New, changed and removed messages in Version 9.0.0	20
What's new and changed in Version 9.0.x	
Continuous Delivery	25
What's new and changed in IBM MQ Version 9.0.5	25
What's new and changed in IBM MQ Version 9.0.4	39
What's new and changed in IBM MQ Version 9.0.3	55
What's new and changed in IBM MQ Version 9.0.2	63
What's new and changed in IBM MQ Version 9.0.1	72
What's changed in Version 9.0.0.x Long Term Support	80
What's changed in Version 9.0.0, Fix Pack 3.	81
What's changed in Version 9.0.0, Fix Pack 2.	83
What's changed in Version 9.0.0, Fix Pack 1.	85
What was new and changed in earlier versions	89
Accessibility features for IBM MQ	93
Release and platform icons in the product documentation	94
IBM MQ information roadmap	96
IBM MQ customer-installable Knowledge Center	100
IBM MQ Version 9.0.x PDF documentation	101
Glossary	103
A	103
B	106
C	107
D	112
E	114
F	115
G	117
H	117
I	118
J	121
K	121
L	122
M	123
N	127
O	128
P	130

Q	133
R	134
S	137
T	142
U	144
V	145
W	145
X	145

IBM MQ Technical overview 147

Introduction to message queuing	147
Main features and benefits of message queuing	149
Message queuing terminology	151
Messages and queues	154
IBM MQ objects	156
Object types	157
Naming IBM MQ objects	175
Object attributes	181
Queue-sharing groups	181
System default objects	182
Distributed queuing and clusters	182
Distributed queuing components	187
Cluster components	197
Publish/subscribe messaging	203
Publish/subscribe components	204
Example of a single queue manager publish/subscribe configuration	231
Distributed publish/subscribe networks	232
IBM MQ Multicast	251
Initial multicast concepts	251
MQ Telemetry	252
Introduction to MQ Telemetry	254
Telemetry use cases	255
Connecting telemetry devices to a queue manager	262
Telemetry connection protocols	263
Telemetry (MQXR) service	263
Telemetry channels	264
MQTT protocol	264
MQTT clients	265
Sending a message to an MQTT client	265
Send a message to an IBM MQ application from an MQTT client	274
MQTT publish/subscribe applications	276
Telemetry applications	276
Integration of MQ Telemetry with queue managers	277
MQTT stateless and stateful sessions	279
When an MQTT client is not connected.	279
Loose coupling between MQTT clients and IBM MQ applications	280
MQ Telemetry security	281
MQ Telemetry globalization	281
Performance and scalability of MQ Telemetry	282
Devices supported by MQ Telemetry	284

Security in IBM MQ	285
IBM MQ.NET managed client TLS support	286
Clients and servers	287
Overview of IBM MQ MQI clients	287
Transaction management and support	295
Extending queue manager facilities	297
IBM MQ Java language interfaces	298
IBM MQ classes for JMS.	298
Architecture and overview of features	300
IBM MQ client for HP Integrity NonStop Server	308
IBM MQ client for HP Integrity NonStop Server technical overview	313
IBM MQ client for HP Integrity NonStop Server supported environments and features	313
IBM MQ for z/OS concepts	314
The queue manager on z/OS	315
The channel initiator on z/OS.	317
Terms and tasks for managing IBM MQ for z/OS	320
Shared queues and queue-sharing groups	322
Intra-group queuing	369
Storage management on z/OS.	384
Logging in IBM MQ for z/OS.	390

Defining your system on z/OS	401
Recovery and restart on z/OS.	412
Security concepts in IBM MQ for z/OS.	429
Availability on z/OS	436
Monitoring and statistics on IBM MQ for z/OS	439
Unit of recovery disposition on z/OS	440
IBM MQ and other z/OS products	443
IBM MQ and CICS	443
IBM MQ and IMS	445
IBM MQ and the z/OS Batch, TSO, and RRS adapters	449
IBM MQ for z/OS and WebSphere Application Server	450

Index 453

Notices 459

Programming interface information	460
Trademarks	461

Sending your comments to IBM . . . 463

Figures

1. Message queuing compared with traditional communication	150	40. Create wildcard subscriptions: wildsubs.tst	245
2. Message channels between two queue managers	170	41. Delete and create subscriptions: fullsubs.tst	245
3. Client-connection and server-connection on an MQI channel.	171	42. Create queue managers: qmgrs.bat	246
4. Overview of the components of distributed queuing	183	43. Update subscriptions: upsubs.bat	246
5. Sending messages	184	44. Publish: pub.bat	246
6. Sending messages in both directions	184	45. MQTT v3 client subscriber	266
7. A cluster of queue managers	186	46. IBM MQ publisher	267
8. A sender-receiver channel	188	47. messageArrived method	267
9. A requester-server channel	188	48. MQTT v3 client publisher	267
10. A requester-sender channel	189	49. Alter maximum number of handles on Windows	284
11. A cluster-sender channel	189	50. Alter maximum number of handles on Linux	284
12. Passing through intermediate queue managers	191	51. IBM MQ.NET managed and unmanaged TLS comparison	286
13. Sharing a transmission queue	191	52. Link between a client and server	288
14. Using multiple channels	192	53. IBM MQ server connected to clients on different platforms	294
15. Queue manager alias	194	54. The layered architecture for IBM JMS providers	301
16. Reply-to queue alias used for changing reply location	196	55. The relationship between IBM MQ classes for JMS and IBM MQ classes for Java.	303
17. Simple publish/subscribe configuration	203	56. Overview of IBM MQ for z/OS	316
18. Example of a topic tree	218	57. Communication between queue managers	318
19. Visualization of a topic tree	219	58. A queue-sharing group	324
20. Extended topic tree	219	59. The components of queue managers in a queue-sharing group	325
21. Visualization of an administrative topic object associated with the Sport/Soccer topic	220	60. Multiple instances of an application servicing a shared queue	344
22. Topic tree with several administrative topic objects	221	61. Basic configuration.	353
23. A topic tree that uses the WILDCARD property, BLOCK	222	62. Configuration adding SMDS and SCM for emergency storage	355
24. Publish to QMA	223	63. Basic configuration.	358
25. Delete and create topics: topics.tst.	226	64. Configuration adding SCM for improved performance	361
26. Create wildcard subscriptions: wildsubs.tst	226	65. Distributed queuing and queue-sharing groups	363
27. Delete and create subscriptions: fullsubs.tst	227	66. A queue-sharing group as part of a cluster	367
28. Create queue managers: qmgrs.bat	227	67. An example of intra-group queuing	371
29. Update subscriptions: upsubs.bat	228	68. An example of migration support.	373
30. Publish: pub.bat	228	69. An example configuration	376
31. Version 6 streams that coexist with version 7 topics	230	70. An example of clustering with intra-group queuing	378
32. Single queue manager publish/subscribe example	232	71. Mapping queues to page sets through storage classes	387
33. Publish/subscribe example with two queue managers	233	72. Buffers, buffer pools, and page sets	389
34. Publish/subscribe cluster	233	73. The logging process	396
35. Publish/subscribe hierarchy	234	74. The offloading process	397
36. Direct routing and topic host routing	236	75. A unit of recovery within an application program	413
37. A topic tree that uses the WILDCARD property, BLOCK	240	76. A unit of recovery showing back out	414
38. Publish to QMA	241	77. The two-phase commit process.	415
39. Delete and create topics: topics.tst.	244	78. The IBM MQ - IMS bridge	447

Tables

1. Table of products and their associated PID values.	9	19. Differences between managed and unmanaged implementations	286
2. Platform icons	95	20. MQGET options permitted when read ahead is enabled.	290
3. IBM MQ information roadmap table	97	21.	313
4. MQ KC framework packages: Links, file names and file descriptions	101	22. Comparison of results for the emergency storage scenario.	358
5. Mapping of PDF files to product documentation sections	102	23. Where to find more information about shared queues and queue-sharing groups	369
6. Transmission protocols for MQI channels	172	24. Short and long retry counts and intervals values	383
7. Topic string concatenation examples	217	25. Where to find more information about storage management	390
8. Publications received on QMA	223	26. IBM MQ sample definitions for system objects	408
9. Publications received on QMA	224	27. Termination using QUIESCE, FORCE, and RESTART	417
10. Publications received on QMB	224	28. Where to find more information about availability	439
11. Publications received on QMA	241	29. Where to find more information about using z/OS Batch with IBM MQ	450
12. Publications received on QMA	242		
13. Publications received on QMB	243		
14. MQMD	277		
15. RFH2	278		
16. Message contents	278		
17. MQTT fixed header properties	278		
18. MQTT Variable header properties	279		

About IBM MQ

Introductory information to help you get started with IBM® MQ Version 9.0, including an introduction to the product and an overview of what is new and what is changed for this release.

Related information:

-  [WebSphere® Certification](#)
-  [Product support and downloads](#)
-  [System requirements](#)
-  [Support technotes](#)
-  [IBM Support Portal](#)
-  [IBM Redbooks® publications](#)
-  [developerWorks®](#)
-  [IBM Middleware User Community](#)
-  [MQSeries.net](#)

Introduction to IBM MQ

You can use IBM MQ to enable applications to communicate at different times and in many diverse computing environments.

What is IBM MQ?

IBM MQ can transport any type of data as messages, enabling businesses to build flexible, reusable architectures such as service-oriented architecture (SOA) environments and microservice architectures. It works with a broad range of computing platforms, application environments and APIs, and communications protocols for security-rich message delivery. IBM MQ provides a communications layer for visibility and control of the flow of messages and data inside and outside your organization.

IBM MQ provides:

- Versatile messaging integration from mainframe to mobile that provides a single, robust messaging backbone for dynamic heterogeneous environments.
- Message delivery with security-rich features that produce auditable results.
- Qualities of service that provide once and once only delivery of messages to ensure messages will withstand application and system outages
- High-performance message transport to deliver data with improved speed and reliability.
- Highly available and scalable architectures to support an application's needs.
- Administrative features that simplify messaging management and reduce time spent using complex tools.
- Open standards development tools that support extensibility and business growth.

An application has a choice of programming interfaces, and programming languages to connect to IBM MQ.

IBM MQ provides these *messaging* and *queuing* capabilities across multiple several modes of operation: *point-to-point* ; *publish/subscribe* ; *file transfer* ; *telemetry*.

Messaging

Programs communicate by sending each other data in messages rather than by calling each other directly.

Queuing

Messages are placed on queues, so that programs can run independently of each other, at different speeds and times, in different locations, and without having a direct connection between them.

Point-to-point

Applications send messages to a queue and receive messages from a queue. Each message is consumed by a single instance of an application. The sender must know the name of the destination, but not where it is.

Publish/subscribe

Applications subscribe to topics. When an application publishes a message on a topic, IBM MQ sends copies of the message to those subscribing applications. The publisher does not know the names of subscribers, or where they are.

File transfer

Files are transferred in messages. Managed File Transfer manages the transfer of files and the administration to set up automated transfers and log the results.

Telemetry

MQ Telemetry provides an IBM MQ Telemetry Transport (MQTT) server that extends your IBM MQ environment to the “Internet of Things” - that is, the numerous mobile, web and M2M devices and applications that sit at the edge of the network.

What can it do for me?

IBM MQ provides a universal messaging backbone with robust connectivity for flexible and reliable messaging for applications and the integration of existing IT assets using a service-oriented architecture (SOA).

- IBM MQ sends and receives data between your applications, and over networks.
- Message delivery is *assured* and *decoupled* from the application. Assured, because IBM MQ exchanges messages transactionally, and decoupled, because applications do not have to check that messages they sent are delivered safely.
- You can secure message delivery between queue managers with TLS.
- With Advanced Message Security (AMS), you can encrypt and sign messages between being put by one application and retrieved by another.
- Application programmers do not need to have communications programming knowledge.

How do I use it?

An IBM MQ messaging system is made up of one or more queue managers. Queue managers are where messaging resources, such as queues, are configured and what applications connect to, either running on the same system as the queue manager or over the network.


A network of connected queue managers supports asynchronous routing of messages between systems, where producing and consuming applications are connected to different queue managers.


IBM MQ can be managed using a variety of tools, from the IBM MQ Explorer GUI, through scripted or interactive command line tools or programmatically.

The applications connecting to IBM MQ can be written in any one of many different programming languages and to many different APIs. From C and Cobol, to Java and .Net to NodeJS and Ruby.

How does it work?

Here is a brief overview of how IBM MQ works.

- First a messaging application must connect to a queue manager. This may require a channel to be created on the queue manager to accept application connections.
- When your application wants to transfer data to another application, it creates a message and puts the data into it. It puts the message onto a queue, or publishes the message to a topic to be delivered to subscribers to the topic.
- The queue or subscriptions may be on the same queue manager or on other, connected, queue managers. If the latter the connected queue managers work together to reliably transfer the message from the producer's queue manager to the target queue managers. The applications do not communicate with each other, the queue managers do.
- Messages can be handled based on a number of different qualities of service, where reliability and persistence is traded for speed. The highest quality of service given to messages are persistent messages that are sent and received under the control of transactions. This will ensure that messages are delivered once and once only even in the event of system, network or application failures.
- IBM MQ channels are used to connect one queue manager to another over a network. You can create MQ channels yourself, or a queue manager can join a cluster of queue managers where MQ channels are automatically created when they are needed.
-  On z/OS®, you can configure multiple queue managers to share queues on the coupling facility. Applications connected to different queue managers can get and put messages to and from the same queues.
- You can have many queues and topics on one queue manager.
- You can have more than one queue manager on one computer.
- An application can run on the same computer as the queue manager, or on a different one. If it runs on the same computer, it is an IBM MQ server application. If it runs on a different computer, it is an IBM MQ client application. Whether it is IBM MQ client or server makes almost no difference to the application. You can build a client/server application with IBM MQ clients or servers.

 If you are new to using IBM MQ, see Getting started with IBM MQ in the *Scenarios* section of this documentation.

What tools and resources come with IBM MQ ?

IBM MQ provides the following tools and resources:

- Control commands, which are run from the command line. You create, start, and stop queue managers with the control commands. You also run IBM MQ administrative and problem determination programs with the control commands.
- IBM MQ script commands (MQSC), which are run by an interpreter. Create queues and topics, configure, and administer IBM MQ with the commands. Edit the commands in a file, and pass the file to the `runmqsc` program to interpret them. You can also run the interpreter on one queue manager, which sends the commands to a different computer to administer a different queue manager.
- The Programmable Command Format (PCF) commands, which you call in your own applications to administer IBM MQ. The PCF commands have the same capability as the script commands, but they are easier to program.
- Sample programs.
- On Windows and Linux x86 and x86-64 platforms, you can use the following utilities:

- The IBM MQ Explorer. You can use the IBM MQ Explorer to administer and configure the entire network of queue managers, irrespective of the platform that they are running on. IBM MQ Explorer does the same administrative tasks as the script commands, but is much easier to use interactively.
- The *Postcard* application to demonstrate messaging and verify your installation.
- Tutorials.

Related concepts:

“What’s new and changed in IBM MQ Version 9.0.0” on page 10

IBM MQ Version 9.0 introduces a new continuous delivery and support model. The initial release under this new model delivers a number of new and enhanced capabilities.

“What’s changed in Version 9.0.0” on page 15

Changes to functions and resources in IBM MQ Version 9.0.0 are described in this section. Review these changes before upgrading queue managers to the latest product version and decide whether you must plan to make changes to existing applications, scripts, and procedures before starting to migrate your systems.

“IBM MQ Technical overview” on page 147

Use IBM MQ to connect your applications and manage the distribution of information across your organization.

Related information:

Scenarios

IBM MQ license information

What you can purchase with IBM MQ and what each purchase entitles you to install.

What you can purchase with IBM MQ

Multiplatforms


For IBM MQ for Multiplatforms, the product offering contains the following chargeable components that can be independently purchased:

5724-H72 IBM MQ

IBM MQ (Server)

 IBM MQ High Availability Replica (formerly IBM MQ Idle Standby)

IBM MQ Advanced


 IBM MQ Advanced High Availability Replica (formerly IBM MQ Advanced Idle Standby)

IBM MQ Advanced for Developers

For the following chargeable components, existing Subscription and Support can be renewed. No new entitlement can be purchased. See the Announcement Letter for more information.

IBM MQ Telemetry

Advanced Message Security

 Advanced Message Security High Availability Replica (formerly Advanced Message Security Idle Standby)

Managed File Transfer Service

 Managed File Transfer Service High Availability Replica (formerly Managed File Transfer Service Idle Standby)

Managed File Transfer Agent

z/OS

IBM MQ for z/OS contains five chargeable components that you can purchase independently:

5655-MQ9

IBM MQ for z/OS

5655-MF9

Managed File Transfer for z/OS

5655-AM9

Advanced Message Security for z/OS

5655-AV1

IBM MQ Advanced for z/OS, Value Unit Edition

5655-AV9

IBM MQ Advanced for z/OS

5655-VU9

IBM MQ for z/OS Value Unit Edition

Multi

What is my enterprise entitled to install? (Multiplatforms)

For IBM MQ for Multiplatforms the components below map directly to components that the IBM MQ installer can install, so for these the mapping between what you have purchased and what you can install is easy.

Important: The IBM MQ installation media contains all the components, but you should only install the subset that you have purchased entitlement for.

5724-H72 IBM MQ

IBM MQ (Server)

Includes:

All components that are not listed as part of the other chargeable components.

Unlimited installs of IBM MQ Client

IBM MQ Telemetry

Includes:

Telemetry Service


Advanced Message Security

Includes:

Advanced Message Security

Managed File Transfer Service

Includes:

 Managed File Transfer Logger DATABASE component

Managed File Transfer Service

Managed File Transfer Tools

Managed File Transfer Agent

Includes:

Managed File Transfer Agent

V 9.0.4 Managed File Transfer Logger DATABASE component
Managed File Transfer Tools

5724-H72 IBM MQ Advanced

See section “What is IBM MQ Advanced?” for more information.

Includes:

V 9.0.4 **Replicated Data Queue Manager feature**
Available on RedHat x86-64 only

V 9.0.4 **IBM MQ Bridge to blockchain**
Delivered on Linux x86-64 only

V 9.0.4 **Managed File Transfer Logger in Client mode**
For FILE mode on clients only.

Important: If your enterprise wants to use the Managed File Transfer Logger in DATABASE mode you still need to install the IBM MQ Server component.

z/OS

What is my enterprise entitled to install? (z/OS platform)

For products on z/OS the installation media contains only what you have purchased:

5655-MQ9

IBM MQ for z/OS

5655-MF9

Managed File Transfer for z/OS

5655-AM9

Advanced Message Security for z/OS

5655-VU9

IBM MQ for z/OS Value Unit Edition

5655-AV9

IBM MQ Advanced for z/OS

5655-AV1

IBM MQ Advanced for z/OS, Value Unit Edition

IBM MQ for z/OS Value Unit Edition (VUE) is restricted to *'Eligible workload'* in a zNALC partition, as defined in the product license.

What is IBM MQ Advanced?

IBM MQ Advanced has been introduced to simplify the process of purchasing entitlement. Your enterprise pays one price and obtains entitlement to multiple IBM MQ components.

The IBM MQ Advanced parts are:

5724-H72 IBM MQ

IBM MQ Advanced

IBM MQ Advanced for Developers

and



5655-AV9



IBM MQ Advanced for z/OS

5655-AV1

IBM MQ Advanced for z/OS, Value Unit Edition

IBM MQ Advanced on Multiplatforms gives your enterprise entitlement to install the following components up to your entitled capacity:

- IBM MQ (Server)
- Advanced Message Security
- Managed File Transfer Service
- Unlimited installs of MQ Telemetry
- Unlimited installs of IBM MQ Client
- Unlimited installs of Managed File Transfer Agent (including the Protocol Bridge Agent), providing the Queue Manager the Agent connects to is entitled using IBM MQ Advanced capacity
-  Replicated Data Queue Manager feature - available on RedHat x86-64 only
-  IBM MQ Bridge to blockchain - delivered on Linux x86-64 only

  You can also install IBM MQ into IBM Cloud Private.

Additionally, your enterprise can mix and match IBM MQ versions as required. Therefore, your 100 PVUs of IBM MQ (Server) entitlement could be split into 50 PVUs of the IBM WebSphere MQ Version 7.5 version and 50 PVUs of the IBM MQ Version 9.0 version of this component.

IBM MQ Advanced for Developers gives entitlement to the same set of components included with IBM MQ Advanced.

Attention: The IBM MQ license defines what is considered as development purposes.

IBM MQ Advanced for z/OS gives entitlement to install the following components up to your entitled capacity:

- Managed File Transfer for z/OS
- Advanced Message Security for z/OS
- Unlimited installs of IBM MQ Client on Multiplatforms
- Unlimited installs of Managed File Transfer Agent on Multiplatforms

Important: Your enterprise needs to purchase entitlement for the IBM MQ for z/OS base queue manager separately.

IBM MQ Advanced for z/OS, Value Unit Edition gives entitlement to install:

- Managed File Transfer for z/OS
- Advanced Message Security for z/OS
- IBM MQ for z/OS Value Unit Edition
- Unlimited installs of IBM MQ Client on Multiplatforms
- Unlimited installs of Managed File Transfer Agent on Multiplatforms, providing the Queue Manager the Agent connects to is entitled using IBM MQ Advanced for z/OS, Value Unit Edition capacity

Attention: This offering includes entitlement for the IBM MQ for z/OS base queue manager.



What are High Availability Replica parts?

High Availability Replica (formerly Idle Standby) parts have been introduced to cater for high availability environments, where the passive system has IBM MQ installed and available, but that system is not doing any IBM MQ processing work, or activity, other than staying up to date with the configuration and activity of the active queue manager. In this case a lower charge might be applicable.

Notes:

1. Use of the IBM MQ multi-instance queue manager feature also requires High Availability Replica entitlement.
2. There is no High Availability Replica part for the MQ Telemetry component. The same MQ Telemetry part needs to be purchased for the active and passive system, unless you have IBM MQ Advanced High Availability Replica, in which case it is included.
3. There is also no High Availability Replica part for Managed File Transfer Agent because the endpoint is not part of the server environment.

Related concepts:

V 9.0.0 “What's new and changed in IBM MQ Version 9.0.0” on page 10

IBM MQ Version 9.0 introduces a new continuous delivery and support model. The initial release under this new model delivers a number of new and enhanced capabilities.

CD “What's new and changed in Version 9.0.x Continuous Delivery” on page 25

Following the initial release of IBM MQ Version 9.0, new function and enhancements are made available by incremental updates within the same version and release as modification releases, for example Version 9.0.3.

LTS “What's changed in Version 9.0.0.x Long Term Support” on page 80

The Long Term Support (LTS) release is a recommended product level for which support, including defect and security updates, are provided over a specified period of time. This version is intended for systems which demand a long term deployment and maximum stability.

“IBM MQ Technical overview” on page 147

Use IBM MQ to connect your applications and manage the distribution of information across your organization.

Related information:

 Video: IBM MQ Advanced (YouTube)

IBM MQ for z/OS product identifiers

V 9.0.3

An overview of the products and their associated product identifiers (PIDs) available with IBM MQ for z/OS.

Table 1 on page 9 describes six different PIDs:

- A stand-alone PID for each of the 3 components (MQ9, AM9, MF9)
- A stand-alone VUE PID for the queue manager (VU9)
- An Advanced PID (AV9), and an Advanced VUE PID (AV1).

Table 1. Table of products and their associated PID values

Product	Queue manager PID	AMS PID	MFT PID
Standalone Note 1	5655-MQ9	5655-AM9	5655-MF9
Standalone VUE	5655-VU9	Note 2	Note 2
Advanced	Note 3	5655-AV9	5655-AV9
Advanced VUE	5655-AV1	5655-AV1	5655-AV1

Notes:

1. Standalone means one of :
 - IBM MQ for z/OS
 - IBM MQ for z/OS Advanced Message Security (AMS)
 - Managed File Transfer (MQMFT) for z/OS
2. Neither AMS nor MQMFT is available under a stand-alone VUE product.
3. The queue manager is not part of the IBM MQ Advanced product.
4. A component from one product can be used with a component from another product. For example, an AMS from IBM MQ Advanced can be used with a queue manager from a stand-alone IBM MQ product.
5. Advanced VUE is a zCAP (Collocated Application Pricing) Defining Program.

Related concepts:

“What’s new and changed in IBM MQ Version 9.0.3” on page 55

IBM MQ Version 9.0.3, which is a Continuous Delivery (CD) release, delivers a number of new and enhanced features on Linux, Windows, and z/OS.

“What’s changed in Version 9.0.3” on page 58

Changes to functions and resources in IBM MQ Version 9.0.3 are described in this section. Review these changes before upgrading queue managers to the latest product version and decide whether you must plan to make changes to existing applications, scripts, and procedures before starting to migrate your systems.

“IBM MQ Technical overview” on page 147

Use IBM MQ to connect your applications and manage the distribution of information across your organization.

Pricing metric for Virtual Processor Cores

A Virtual Processor Core (VPC) is a unit of measurement by which a program can be licensed. You must obtain entitlement for each VPC made available to the program.

The pricing of IBM MQ is based on the number of VPCs that your server uses. A VPC can be either a virtual core assigned to a virtual server, or a processor core in a non-partitioned physical server. You must license each virtual processor made available to the IBM MQ.

For more information about how to determine the usage of VPCs, see the developerWorks article [Determining the usage of Virtual Processor Cores \(VPC\)](#). This article explains how to use the IBM License Metric Tool to configure and create a report that you can use to determine the number of VPCs that you need to license. The article includes examples of how to calculate VPC consumption for virtual and physical server environments.

What's new and changed in IBM MQ Version 9.0.0

V 9.0.0

IBM MQ Version 9.0 introduces a new continuous delivery and support model. The initial release under this new model delivers a number of new and enhanced capabilities.

New delivery and support model

IBM MQ Version 9.0 introduces a new delivery and support model for IBM MQ. From Version 9.0, two release types will be made available; the Long Term Support (LTS) release and Continuous Delivery (CD) release.

LTS

The Long Term Support release is a recommended product level for which support, including defect and security updates, will be provided over a specified period of time. This version is intended for systems which demand a long term deployment and maximum stability.

CD

Continuous Delivery releases deliver new functional enhancements, in addition to fixes and security updates, on a much shorter cadence, so providing much more rapid access to those new functions. This version is intended for systems where applications want to exploit the very latest capabilities of IBM MQ.

For further information see IBM MQ release types and IBM MQ FAQ for Long Term Support and Continuous Delivery releases.

Related concepts:

CD

“What's new and changed in Version 9.0.x Continuous Delivery” on page 25
Following the initial release of IBM MQ Version 9.0, new function and enhancements are made available by incremental updates within the same version and release as modification releases, for example Version 9.0.3.

LTS

“What's changed in Version 9.0.0.x Long Term Support” on page 80
The Long Term Support (LTS) release is a recommended product level for which support, including defect and security updates, are provided over a specified period of time. This version is intended for systems which demand a long term deployment and maximum stability.

“What was new and changed in earlier versions” on page 89

Links to information about new features and changes to functions and resources, including stabilizations, deprecations and removals, that occurred in versions of the product before IBM MQ Version 9.0.

LTS

CD

“Release and platform icons in the product documentation” on page 94
In the IBM MQ Version 9.0 product documentation, icons are used to identify release at which a feature is made available. Icons are also used to identify information that applies to a specific platform or group of platforms only.

Related information:

[Video: IBM MQ Version 9.0 overview \(YouTube\)](#)

[What's new and what's changed in IBM MQ Explorer](#)

[System Requirements for IBM MQ](#)

[IBM MQ, WebSphere MQ, and MQSeries product readmes web page](#)

What's new in Version 9.0.0

V 9.0.0

The initial release of IBM MQ Version 9.0 under the new continuous delivery model, delivers a number of new and enhanced capabilities.

New family features

- “Additional quality of protection for AMS” on page 12
- “Web addressable access to the client channel definition table (CCDT)” on page 12
- “AMS supported in non-IBM JREs in Java clients” on page 12
- “Updated Resource Adapter for traditional WebSphere Application Server” on page 12
- “Enhanced Unicode data conversion support” on page 12
- Managed File Transfer Protocol Bridge Agent logging enhancements


New features for IBM MQ for Multiplatforms

- “Performance monitoring API and enhanced activity trace available through publish/subscribe” on page 13
- “LDAP authorization on Windows platforms” on page 14
- “runmqsc command line editor for UNIX” on page 14

New features for IBM MQ for z/OS

- “Data manager page set statistics” on page 14
- “JMS support within CICS OSGI Server” on page 14
- “JMS support within IMS Server” on page 14
- “Improved control of messages by using message expiry” on page 15
- “Increased number of active logs” on page 15
- “Using IBM z/OSMF to automate IBM MQ” on page 15
- “z/OS: Changes to exits” on page 19

Related information:

 [Video: IBM MQ Version 9.0 overview video \(YouTube\)](#)

[What's new and what's changed in IBM MQ Explorer](#)

 [System Requirements for IBM MQ](#)

 [IBM MQ, WebSphere MQ, and MQSeries product readmes web page](#)

New family features

 **V 9.0.0**

IBM MQ Version 9.0.0 delivers a new mode of operation and also support for non-IBM Java™ runtime environments (JREs) for Advanced Message Security, web addressable access to the Client Channel Definition Table (CCDT), support for the IBM MQ Version 9.0 resource adapter in WebSphere Application Server traditional, enhanced Unicode data support across all platforms, and logging enhancements for the Protocol Bridge Agent in Managed File Transfer.

- “Additional quality of protection for AMS” on page 12
- “Web addressable access to the client channel definition table (CCDT)” on page 12
- “AMS supported in non-IBM JREs in Java clients” on page 12
- “Updated Resource Adapter for traditional WebSphere Application Server” on page 12
- “Enhanced Unicode data conversion support” on page 12
- “Managed File Transfer Protocol Bridge Agent logging enhancements” on page 13

Additional quality of protection for AMS

To complement the existing Integrity and Privacy privacy policies, Advanced Message Security (AMS) provides a new, third alternative, Confidentiality (Encryption only with optional key reuse), in IBM MQ Version 9.0.

Significant CPU cost savings can be made with Confidentiality policies through symmetric key reuse. This new mode of operation continues to use the PKCS#7 format to share a symmetric encryption key. However, there is no digital signature, which eliminates some of the per message asymmetric key operations. The symmetric key still needs to be encrypted with asymmetric key operations for each recipient, but the symmetric key can be optionally reused over multiple messages that are destined for the same recipients. If key reuse is permitted by policy, then only the first message requires asymmetric key operations. Subsequent messages only need to use symmetric key operations. For more information, see Qualities of protection available with AMS.

Web addressable access to the client channel definition table (CCDT)

IBM MQ Version 9.0 improves the ability for clients to remain connected to IBM MQ queue managers by hosting the CCDT in a central location that is accessible through a URI, removing the need to individually update the CCDT for each deployed client. A client channel definition table can be located through a URL in any of the following ways:

- By programming using MQCNO
- By using environment variables
- By using mqclient.ini file stanzas.

Note: You can use the environment variable option only for native programs connecting as clients, that is C, COBOL, or C++ applications. The environment variables have no effect for Java, JMS or managed .NET applications.

For more information, see Web addressable access to the client channel definition table.

AMS supported in non-IBM JREs in Java clients

In earlier releases, the AMS relied on IBM-provided encryption packages that were included in the Java runtime environment (JRE) or Java Secure Socket Extension (JSSE) shipped with IBM MQ and other IBM products.

From Version 9.0, the AMS has been redesigned to use an alternative crypto library, the open source Bouncy Castle implementation, which is built into the IBM MQ classes for Java and IBM MQ classes for JMS, not the JRE. This means that you can now use alternative JREs without needing to install additional libraries. For more information, see Support for non-IBM JREs.

Updated Resource Adapter for traditional WebSphere Application Server

The IBM MQ Version 9.0 resource adapter is pre-installed within WebSphere Application Server traditional Version 9.0. Therefore, there is no requirement to install a new resource adapter.

Note: An IBM MQ Version 9.0 resource adapter can connect in CLIENT or BINDINGS transport mode to any in-service IBM MQ queue manager.

Enhanced Unicode data conversion support

From Version 9.0, IBM MQ supports all Unicode characters defined in the Unicode 8.0 standard in data conversion, including full support for UTF-16. For more information, see Data conversion.

In addition, a new file `ccsid_part2.tbl` is provided, which takes precedence over the `ccsid.tbl` file and:

- Allows you to add or modify CCSID entries
- Specify default data conversion
- Specify data for different command levels

The `ccsid_part2.tbl` file is applicable to the following platforms only:

-  Linux - all versions
-  Solaris
-  Windows

For more information, see The `ccsid_part2.tbl` file.

Managed File Transfer Protocol Bridge Agent logging enhancements



From Version 9.0, Managed File Transfer provides a new logging feature to capture interactions between the Protocol Bridge Agent and FTP/SFTP/FTPS file servers. The agent log is set to a level that enables the collection of first hand diagnostic information. Logging is enabled and disabled by using the new command `fteSetAgentLogLevel` or by using the `agent.properties` file.

For more information, see `fteSetAgentLogLevel` and The `agent.properties` file.

New features for Multiplatforms



IBM MQ for Multiplatforms Version 9.0.0 for delivers the addition of LDAP authorization on Windows, and the ability to use the arrow keys on UNIX when you use `runmqsc`.

- “Performance monitoring API and enhanced activity trace available through publish/subscribe”
-  “LDAP authorization on Windows platforms” on page 14
-  “runmqsc command line editor for UNIX” on page 14

Performance monitoring API and enhanced activity trace available through publish/subscribe

IBM MQ Version 9.0 publishes information messages to a range of system level topic strings. An authorized user can subscribe to these topics to receive monitoring information for the queue manager and application activity on it.

Performance monitoring API

In IBM MQ Version 9.0, statistics are published to a system topic under `$/SYS/MQ/INFO/QMGR` that help users monitor resources.

These statistics can be viewed by running the `amqsrua` sample, or by writing an application that subscribes to the resource monitoring system topic in a similar way to `amqsrua`. For more information see Monitoring system resource usage by using the `amqsrua` command and Developing your own resource monitoring program.

Enhanced application activity trace

Application activity trace produces detailed information about the behavior of applications that are connected to a queue manager. IBM MQ Version 9.0 adds the ability to dynamically subscribe to this information rather than collecting the information through queue manager level configuration. For more information, see Subscribing to application activity trace information.

For more information about how system topics in queue manager topic trees are used for resource monitoring and for application activity trace, see System topics for monitoring and activity trace.

LDAP authorization on Windows platforms

The LDAP authorization, originally included on UNIX platforms in IBM MQ Version 8.0.0, Fix Pack 2, is extended to the Windows platforms. For more information, see Overview of LDAP authorization.

runmqsc command line editor for UNIX

IBM MQ Version 9.0 adds the ability to use the up and down arrow keys, and tab key commands to control **runmqsc** commands on UNIX.

You can use the following keys:

- Use the up arrow key to recall the previous **runmqsc** command
- Use the down arrow key to recall the next **runmqsc** command
- Use the tab key to offer completion options for the next keyword in the command

For more information, see The **runmqsc** command line editor on UNIX.

New features for z/OS

For Version 9.0.0, IBM MQ for z/OS delivers additional data manager pageset statistics and new IBM z/OS Management Facility (z/OSMF) workflows and templates that enable simplified queue manager and queue manager resource provisioning. Version 9.0 also delivers a number of features that were previously made available in Version 8.0 fix packs. These features include the use of JMS in a CICS® OSGi JVM server and IMS™, enforcing lower expiration times in messages, and increasing the number of active log data sets.

- “Data manager page set statistics”
- “JMS support within CICS OSGI Server”
- “JMS support within IMS Server”
- “Improved control of messages by using message expiry” on page 15
- “Increased number of active logs” on page 15
- “Using IBM z/OSMF to automate IBM MQ” on page 15

Data manager page set statistics

IBM MQ Version 9.0 provides new pageset usage information within existing queue manager SMF records to facilitate better management of local queues within a queue manager by recording information such as pageset input-output rates and highest usage. For more information, see Data manager pageset data records.

JMS support within CICS OSGI Server

Standards-based messaging support for applications that run within the CICS OSGI Server (CICS Transaction Server Version 5.2 or later) environment is provided through the use of IBM MQ classes for JMS. For more information, see Using IBM MQ classes for JMS in a CICS OSGi JVM server.

JMS support within IMS Server

Standards-based messaging support within an IMS Version 13 environment is provided through the use of IBM MQ classes for JMS. For more information, see Using IBM MQ classes for JMS in IMS.

Improved control of messages by using message expiry

An override capability for message expiry within a queue manager is provided through the use of the CAPEXPY attribute. This capability is available for point-to-point messaging or publish and subscribe. For more information, see Enforcing lower expiration times.

Increased number of active logs

The number of active logs that are available to be defined to a single queue manager is increased from 31 to up to a maximum of 310. This increases the availability of a queue manager in failure scenarios. For more information, see Log data set definitions.

Using IBM z/OSMF to automate IBM MQ

From IBM MQ Version 9.0, you can use the IBM z/OS Management Facility (z/OSMF) to more easily manage the day-to-day operations of IBM MQ. For more information, see Using IBM z/OSMF to automate IBM MQ.

Related concepts:

“z/OS: Prerequisites for Version 9.0” on page 19

Some prerequisites have changed for Version 9.0.

“z/OS: Changes to exits” on page 19

From IBM MQ for z/OS Version 9.0.0, exits that rely upon being able to modify static storage within the load module must be modified to be reentrant.

“z/OS: Changes to OPMODE” on page 20

In previous versions of IBM MQ, if a queue manager was started with OPMODE=(COMPAT,vrm), new functions added at command level vrm would always be available. In IBM MQ Version 9.0 this is no longer the case.

“New z/OS messages” on page 24

A number of new messages have been added for IBM MQ for z/OS Version 9.0.0.

What's changed in Version 9.0.0




Changes to functions and resources in IBM MQ Version 9.0.0 are described in this section. Review these changes before upgrading queue managers to the latest product version and decide whether you must plan to make changes to existing applications, scripts, and procedures before starting to migrate your systems.

For additional information about system requirements for Version 9.0, see the System Requirements for IBM MQ Version 9.0 web page. You can use the Software Product Compatibility Reports (SPCR) tool to find information on supported operating systems, system requirements, prerequisites, and optional supported software. The web page includes more information about the SPCR tool and links to reports for each supported platform.

Related information:

What's new and what's changed in IBM MQ Explorer

 System Requirements for IBM MQ Version 9.0

 IBM MQ, WebSphere MQ, and MQSeries product readmes web page

Changes to supported platforms

For IBM MQ Version 9.0, some changes are made to the platforms that are supported.


From Version 9.0, IBM MQ no longer supports components on:

- Windows x86-32
- Linux x86-32
- Linux POWER System - Big Endian
- IBM 4690

Furthermore, Microsoft Visual Studio 2005 is no longer supported.

Related information:


Checking requirements

 System Requirements for IBM MQ

Changes to commands

 V 9.0.0

A number of commands have changed at IBM MQ Version 9.0

- **fteCreateWebAgent** command
- **fteMigrateAgent** command
- “SET POLICY, SETMQMSPL, and setmqspl commands”
-  “DISPLAY SYSTEM command” on page 17

fteCreateWebAgent command

The **fteCreateWebAgent** command is removed at Version 9.0. See Removal of Managed File Transfer Web Gateway support.

fteMigrateAgent command

The **fteMigrateAgent** command has been changed to remove support for Web agents and to display an error message if you attempt to migrate a Managed File Transfer Web agent from an earlier release to Version 9.0. See Removal of Managed File Transfer Web Gateway support.

SET POLICY, SETMQMSPL, and setmqspl commands

The **SET POLICY** and **SETMQMSPL** commands have a new parameter **KEY REUSE**, and the **setmqspl** command has a new parameter **Reuse**, that are used in conjunction with the Advanced Message Security additional Quality of Protection of confidentiality.

See Qualities of protection available with AMS for further information.

 z/OS

DISPLAY SYSTEM command

The DISPLAY SYSTEM command has been changed to display the level of new functions that are available, in addition to the version of IBM MQ for z/OS that the queue manager can fall back to.

Command level changes for Version 9.0

▶ V 9.0.0

The command level changes to 900 at IBM MQ Version 9.0

Related information:

strmqm

Deprecated, stabilized and removed features

▶ V 9.0.0

A number of features are deprecated, stabilized or removed from IBM MQ Version 9.0.

- ▶ IBM i “Deprecation of compatibility mode for RPG applications on IBM i”
- “Deprecation of IBM MQ transport for SOAP” on page 18
- “Deprecation of useJNDI activation specification property” on page 18
- “Deprecation of IBM MQ support for Microsoft Active X and Microsoft Visual Basic 6.0” on page 18
- “Stabilization of IBM MQ classes for Java” on page 18
- Removal of Managed File Transfer Web Gateway support

▶ IBM i

Deprecation of compatibility mode for RPG applications on IBM i

From Version 9.0, IBM MQ no longer provides support for RPG applications that use dynamic linkage known as compatibility mode. This mode of operation was needed for applications that are written before MQSeries® Version 5.1, and subsequent versions of the product provided a compatible runtime environment for these applications, even though the copybooks needed for compiling them were removed in IBM WebSphere MQ Version 6.0. Dynamic linkage (compatibility mode) was provided by the following programs in library QMQM, which are removed at IBM MQ Version 9.0:

- AMQVSTUB
- AMQZSTUB
- QMQM
- MQCLOSE
- MQCONN
- MQDISC
- MQGET
- MQINQ
- MQOPEN
- MQPUT
- MQPUT1
- MQSET

From Version 9.0, applications that use this compatibility mode of operation need to be recompiled to use the static bound MQ calls that are provided by the LIBMQM and LIBMQM_R service programs. Sample programs, such as AMQ3PUT4 and AMQ3GET4, show you how to use this programming model. For more information about using these MQ calls, see IBM i Application Programming Reference (ILE/RPG).

Notes:

- You need to recode applications, currently using the CALL 'QMOM' interface, to use the LIBMQM service program instead.

The program objects and service programs in the preceding list, for example, QMQM, MQCONN, MQPUT, AMQVSTUB, and AMQZSTUB, are removed in IBM MQ Version 9.0, and applications that were coded to use compatibility mode cease to work.

- If applications are bound to the LIBMQM service program at IBM MQ Version 8.0, you should not need to recompile or relink those applications at IBM MQ Version 9.0.
- It is not possible to install more than one version of IBM MQ for IBM i on the same partition.

Deprecation of IBM MQ transport for SOAP

From Version 9.0, the IBM MQ transport for SOAP is deprecated. This includes the following product features:

- IBM MQ Java Listener
- IBM MQ .NET 1 and 2 Listener
- IBM MQ Java Axis2 Client
- IBM MQ Java client (deprecation already announced in Version 8.0)
- IBM MQ .NET 1 and 2 clients (deprecation already announced in Version 8.0)
- IBM MQ bridge for HTTP (deprecation already announced in Version 8.0)

Deprecation of useJNDI activation specification property

The useJNDI property is deprecated at IBM MQ Version 9.0 as the DestinationLookup property is equivalent and is the JMS 2.0 specification defined way of implementing the same function. For more information, see *Configuring the resource adapter for inbound communication*.

Deprecation of IBM MQ support for MicrosoftActive X and Microsoft Visual Basic 6.0

IBM MQ support for MicrosoftActive X and Microsoft Visual Basic 6.0 is deprecated. IBM MQ classes for .NET are the recommended replacement technology. See *Using the Component Object Model Interface (IBM MQ Automation Classes for ActiveX)* and *Coding in Visual Basic*.

Stabilization of IBM MQ classes for Java

IBM will make no further enhancements to the IBM MQ classes for Java and they are functionally stabilized at the level shipped in IBM MQ Version 8.0. Existing applications that use the IBM MQ classes for Java will continue to be fully supported, but this API is stabilized, so new features will not be added and requests for enhancements rejected. Fully supported means that defects will be fixed together with any changes necessitated by changes to IBM MQ System Requirements.

Removal of Managed File Transfer Web Gateway support

Before Version 9.0, the Managed File Transfer Web Gateway allowed you to transfer files to Managed File Transfer agents and retrieve transfer status using a HTTP client. The Web Gateway was deprecated at IBM MQ Version 8.0, and is removed at Version 9.0.

The `fteCreateWebAgent` command is removed at Version 9.0. If you attempt to migrate a web agent from an earlier release to Version 9.0, an error message is displayed to explain that the migration of a web agent is not supported (see `fteMigrateAgent` (migrate a WebSphere MQ File Transfer Edition Version 7.0 agent to IBM WebSphere MQ Version 7.5 or later)).

Related information:

Deprecated cipherspecs

z/OS: Prerequisites for Version 9.0



Some prerequisites have changed for Version 9.0.

IBM MQ for z/OS Version 9.0 requires z/OS Version 2.1 or higher.

Advanced Message Security, when configured to use the new quality of protection, CONFIDENTIALITY, with encryption algorithm (**ENCALG**) AES256 requires IBM System z10 or later hardware. For more information about this new quality of protection, see Qualities of protection available with AMS.

For more information on supported operating systems, system requirements, prerequisites, and optional supported software, see the System Requirements for IBM MQ Version 9.0 web page.

z/OS: Changes to exits



From IBM MQ for z/OS Version 9.0.0, exits that rely upon being able to modify static storage within the load module must be modified to be reentrant.

Writing channel exit programs on z/OS stipulates that IBM MQ exits must be reentrant and capable of running anywhere in virtual storage.

Non-reentrant C programs are those that might use variables declared with the static storage class, or variables receiving the extern storage class, or might contain writable static strings.

Non-reentrant exits have, technically, never been supported in IBM MQ for z/OS, but prior to IBM MQ for z/OS Version 9.0.0, it was possible for some of these exits to process successfully

From IBM MQ for z/OS Version 9.0.0, exits that rely upon being able to modify static storage within the load module no longer function correctly. You should modify such exits to be reentrant.

You might achieve this in some cases by:

- Moving static or global variables inside the scope of a function, or
- Dynamically allocating storage used for any static variables and using, for example, the IBM MQ **ExitUserArea**, or other operating system facilities, such as name or token services to obtain addressability.

z/OS: Changes to OPMODE

z/OS V 9.0.0

In previous versions of IBM MQ, if a queue manager was started with `OPMODE=(COMPAT,vrm)`, new functions added at command level `vrm` would always be available. In IBM MQ Version 9.0 this is no longer the case.

The availability of new functions and backward migration for IBM MQ for z/OS is controlled by the `OPMODE` parameter in the `CSQ6SYSP` macro.

LTS

Note the following:

- IBM WebSphere MQ Version 7.1 or earlier new functions are always available at Version 9.0.
- IBM MQ Version 8.0 new functions are only available at Version 9.0 when the queue manager has been started with `OPMODE=(NEWFUNC,800)` or `OPMODE=(NEWFUNC,900)`.

This means that IBM MQ Version 8.0 new functions are **not** available, should the queue manager be started with `OPMODE=(COMPAT,900)`.

Important: Starting a Version 9.0 queue manager with `OPMODE=(COMPAT,900)`, when it has previously been running at `OPMODE=(NEWFUNC,800)`, stops previously enabled Version 8.0 new functions running.

For example, if the BSDS has been converted to Version 2, to enable 8-byte log RBA, the queue manager will not be able to read the BSDS and cannot start. See [z/OS: Switching from new function mode to compatibility mode for a full list of functions affected](#).

- To prevent Version 8.0 new functions from being unintentionally made unusable, the procedure to migrate to IBM MQ Version 9.0 LTS has changed. You should not recompile the system parameter (`ZPARM`) module during the process of migrating to IBM MQ Version 9.0 LTS.

The `DISPLAY SYSTEM` command has been changed to display the level of new functions that are available, in addition to the version of IBM MQ for z/OS that the queue manager can fall back to.

Related information:

Checking requirements



System Requirements for IBM MQ

New, changed and removed messages in Version 9.0.0

V 9.0.0

A summary of the new messages that have been added for IBM MQ Version 9.0.0, with links to further information. Any messages that have been changed or removed are also listed.

Related information:

What's new and what's changed in IBM MQ Explorer

 System Requirements for IBM MQ

 IBM MQ, WebSphere MQ, and MQSeries product readmes web page

New IBM MQ AMQ messages

A number of new messages have been added for IBM MQ Version 9.0.0.

AMQ3xxx: Utilities and tools

AMQ3000 This application has ended. See above for any problems found.

AMQ3001 No Certificate could be found for the Queue Manager *insert_3*

AMQ3002 No personal certificate could be found for the client in the client's key repository.

AMQ3003 No key repository could be found for the queue manager *insert_3*

AMQ3004 No key repository could be found for the client

AMQ3005 No stash file could be found for the key repository in use by *insert_3*

AMQ3006 No stash file could be found for the key repository in use by the client

AMQ3007 The SSLKEYR attribute of the queue manager is blank.

AMQ3008 No key repository could be found for the queue manager *insert_3*

AMQ3009 No key repository could be found for the Client

AMQ3010 Invalid access rights found for the key repository in use by queue manager *insert_3*

AMQ3011 Invalid access rights found for the stash file in use by queue manager *insert_3*

AMQ3012 Invalid access rights found for the key repository in use by the client

AMQ3013 Invalid access rights found for the stash file in use by the Client

AMQ3014 Invalid password found in the stash file for key repository *insert_3*

AMQ3015 Certificate has a 'valid-from' time in the future.

AMQ3016 Certificate has expired.

AMQ3017 *insert_3* certificate failed authentication with the *insert_4* key repository

AMQ3018 Client connection refused by server.

AMQ3019 Unable to connect to queue manager *insert_3*.

AMQ3020 Unable to open the queue manager *insert_3*.

AMQ3021 Unable to open the queue *insert_4* on *insert_3*.

AMQ3022 Unable to inquire the queue manager *insert_3*

AMQ3023 Unable to put to queue *insert_4* on *insert_3*.

AMQ3024 Unable to get from queue *insert_4* on *insert_3*.

AMQ3025 Unable to close PCF reply queue

AMQ3026 Unable to disconnect from *insert_3*.

AMQ3027 SSL Certificate Revocation List parameter found (SSLCRLNL = *insert_3*)

AMQ3028 Relative path given for client key repository.

AMQ3029 Value specified in Queue Manager CERTLABL attribute is not valid.

AMQ3030 Value specified in Client CERTLABL attribute is not valid.

AMQ3031 This program encountered a internal error and had to quit.

AMQ3032 There was a problem with the arguments supplied for the client checks.

AMQ3033 There was a problem with the arguments supplied.

AMQ3034 There was a problem with the arguments supplied.

AMQ3035 No Client user name or Client Certificate label provided.

AMQ3036 The Channel *insert_3* is not SSL enabled but has a CERTLABL value of *insert_4*.

AMQ3037 No certificate could be found for the channel *insert_3*.

AMQ3038 Unable to find a channel called *insert_4* on Queue Manager *insert_3*.

AMQ3039 No personal certificate could be found for the client in the client's key repository and the channel requires a client certificate.

AMQ3040 The user running this program does not have access rights to examine a directory

AMQ3041 A filename is longer than this application can handle.

AMQ3042 The user running this program does not have access rights to examine a file

AMQ3043 An unknown error was encountered while trying to access a file.

AMQ3044 Unable to verify if the mqm user can access a file.

AMQ3045 Unable to retrieve a password.

AMQ3046 *insert_3s* certificate failed validation.

AMQ3047 Invalid access rights found for the parent directory of the key repository in use by queue manager *insert_3*.

AMQ3048 Invalid access rights found for the parent directory of the key repository in use by the client.

AMQ3049 The queue manager is not registered as an application with Digital Certificate Manager (DCM).

AMQ3050 The port number *insert_3* was invalid.

AMQ3051 A certificate failed authentication.

AMQ3052 A certificate label has been specified on channel *insert_3* while system store is in use.

AMQ3053 The usage of *insert_3* is incorrect.

AMQ3054 The usage of *insert_3* is incorrect.

AMQ6xxx: Common Services

AMQ6317 MBCS error

AMQ6318 SBCS error

AMQ6720 A disk quota exhausted condition was encountered when formatting a new log file in location *insert_3*.

AMQ690C (IBM i) IBM MQ

AMQ690D (IBM i) MQ

AMQ690E (IBM i) &WS_long

AMQ690F (IBM i) IBM MQ_legacy_long

AMQ691A (IBM i) IBM MQ_cert_label

AMQ7xxx: IBM MQ product: Utilities and tools

AMQ7823 Disk quota exhausted.

AMQ8xxx: Administration

AMQ8521 Command completion and history unavailable.

AMQ8521 (AIX) Command completion and history unavailable.

AMQ8521 (HP-UX) Command completion and history unavailable.

AMQ8921 MQCB call failed with reason code *insert_1*.

AMQ8C0D (IBM i) Limit security PCF commands

AMQ8D0C (IBM i) Symmetric key reuse

AMQ9xxx: Remote

AMQ9088 Key reuse count:

AMQ9089 Invalid value specified for the key reuse argument.

AMQ9090 Client application lacks one or more required AMS capabilities.

AMQ9091 Key reuse is not valid for policy.

AMQ9092 Message with confidentiality protection has been put into the queue *insert_3*.

AMQ9093 *insert_3* - message was encrypted using *insert_5*.

AMQ9094 &AMS_short security policy internal error: message could not be protected: *insert_5*
error code *insert_3* (*insert_1*) in *insert_4*.

AMQ9095 &AMS_short security policy internal error: message could not be unprotected: *insert_5*
error code *insert_3* (*insert_1*) in *insert_4*.

AMQ9096 &AMS_short distribution list error.

AMQ9795 The client channel definition could not be retrieved from its URL, error code (*insert_1*).

AMQ9796 Slow URL retrieval for *insert_3*.

Related information:

IBM MQ AMQ messages

Additional quality of protection for AMS

Web addressable access to the client channel definition table (CCDT)

New and removed Managed File Transfer messages

A number of new messages have been added for Managed File Transfer Version 9.0.0. Some messages available in earlier releases have been removed.

New messages

BFGAG0001 - BFGAG9999

BFGAG0182 Web agent is not supported.

BFGCH0001 - BFGCH9999

BFGCH0120 An invalid component name, *insert_0* specified.

BFGCH0121 An invalid operation type, *insert_0* was specified.

BFGCL0001 - BFGCL9999

BFGCL0745 Creation of Web agent is not supported.

BFGCL0746 The command was run without a logAgent parameter. Run the command with the -h parameter to see usage information.

BFGCL0747 The log specification *insert_0* is not valid.

BFGCL0748 The log request was successfully sent to agent *insert_0*.

BFGCL0749 Migration of a Web agent *insert_0* is not supported.

BFGUT0001 - BFGUT9999

BFGUT0025 Protocol bridge agent log specification has changed to *insert_0*.

BFGUT0026 Invalid Protocol bridge agent log specification. The log specified is *insert_0*.

BFGUT0027 The system is unable to log protocol bridge command/response messages to the protocol bridge agent event log file in directory *insert_0* (reason: *insert_1*). Agent event log messages will be output to the console.

BFGUT0028 The system is unable to access the current protocol bridge agent event log file: (ErrorManager code: *insert_0*) *insert_1* *insert_2*

BFGUT0029 Protocol bridge agent log filter specification has changed to *insert_0*.

BFGUT0030 Invalid Protocol bridge agent log filter specified. The filter specified is *insert_0*.

BFGUT0031 Failed to set protocol bridge agent log filter. The filter specified is *insert_0*.

Removed messages

BFGAP0001 - BFGAP9999

BFGAP0064 The specified logger name is blank.

BFGAP0065 An internal error has occurred. Unable to complete the command because the logger directory *insert_0* does not exist.

BFGDM0001 - BFGDM9999

BFGDM0122 The batch size for monitor *insert_0* has been set to 1 at execution time because the batch size for a monitor on a queue cannot exceed this value.

Related information:

Managed File Transfer diagnostic messages

Managed File Transfer Protocol Bridge Agent logging enhancements

Removal of Managed File Transfer Web Gateway support

A number of features are deprecated, stabilized or removed from IBM MQ Version 9.0.

New z/OS messages



A number of new messages have been added for IBM MQ for z/OS Version 9.0.0.

Messages added in IBM MQ for z/OS Version 9.0.0

Coupling Facility manager messages

CSQE162E *csect-name* Structure *struc-name* could not be deleted, RC=*return-code* reason=*reason*

Recovery log manager messages

CSQJ165I zHyperWrite bypassed for active log data set *dsname*

CSQJ166E PPRC configuration is inconsistent for active log copy *n*

CSQJ167E zHyperWrite enabled but no active logs have PPRC configured

Distributed queuing messages

CSQX130E *csect-name* *queue-name* is defined on a non-recoverable CF structure



Advanced Message Security messages

CSQ0469E Invalid value specified for key reuse argument

CSQ0471E Key reuse not valid for policy

Related information:

Messages for IBM MQ for z/OS

What's new and changed in Version 9.0.x Continuous Delivery



Following the initial release of IBM MQ Version 9.0, new function and enhancements are made available by incremental updates within the same version and release as modification releases, for example Version 9.0.3.

Continuous Delivery releases (CD releases) deliver new functional enhancements, in addition to fixes and security updates, on a much shorter cadence, so providing much more rapid access to those new functions. This delivery model is intended for systems where applications want to exploit the very latest capabilities of IBM MQ.

The individual features provided by a CD release are stand-alone functions but may increase in capability in subsequent CD releases, enhancing earlier CD features and forming a suite of related capabilities.


It is also possible for newly created features in one CD release to be modified in later CD releases to improve the user experience. Where migration to the improved capability is required it will be documented.

For further information see IBM MQ release types and IBM MQ FAQ for Long Term Support and Continuous Delivery releases.

Related concepts:

 “What's new and changed in IBM MQ Version 9.0.0” on page 10



IBM MQ Version 9.0 introduces a new continuous delivery and support model. The initial release under this new model delivers a number of new and enhanced capabilities.

 “What's changed in Version 9.0.0.x Long Term Support” on page 80

The Long Term Support (LTS) release is a recommended product level for which support, including defect and security updates, are provided over a specified period of time. This version is intended for systems which demand a long term deployment and maximum stability.

“What was new and changed in earlier versions” on page 89

Links to information about new features and changes to functions and resources, including stabilizations, deprecations and removals, that occurred in versions of the product before IBM MQ Version 9.0.

  “Release and platform icons in the product documentation” on page 94

In the IBM MQ Version 9.0 product documentation, icons are used to identify release at which a feature is made available. Icons are also used to identify information that applies to a specific platform or group of platforms only.

Related information:

 [System Requirements for IBM MQ](#)

 [IBM MQ, WebSphere MQ, and MQSeries product readmes web page](#)

What's new and changed in IBM MQ Version 9.0.5



IBM MQ Version 9.0.5, which is a Continuous Delivery (CD) release, delivers a number of new and enhanced features on AIX®, Linux, Windows, and z/OS.

Related information:

 System Requirements for IBM MQ

 IBM MQ, WebSphere MQ, and MQSeries product readmes web page

What's new in IBM MQ Version 9.0.5




IBM MQ Version 9.0.5 delivers a number of new and enhanced features on AIX, Linux, Windows, and z/OS.

The individual features provided by a CD release are stand-alone functions but may increase in capability in subsequent CD releases, enhancing earlier CD features and forming a suite of related capabilities.




New IBM MQ for Multiplatforms features

The following new features are available with both IBM MQ base product entitlement, and with IBM MQ Advanced entitlement:

- Further enhancements to the REST API - View channels and run mqsc commands on a remote queue manager
- “Improvements to error logging” on page 28
- “New ImplSyncOpenOutput tuning parameter” on page 28
- “New method for LDAP authorization” on page 28
- “AMQP service starting by default” on page 29
-  “IBM MQ custom channel for WCF enabled for send replying messages in MQSTR format” on page 29



The following new features are available only with IBM MQ Advanced entitlement:

-  “Disaster recovery replicated data queue managers (DR RDQMs)” on page 29
- “New queue manager attribute for identifying IBM MQ Advanced entitlement” on page 29
- Further enhancements to the REST API - Managed file transfer support
- “New fteListMonitors parameter for bulk backing up of MFT resource monitors” on page 27
- “Ability to use colon separated lists of trace specifications in certain Managed File Transfer Logger commands” on page 28



New IBM MQ for z/OS features

The following new feature is available both with IBM MQ for z/OS base product entitlement, and with IBM MQ Advanced for z/OS or IBM MQ Advanced for z/OS, Value Unit Edition entitlement:

- Further enhancements to the REST API - View channels and run mqsc commands on a remote queue manager

The following new features are available with IBM MQ Managed File Transfer for z/OS, IBM MQ Advanced for z/OS or IBM MQ Advanced for z/OS, Value Unit Edition entitlement:

- Further enhancements to the REST API - Managed file transfer support
- “New fteListMonitors parameter for bulk backing up of MFT resource monitors” on page 27



- “Ability to use colon separated lists of trace specifications in certain Managed File Transfer Logger commands” on page 28

  The following new features are available only with IBM MQ Advanced for z/OS, Value Unit Edition entitlement:

- “Advanced Message Security on JMS for z/OS queue managers” on page 30
- “AMS security policy information is available on MQOPEN, instead of requiring a separate query” on page 30

New features on all supported platforms:    


IBM MQ Version 9.0.5 delivers a number of new features on all platforms that are supported by this release, that is on AIX, Linux, Windows, and z/OS.


- “Further enhancements to the REST API”
-  “Ability to use colon separated lists of trace specifications in certain Managed File Transfer Logger commands” on page 28
-  “New fteListMonitors parameter for bulk backing up of MFT resource monitors”

Further enhancements to the REST API

In Version 9.0.5, the REST API has been extended to:

- View channels and run mqsc commands on a remote queue manager. That is, you can now specify a remote queue manager in the resource URL for the `/admin/qmgr/{qmgrName}/channel` and `/admin/action/qmgr/{qmgrName}/mqsc` resource URLs.

For more information about these resources, see REST API resources.

-  Add support for certain Managed File Transfer commands, including:
 - Listing transfers - see Obtaining an MFT list of transfers for more information.
 - Details about file transfer agents - see Obtaining the MFT agent status for more information.

Important: The MFT portion of the REST API does not apply to the IBM MQ Appliance.



New `fteListMonitors` parameter for bulk backing up of MFT resource monitors

From Version 9.0.5, the `-od` parameter is added to the `fteListMonitors` command. By specifying this parameter, you can back up more than one resource monitor at a time by exporting their definitions in bulk to a specified directory. Each resource monitor definition is saved to a separate XML file with a name in the format `agent name.monitor name.xml`.

The `-od` parameter is particularly useful if you have a large number of resource monitors that you want to back up because you need to run the `fteListMonitors -od` command once only, instead of having to run the `fteListMonitors -ox` command separately for each resource definition, or use a separate script to run the `fteListMonitors -ox` command for each resource monitor.

For more information, see Backing up and restoring MFT resource monitors and `fteListMonitors: list MFT resource monitors`.







Ability to use colon separated lists of trace specifications in certain Managed File Transfer Logger commands

From Version 9.0.5 onwards, you can specify a colon-separated list of trace specifications in the `fteSetAgentTraceLevel` and `fteSetLoggerTraceLevel` commands.

This change enables you to specify trace specifications in exactly the same way as you do for the existing `agent.properties` and `logger.properties` files.

New AIX, Linux and Windows features:    

On AIX, Linux and Windows, IBM MQ Version 9.0.5 delivers a number of new features.

- “Improvements to error logging”
- “New method for LDAP authorization”
- “AMQP service starting by default” on page 29
-  “IBM MQ custom channel for WCF enabled for send replying messages in MQSTR format” on page 29
-   “Disaster recovery replicated data queue managers (DR RDQMs)” on page 29
-  “New queue manager attribute for identifying IBM MQ Advanced entitlement” on page 29

Improvements to error logging

Version 9.0.5 introduces various improvements to the management and output of error logs. The main changes are, that you can:

- Log diagnostic messages, using additional file services and `syslog` on UNIX platforms, as well as `AMQERR01.LOG`.
- Use JSON for the description of the messages, as well as the existing format; see JSON format diagnostic messages.
- Reformat a log into another language or style; see `mqrc`.

For more information, see Diagnostic message services, and `QMErrorLog` service.

New `ImplSyncOpenOutput` tuning parameter

From Version 9.0.5, a new tuning parameter in the `qm.ini` file, **`ImplSyncOpenOutput`** improves performance on put operations of persistent messages outside of syncpoint. For more information, see Implicit syncpoint and Tuning parameters.

New method for LDAP authorization

Version 9.0.5 introduces a new authorization method for users and groups in an LDAP repository.

This method adds support for LDAP configurations where group membership is indicated by an attribute in the group entry containing the short user names of all the users that belong to the group.

This new authorization method is enabled by setting **`AUTHORMD`** to `SRCHGRPSN` on an authentication information object with an **`AUTHTYPE`** of `IDPWLDPAP`.

For more information, see the **`AUTHORMD`** parameter of the `DEFINE AUTHINFO` command.

AMQP service starting by default

From Version 9.0.5, the AMQP service is started by default, which means that:

- AMQP channels can be started and stopped as soon as the queue manager is started.
- The AMQP JVM will be running, and show up in ps/top/task manager as soon as a queue manager is started, but if, and only if, the MQSeriesAMQP component is installed.

For more information, see [Creating and using AMQP channels](#).

Windows

IBM MQ custom channel for WCF enabled for send replying messages in MQSTR format

From Version 9.0.5, if you put a WCF request message onto a queue using IBM MQ Custom channel for WCF, in MQMFT_STRING (MQSTR) format, you can select to receive the reply message in the same MQMFT_STRING (MQSTR) format.

For more information, see [Sending MQSTR formatted messages](#).

Linux

MQ Adv.

Disaster recovery replicated data queue managers (DR RDQMs)

From Version 9.0.5, you can build a disaster recovery solution based on replicated data queue managers (RDQMs).

You have a primary instance of a DR queue manager running on one server, and a secondary instance of the queue manager on another server (that acts as the recovery node). Data is replicated between the queue manager instances. If you lose your primary queue manager, you can manually make the secondary instance into the primary instance and start the queue manager, then resume work from the same place.

You can configure for synchronous replication of data between primary and secondary queue managers, or asynchronous replication.

For more information on DR RDQM, see [RDQM disaster recovery](#).

MQ Adv.

New queue manager attribute for identifying IBM MQ Advanced entitlement

At Version 9.0.4, a new queue manager attribute was added to enable z/OS queue managers with IBM MQ Advanced entitlement to identify themselves as Advanced queue managers. From Version 9.0.5, this capability is extended to all platforms.

For more information, see the **ADVCAP** attribute of the DISPLAY QMGR MQSC command, and the **MQIA_ADVANCED_CAPABILITY** attribute of the Inquire Queue Manager (**MQCMD_INQUIRE_Q_MGR**) PCFcommand.

New IBM MQ Advanced for z/OS, Value Unit Edition features:

IBM MQ Version 9.0.5 delivers the following new features that are available with IBM MQ Advanced for z/OS, Value Unit Edition entitlement.

- “Advanced Message Security on JMS for z/OS queue managers”
- “AMS security policy information is available on MQOPEN, instead of requiring a separate query”

Advanced Message Security on JMS for z/OS queue managers

Version 9.0.5 adds AMS support to a IBM MQ classes for JMS or IBM MQ classes for Java application, running on z/OS, using a client connection to a z/OS queue manager, licensed as IBM MQ Advanced for z/OS, Value Unit Edition.

For more information, see Advanced Message Security support.

AMS security policy information is available on MQOPEN, instead of requiring a separate query

When clients use Advanced Message Security (AMS), they can access the AMS security policy information for a given queue using MQOPEN, instead of having to use a separate query. This improves performance for queues that are not protected by AMS, on a queue manager that is AMS enabled.

What's changed in Version 9.0.5



Changes to functions and resources in Version 9.0.5 are described in this section. Review these changes before upgrading queue managers to the latest product version and decide whether you must plan to make changes to existing applications, scripts, and procedures before starting to migrate your systems.

For additional information about system requirements for Version 9.0, see the System Requirements for IBM MQ Version 9.0 web page. You can use the Software Product Compatibility Reports (SPCR) tool to find information on supported operating systems, system requirements, prerequisites, and optional supported software. The web page includes more information about the SPCR tool and links to reports for each supported platform.

Changes from earlier versions of IBM MQ

Review this information if you are moving from Long Term Support to Continuous Delivery, or if you are moving to Version 9.0.5 from an earlier Continuous Delivery release.

- “Removal of JSON4J.jar file and com.ibm.msg.client.mqlight package” on page 31
- “Additional permission for java.security.policy file” on page 31
- “New attribute to allow TLS v1.0 to be optionally disabled on a queue manager” on page 31
- “Change to output of fteDisplayVersion -v command” on page 32

Changes from earlier CD releases

Review this information if you are moving to Version 9.0.5 from an earlier Continuous Delivery release.

-  “Simplification of Cross-Site Request Forgery (CSRF) protection support” on page 32
- “BluemixRegistration stanza renamed to ReportingService stanza” on page 33
- “Changes to IBM MQ Console” on page 33

Changes in Version 9.0.5 to functions and resources from earlier versions of IBM MQ are described in this section.

Review this information if you are moving from Long Term Support to Continuous Delivery, or if you are moving to Version 9.0.5 from an earlier Continuous Delivery release.

For additional information about system requirements for Version 9.0, see the System Requirements for IBM MQ Version 9.0 web page. You can use the Software Product Compatibility Reports (SPCR) tool to find information on supported operating systems, system requirements, prerequisites, and optional supported software. The web page includes more information about the SPCR tool and links to reports for each supported platform.

- “Removal of JSON4J.jar file and com.ibm.msg.client.mqlight package”
- “Additional permission for java.security.policy file”
- “New attribute to allow TLS v1.0 to be optionally disabled on a queue manager”
- “Change to output of fteDisplayVersion -v command” on page 32

Removal of JSON4J.jar file and com.ibm.msg.client.mqlight package

The JSON4J.jar file and com.ibm.msg.client.mqlight package are not needed by the IBM MQ classes for Java and IBM MQ classes for JMS, therefore the following changes are made from Version 9.0.5:

- The JSON4J.jar file is removed from the *V.R.M.F-WS-MQ-Install-Java-All.jar* file, where *V.R.M.F* is the product version number, for example 9.0.5.0.
- The reference to JSON4J.jar file is removed from the class path statement within the manifest file for the com.ibm.mq.allclient.jar file.
- The package com.ibm.msg.client.mqlight is no longer included inside the com.ibm.mq.allclient.jar file.

See Installing the IBM MQ classes for JMS separately, What is installed for IBM MQ classes for JMS, and What is installed for IBM MQ classes for Java.

Additional permission for java.security.policy file

From Version 9.0.5, if your Java application uses the Java Security Manager, you must add a RuntimePermission to the java.security.policy file, otherwise, exceptions will be thrown to the application. This RuntimePermission is required by the client as part of managing the assignment and closure of multiplexed conversations over TCP/IP connections to queue managers.

For more information, see Running IBM MQ classes for Java applications under the Java Security Manager.

New attribute to allow TLS v1.0 to be optionally disabled on a queue manager

From Version 9.0.5, a new attribute is available in the qm.ini file, under the SSL stanza:

SSL:

```
AllowTLSV1=NO
```

If this attribute is set in the qm.ini file before the queue manager is started, the queue manager does accept inbound connections using the TLS v1.0 protocol. Similarly, if an LDAP connection is configured using an AUTHINFO object, only TLS 1.2 is used to communicate with the LDAP server if secure communication is enabled for the AUTHINFO object.

Alternatively, the **AMQ_TLS_V1_DISABLE** environment variable can be set for the environment used to start the queue manager, listener, and channel processes.

If either property is set, as well as disallowing TLS 1.0 connection attempts at the network layer, the queue manager's command server also rejects attempts to define or alter a channel definition to use a TLS 1.0 CipherSpec.

The default queue manager behavior is unchanged, such that TLS 1.0 connections continue to be accepted if the new attribute or environment variable is not set.

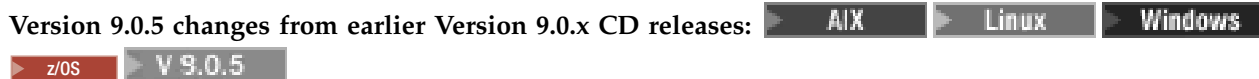
Change to output of **fteDisplayVersion -v** command

From Version 9.0.5, IBM MQ Light Service for Bluemix® JMS Provider is removed from the output of **fteDisplayVersion -v** command. Following this removal, you should review any automation that is based on the output from the **fteDisplayVersion** command.

Related concepts:

“What was new and changed in earlier versions” on page 89

Links to information about new features and changes to functions and resources, including stabilizations, deprecations and removals, that occurred in versions of the product before IBM MQ Version 9.0.



Changes to functions and resources introduced in earlier Version 9.0.x CD releases are described in this section.

Review this information if you are moving to Version 9.0.5 from an earlier Continuous Delivery release.

For additional information about system requirements for Version 9.0, see the System Requirements for IBM MQ Version 9.0 web page. You can use the Software Product Compatibility Reports (SPCR) tool to find information on supported operating systems, system requirements, prerequisites, and optional supported software. The web page includes more information about the SPCR tool and links to reports for each supported platform.

- “Simplification of Cross-Site Request Forgery (CSRF) protection support”
- “BluemixRegistration stanza renamed to ReportingService stanza” on page 33
- “Changes to IBM MQ Console” on page 33



Simplification of Cross-Site Request Forgery (CSRF) protection support

The CSRF protection support provided by the IBM MQ REST API has been simplified. These simplifications are as follows:

1. In Version 9.0.4 and earlier, callers of the REST API must copy the value of the `csrfToken` cookie, and set it in an HTTP header called `ibm-mq-rest-csrf-token`, if they want to use any HTTP verb other than GET. In Version 9.0.5 and onwards, the `csrfToken` cookie no longer exists. CSRF protection is provided by the presence of the `ibm-mq-rest-csrf-token` HTTP header in requests to the REST API. This update requires any applications that used the `csrfToken` cookie to be changed. For more information on this change, see:
 - Using HTTP basic authentication with the REST API
 - Using token-based authentication with the REST API from IBM MQ Version 9.0.5
2. The `mqRestCsrfExpirationInMinutes` configuration property, added in Version 9.0.4, has been removed

BluemixRegistration stanza renamed to ReportingService stanza

AIX **Linux** **Windows** From IBM MQ Version 9.0.5, the BluemixRegistration stanza, in the qm.ini file, has been renamed to be the ReportingService stanza. See qm.ini for details.

z/OS From IBM MQ Version 9.0.5, the BluemixRegistration stanza, in the CSQMCMIN dataset, has been renamed to be the ReportingService stanza. See Configuring the ReportingService stanza for details.

If you are upgrading to Version 9.0.5 from an earlier release of the product, you must migrate your existing BluemixRegistration stanza to the ReportingService stanza.

Changes to IBM MQ Console

The IBM MQ Console has changed in appearance. Some of the controls for the console itself have also changed. See Administration using the IBM MQ Console for details.

New, changed and removed messages in Version 9.0.5

AIX **Linux** **Windows** **z/OS** **V 9.0.5**

A summary of the new messages that have been added for IBM MQ Version 9.0.5, with links to further information. Any messages that have been changed or removed are also listed.

- “New and changed AMQ messages”
- “New, changed and removed REST API messages” on page 36
- “New and changed Managed File Transfer messages” on page 37
- **z/OS** “New and changed z/OS (CSQ) messages” on page 37

AIX **Linux** **Windows**

New and changed AMQ messages

The following AMQ messages are new for Version 9.0.5:

AMQ3xxx: Utilities and tools

AMQ3726 to AMQ3773

AMQ3932

AMQ3935 to AMQ3943

AMQ6xxx: Common Services

AMQ6325

AMQ6326

AMQ7xxx: IBM MQ product

AMQ7824 to AMQ7826

AMQ8xxx: Administration

AMQ870B

AMQ8D3B to AMQ8D3F

The following AMQ messages are changed for Version 9.0.5:

AMQ3xxx: Utilities and tools

AMQ3726 (Severity changed)

AMQ3824 (key to parameters added)

AMQ3825 (key to parameters added)

AMQ3844 (key to parameters added)

AMQ3894 (key to parameters added)

AMQ5xxx: Installable services

AMQ5019 to AMQ5021 (Severity changed)

AMQ5032 (Severity changed)

AMQ5036 (Severity changed)

AMQ5042 (Severity changed)

AMQ5054 (Severity changed)

AMQ5055 (Severity changed)

AMQ5058 to AMQ5060 (*Bluemix* changed to *IBM Cloud*)

AMQ5061 (Severity changed)

AMQ5657 (Severity changed)

AMQ5659 (Severity changed)

AMQ5975 (Response field added)

AMQ6xxx: Common Services

AMQ6004 (Severity changed)

AMQ6064 (Severity changed)

AMQ6107 (Severity changed)

AMQ6109 (Severity changed)

AMQ6110 (Severity changed)

AMQ6120 to AMQ6122 (Severity changed)

AMQ6125 (Severity changed)

AMQ6177 (Severity changed)

AMQ6187 (Severity changed)

AMQ6257 (Explanation improved)

AMQ6258 (Explanation improved)

AMQ6266 (Severity changed)

AMQ6276 (Severity changed)

AMQ6354 (Severity changed)

AMQ6572 (Severity changed)

AMQ6584 (Severity changed)

AMQ6648 (key to parameters added)

AMQ6686 (Severity changed)

AMQ6999 (Severity changed)

AMQ7xxx: IBM MQ product

AMQ7005 (Severity changed)

AMQ7008 (Severity changed)

AMQ7010 to AMQ7016 (Severity changed)
AMQ7021 (Severity changed)
AMQ7025 (Severity changed)
AMQ7026 (Severity changed)
AMQ7041 (Severity changed)
AMQ7068 (Severity changed)
AMQ7072 (QM.INI changed to qm.ini)
AMQ7073 (Severity changed)
AMQ7077 (Severity changed)
AMQ7078 (Severity changed)
AMQ7091 to AMQ7098 (Severity changed)
AMQ7206 (Severity changed)
AMQ7207 (Severity changed)
AMQ7214 (Severity changed)
AMQ7215 (Severity changed)
AMQ7264 (Severity changed)
AMQ7285 (typo fixed)
AMQ7291 (Severity changed)
AMQ7299 (Severity changed)
AMQ7342 (Severity changed)
AMQ7770 to AMQ7781 (Severity changed)
AMQ7783 to AMQ7799 (Severity changed)
AMQ7819 (Severity changed)

AMQ8xxx: Administration

AMQ8104 (Severity changed)
AMQ8110 (Severity changed)
AMQ8117 (Severity changed)
AMQ8118 (Severity changed)
AMQ8135 (Severity changed)
AMQ8138 (Severity changed)
AMQ8140 (Severity changed)
AMQ8143 (Severity changed)
AMQ8145 to AMQ8148 (Severity changed)
AMQ8150 (Severity changed)
AMQ8151 (Severity changed)
AMQ8154 to AMQ8157 (Severity changed)
AMQ8178 to AMQ8180 (Severity changed)
AMQ8190 to AMQ8196 (Severity changed)

AMQ8202 (Severity changed)
AMQ8204 to AMQ8211 (Severity changed)
AMQ8214 (Severity changed)
AMQ8226 (Severity changed)
AMQ8242 (Severity changed)
AMQ8243 (Severity changed)
AMQ8298 (Severity changed)
AMQ8304 (Severity changed)
AMQ8477 (Severity changed)
AMQ8478 (Severity changed)
AMQ8481 (Severity changed)
AMQ8501 to AMQ8504 (Severity changed)
AMQ8506 (Severity changed)
AMQ8712 (Severity changed)
AMQ8737 (typo fixed)
AMQ8760 to AMQ8769 (Severity changed)
AMQ8923 (Severity changed)
AMQ8927 (Severity changed)

AMQ9xxx: Remote

AMQ9239 (Severity changed)
AMQ9269 (Severity changed)
AMQ9297 (Severity changed)

New, changed and removed REST API messages

The following REST API messages are new for Version 9.0.5:

MQWB00xx: REST API messages MQWB0000 to 0099
MQWB0086E

MQWB04xx: REST API messages MQWB0400 to 0499
MQWB0400E to MQWB0419E

MQWB20xx: REST API messages MQWB2000 to 2099
MQWB2020E

The following REST API messages are changed for Version 9.0.5:

MQWB01xx: REST API messages MQWB0100 to 0199
MQWB0100E (one word removed from the Response)

The following REST API messages are removed for Version 9.0.5:

MQWB01xx: REST API messages MQWB0100 to 0199
MQWB0101E

New and changed Managed File Transfer messages

The following Managed File Transfer messages are new for Version 9.0.5:

BFGCL0001 - BFGCL9999

BFGCL0761E

BFGCL0762E

BFGCL0763E

BFGCL0764I

BFGCL0765I

BFGCL0766I

BFGCL0767I

BFGCL0768

IBFGCL0769I

BFGCL0770W

BFGCL0771W

BFGCL0772I

BFGCL0773I

BFGCL0774E

BFGCL0775E

BFGCL0776E

BFGCL0777E

BFGCL0778E

BFGCL0779E

BFGDB0001 - BFGDB9999

BFGDB0081E

The following Managed File Transfer messages are changed for Version 9.0.5:

BFGCL0001 - BFGCL9999

BFGCL0752E

BFGII0001 - BFGII9999

BFGII0024E (typo fixed)

z/OS

New and changed z/OS (CSQ) messages

The following IBM MQ for z/OS messages are new for Version 9.0.5:

Message manager messages (CSQM...)

CSQM062I

Agent services messages (CSQV...)

CSQV460I

CSQV461D

The following CSQ messages are changed for Version 9.0.5:

Message manager messages (CSQM...)

- CSQM102E (property name changed from CSQWEAK to CSQXWEAK)
- CSQM118I (updated title of a linked topic)
- CSQM119I (updated title of a linked topic)
- CSQM563E (variable formatting correction)
- CSQM572E (variable formatting correction)
- CSQM573E (variable formatting correction)
- CSQM574E (renaming of Bluemix)
- CSQM575E (Version 9.0.4 flag added)

Utilities messages (CSQU...)

- CSQU555E (updated title of a linked topic)
- CSQU581E (updated title of a linked topic)
- CSQU582E (updated title of a linked topic)
- CSQU584E (updated title of a linked topic)

Distributed queuing messages (CSQX...)

- CSQX193I (renaming of Bluemix)
- CSQX194E (renaming of Bluemix and variable formatting correction)
- CSQX198E (renaming of Bluemix and variable formatting correction)
- CSQX199E (renaming of Bluemix and Version 9.0.5 flag)
- CSQX208E (reworking of links to further information)
- CSQX630E (typo fixed)
- CSQX633E (updated title of a linked topic)
- CSQX634E (updated title of a linked topic)
- CSQX637E (updated title of a linked topic)
- CSQX642E (updated title of a linked topic)
- CSQX644E (updated title of a linked topic)
- CSQX674E (property name changed from CSQWEAK to CSQXWEAK)
- CSQX967I (variable formatting correction)

Initialization procedure and general services messages (CSQY...)

- CSQY333E (grammar improved)
- CSQY336E (grammar improved)
- CSQY337E (grammar improved)

Advanced Message Security (CSQ0...)

- CSQ0214E (grammar improved)
- CSQ0215E (grammar improved)
- CSQ0216E (grammar improved)

DB2[®] manager messages (CSQ5...)

- CSQ5004E (updated title of a linked topic)
- CSQ5037I (updated title of a linked topic)

Related information:

IBM MQ AMQ messages

REST API messages

Managed File Transfer diagnostic messages

Messages for IBM MQ for z/OS

What's new and changed in IBM MQ Version 9.0.4

IBM MQ Version 9.0.4, which is a Continuous Delivery (CD) release, delivers a number of new and enhanced features on AIX, Linux, Windows, and z/OS. Version 9.0.4 also provides fixes to earlier CD releases.

Related information:

 [System Requirements for IBM MQ](#)

 [IBM MQ, WebSphere MQ, and MQSeries product readmes web page](#)

What's new in IBM MQ Version 9.0.4

IBM MQ Version 9.0.4 delivers a number of new and enhanced features on AIX, Linux, Windows, and z/OS.


The individual features provided by a CD release are stand-alone functions but may increase in capability in subsequent CD releases, enhancing earlier CD features and forming a suite of related capabilities.

**Additional platform support for Continuous Delivery**

Support for AIX is added for Continuous Delivery from IBM MQ Version 9.0.4.

New IBM MQ for Multiplatforms features

The following new features are available with both IBM MQ base product entitlement, and with IBM MQ Advanced entitlement:

- “Enhancements for the administrative REST API” on page 40
- “New commands for configuring the mqweb server” on page 41
- “The messaging REST API” on page 41
- “Local OS authentication for IBM MQ Console and administrative REST API” on page 41
- “Changes to the dspmqweb command” on page 42
- “migmqlog command” on page 42
-  “Monitoring of IBM MQ for AIX queue managers through IBM Cloud Product Insights” on page 42
- “Support for HTTP proxy access for queue managers that connect to IBM Cloud Product Insights” on page 42
- “Support for creating event messages for Salesforce platform events” on page 42



The following new features are available only with IBM MQ Advanced entitlement:

- ▶ Linux “IBM MQ Bridge to blockchain is available on Linux” on page 43
- ▶ Linux “RDQM high availability” on page 43
- ▶ AIX ▶ Linux ▶ Windows “Ability to run the Managed File Transfer Logger in client mode” on page 43

New IBM MQ for z/OS features

The following new features are available both with IBM MQ for z/OS base product entitlement, and with IBM MQ for z/OS Value Unit Edition (VUE) or IBM MQ Advanced for z/OS, Value Unit Edition entitlement:

- “Enhancements for the administrative REST API”
- “New commands for configuring the mqweb server” on page 41
- “The messaging REST API” on page 41
- “Db2 Universal Table Spaces ” on page 44

The following new feature is available with IBM MQ Managed File Transfer for z/OS, IBM MQ Advanced for z/OS, or IBM MQ Advanced for z/OS, Value Unit Edition entitlement:

- “Ability to run the Managed File Transfer Logger in client mode on z/OS” on page 44

▶ z/OS ▶ MQ Adv. VUE The following new features are available only with IBM MQ Advanced for z/OS, Value Unit Edition entitlement:

- “Using an HTTP proxy to connect to IBM Cloud Product Insights in IBM Cloud” on page 44
- “Java and JMS client connectivity to z/OS queue managers” on page 44

Related concepts:

“IBM MQ license information” on page 4

What you can purchase with IBM MQ and what each purchase entitles you to install.

New features on all supported platforms: ▶ AIX ▶ Linux ▶ Windows ▶ z/OS
▶ V 9.0.4

IBM MQ Version 9.0.4 delivers a number of new features for the REST API on all platforms that are supported by this release, that is on AIX, Linux, Windows, and z/OS.

- “Enhancements for the administrative REST API”
- “New commands for configuring the mqweb server” on page 41
- “The messaging REST API” on page 41

Enhancements for the administrative REST API

From Version 9.0.4, the following enhancements are available for the administrative REST API:

- You can run any MQSC command by using the HTTP POST method with the `/admin/action/qmgr/{qmgrName}/mqsc` resource. For more information, see `/admin/action/qmgr/{qmgrName}/mqsc`.
- You can request information about the attributes of a subscription by using the HTTP GET method with the `/admin/qmgr/{qmgrName}/subscription` resource. For more information, see `/admin/qmgr/{qmgrName}/subscription`.
- You can request information about the attributes, and status, of a channel by using the HTTP GET method with the `/admin/qmgr/{qmgrName}/channel` resource. For more information, see `/admin/qmgr/{qmgrName}/channel`.
- The attribute **enableMediaImageOperations** has been added to the queue REST API. This attribute is in the extended object. For more information, see Request body attributes for queues.

- You can use the administrative REST API to administer queue managers that are not in the same installation as the mqweb server. That is, you can administer remote queue managers and the IBM MQ objects that are associated with the remote queue manager. Therefore, you can use the REST API to administer your entire IBM MQ network with only one installation that runs the mqweb server. For more information, see Remote administration using the REST API.
- CSRF synchronizer tokens for the administrative REST API are periodically regenerated. You can modify configuration of the CSRF protection for the administrative REST API by using the **setmqweb properties** command. For more information, see Configuring CSRF token protection.


New commands for configuring the mqweb server

From Version 9.0.4, you can use the **setmqweb properties** command to configure properties of the mqweb server that is used to support the IBM MQ Console and administrative REST API. For example, you can set the HTTPS host port, and the log file size. You can use the **dspmqweb properties** command to display the properties. For more information about the commands and the properties that you can set, see dspmqweb, and setmqweb.

The messaging REST API

From Version 9.0.4, you can use the messaging REST API to send and receive IBM MQ messages. This feature is intended as a replacement for the IBM MQ bridge for HTTP:




- You can send an IBM MQ text message to a specified queue by using the HTTP POST method with the `/messaging/qmgr/{qmgrName}/queue/{queueName}/message` resource. For more information, see POST.
- You can receive an IBM MQ text message from a specified queue by using the HTTP DELETE method with the `/messaging/qmgr/{qmgrName}/queue/{queueName}/message` resource. For more information, see DELETE.

Note:  The messaging REST API is not supported on the IBM MQ Appliance in Version 9.0.4.

For more information about the messaging REST API, see Messaging using the REST API.

New AIX, Linux and Windows features:

On AIX, Linux and Windows, IBM MQ Version 9.0.4 delivers a number of new features.

- “Local OS authentication for IBM MQ Console and administrative REST API”
-   “Changes to the dspmqweb command” on page 42
- “migmqlog command” on page 42
-  “Monitoring of IBM MQ for AIX queue managers through IBM Cloud Product Insights” on page 42
- “Support for HTTP proxy access for queue managers that connect to IBM Cloud Product Insights” on page 42
- “Support for creating event messages for Salesforce platform events” on page 42

Local OS authentication for IBM MQ Console and administrative REST API

To make use of the IBM MQ Console or administrative REST API, users need to authenticate against a user registry, defined to the mqweb server. From Version 9.0.4, you can use the local OS authentication feature. This feature allows the web server to access the local operating system to validate passwords, identify users and calculate group membership.

For more information, see [Configuring users and roles](#).

Changes to the dspmqweb command

In Version 9.0.3, and earlier, the **dspmqweb** command is used to return the status of the mqweb server, that is, whether the mqweb server is running. From Version 9.0.4, the **dspmqweb** control command can be used to return the status of the mqweb server, or return the configuration properties that are set for the mqweb server.

To return the status of the mqweb server, use the **status** parameter, or issue the command without a parameter. To return the configuration properties, use the **properties** parameter.

For more information, see [dspmqweb](#).

migmqlog command

Version 9.0.4 introduces the **migmqlog** command. This command enables you to migrate logs, either in place, or to a new location. For more information, see [migmqlog](#).



Monitoring of IBM MQ for AIX queue managers through IBM Cloud Product Insights

From Version 9.0.4, you can connect your AIX queue managers to your IBM Cloud Product Insights service instance in IBM Cloud to monitor the startup and usage metrics.

For more information, see [Configuring IBM MQ for use with IBM Cloud Product Insights service in IBM Cloud](#).

Support for HTTP proxy access for queue managers that connect to IBM Cloud Product Insights

From Version 9.0.4, you can connect your queue managers that might not have access to the internet, to IBM Cloud Product Insights through an HTTP proxy that your organization might provide.

For more information, see [Connecting to Product Insights in IBM Cloud through an HTTP proxy](#).

Support for creating event messages for Salesforce platform events

From Version 9.0.4, you can use the IBM MQ Bridge to Salesforce to process a message from an IBM MQ topic and send it to Salesforce as an event message that corresponds to an existing platform event. For more information, see [Creating event messages for Salesforce platform events](#).

New IBM MQ Advanced features on Multiplatforms:

AIX

Linux

Windows

V 9.0.4

MQ Adv.

IBM MQ Version 9.0.4 delivers a number of new features that are available only with IBM MQ Advanced entitlement on Multiplatforms.

- **Linux** “IBM MQ Bridge to blockchain is available on Linux”
- **Linux** “RDQM high availability”
- **AIX** **Linux** **Windows** “Ability to run the Managed File Transfer Logger in client mode”

Linux

IBM MQ Bridge to blockchain is available on Linux

From Version 9.0.4, you can use the IBM MQ Bridge to blockchain to connect your IBM MQ Advanced queue manager to:

- Your local Hyperledger Fabric blockchain network that is running in a Docker container.
- Your blockchain network that is running in a Kubernetes container cluster in IBM Cloud (formerly Bluemix).

You can then use the bridge to send queries, make updates to, and receive responses from your blockchain network. For more information, see [Configuring IBM MQ for use with blockchain](#).

Linux

RDQM high availability

From Version 9.0.4, the replicated data queue manager (RDQM) high availability solution is available for Linux platforms only. Queue managers that use RDQM have messages and logs written to the local storage, and this storage is automatically, synchronously replicated by the IBM MQ system to two other systems. This provides the safety of two failover systems without the need for shared disks or external high-availability management solutions.

For more information, see [RDQM high availability](#).

Ability to run the Managed File Transfer Logger in client mode

From Version 9.0.4 onwards, the Managed File Transfer Logger, configured to use the FILE mode, can connect to the coordination queue manager in client mode.

To run the Managed File Transfer Logger as a client in FILE mode, you must have an IBM MQ Advanced license.

To use the logger as a client, you must use one or more of the additional parameters in the `fteCreateLogger` command. There are, also, extra parameters for use in the `logger.properties` file.

Related concepts:

“IBM MQ license information” on page 4

What you can purchase with IBM MQ and what each purchase entitles you to install.

New z/OS features:

On the z/OS platform, IBM MQ Version 9.0.4 provides additional samples for Db2 V12.

- “Db2 Universal Table Spaces ”

Db2 Universal Table Spaces

Version 9.0.4 provides additional samples, demonstrating how a queue-sharing group can be set up to use Universal Table Spaces (UTS) in Db2 V12.

For more information, see Planning your Db2 environment.

Related information:

z/OS installation overview

New IBM MQ Advanced for z/OS, Value Unit Edition features:

IBM MQ Version 9.0.4 delivers a number of new features that are available with IBM MQ Advanced for z/OS, Value Unit Edition entitlement.

- “Using an HTTP proxy to connect to IBM Cloud Product Insights in IBM Cloud”
- “Java and JMS client connectivity to z/OS queue managers”
- “Ability to run the Managed File Transfer Logger in client mode on z/OS”

Using an HTTP proxy to connect to IBM Cloud Product Insights in IBM Cloud

Version 9.0.4 adds the capability of connecting to Product Insights in IBM Cloud through an HTTP proxy.

Java and JMS client connectivity to z/OS queue managers

From Version 9.0.4, you can connect an IBM MQ classes for JMS, or IBM MQ classes for Java, application to a queue manager on z/OS, that has the **ADVCAP** (ENABLED) attribute, by using a client connection.

For more information, see Java and JMS client connectivity to z/OS queue managers.

Ability to run the Managed File Transfer Logger in client mode on z/OS

From Version 9.0.4, the Managed File Transfer Logger is a separately installable feature.

On z/OS, to run the Managed File Transfer Logger as a client in FILE mode, you must have IBM MQ Managed File Transfer for z/OS, IBM MQ Advanced for z/OS, or IBM MQ Advanced for z/OS, Value Unit Edition entitlement.

To use the logger as a client, you must use one or more of the additional parameters in the `fteCreateLogger` command. There are, also, extra parameters for use in the `logger.properties` file.

Related concepts:

“IBM MQ license information” on page 4

What you can purchase with IBM MQ and what each purchase entitles you to install.

What's changed in Version 9.0.4



Changes to functions and resources in Version 9.0.4 are described in this section. Review these changes before upgrading queue managers to the latest product version and decide whether you must plan to make changes to existing applications, scripts, and procedures before starting to migrate your systems.

For additional information about system requirements for Version 9.0, see the System Requirements for IBM MQ Version 9.0 web page. You can use the Software Product Compatibility Reports (SPCR) tool to find information on supported operating systems, system requirements, prerequisites, and optional supported software. The web page includes more information about the SPCR tool and links to reports for each supported platform.

Changes from earlier versions of IBM MQ












Review this information if you are moving from Long Term Support to Continuous Delivery, or if you are moving to Version 9.0.4 from an earlier Continuous Delivery release.

- “Change to Idle Standby Parts” on page 46
- “Changes to the dmpmqcfg command” on page 47
- “Changes to the strmqm command” on page 47
- “Changes to trace on AIX” on page 47
- “Changes to SYSTEM.DEFAULT.AUTHINFO.IDPWOS and SYSTEM.DEFAULT.AUTHINFO.IDPWLDAP connection authentication information objects” on page 47
- “Changes to qm.ini ChlauthEarlyAdopt attribute for new queue managers” on page 47
- “Changes to client channel authentication for z/OS queue managers” on page 48
- “Changes to user authentication with the Java client and IBM MQ Explorer” on page 48
- “Changes to Bouncy Castle JAR files used by IBM MQ classes for Java and IBM MQ classes for JMS” on page 48
- “Comparison check of transfer identifier and value of groupId attribute restored for message-to-file transfers” on page 48
- “Change to behavior of Managed File Transfer fteDeleteAgent command” on page 48
- “Change to required permissions for Managed File Transfer agent authority queues” on page 49
- “Change to order of authority checks when a Managed File Transfer agent receives a request to cancel a file transfer” on page 49

Changes from earlier CD releases

Review this information if you are moving to Version 9.0.4 from an earlier Continuous Delivery release.

- “Changes to error logging” on page 49









-    “New attribute added to the administrative Queue REST APIs” on page 50
-     “Changes to unit of work identifiers in the response body attributes for queues” on page 50
-     “Changes to the base URL for the administrative REST API” on page 50

Version 9.0.4 changes from earlier versions of IBM MQ:

Changes in Version 9.0.4 to functions and resources from earlier versions of IBM MQ are described in this section.

Review this information if you are moving from Long Term Support to Continuous Delivery, or if you are moving to Version 9.0.4 from an earlier Continuous Delivery release.

For additional information about system requirements for Version 9.0, see the System Requirements for IBM MQ Version 9.0 web page. You can use the Software Product Compatibility Reports (SPCR) tool to find information on supported operating systems, system requirements, prerequisites, and optional supported software. The web page includes more information about the SPCR tool and links to reports for each supported platform.

-    “Change to Idle Standby Parts”
- “Changes to the dmpmqcfg command” on page 47
- “Changes to the strmqm command” on page 47
-  “Changes to trace on AIX” on page 47
- “New environment variable AMQ_SSL_LDAP_SERVER_VERSION” on page 47
- “Changes to SYSTEM.DEFAULT.AUTHINFO.IDPWOS and SYSTEM.DEFAULT.AUTHINFO.IDPWLDAP connection authentication information objects” on page 47
-    “Changes to qm.ini ChlauthEarlyAdopt attribute for new queue managers” on page 47
-  “Changes to client channel authentication for z/OS queue managers” on page 48
- “Changes to user authentication with the Java client and IBM MQ Explorer” on page 48
- “Changes to Bouncy Castle JAR files used by IBM MQ classes for Java and IBM MQ classes for JMS” on page 48
- “Comparison check of transfer identifier and value of groupId attribute restored for message-to-file transfers” on page 48
- “Change to behavior of Managed File Transfer fteDeleteAgent command” on page 48
- “Change to required permissions for Managed File Transfer agent authority queues” on page 49
- “Change to order of authority checks when a Managed File Transfer agent receives a request to cancel a file transfer” on page 49

Change to Idle Standby Parts

From Version 9.0.4, Idle Standby Parts have been renamed to High Availability Replica parts. For more information, see “IBM MQ license information” on page 4.

Changes to the `dmpmqcfg` command

From Version 9.0.4, a command line switch `-w` is added to `dmpmqcfg`. This switch allows you increase the time, in seconds, that `dmpmqcfg` waits for a reply message from the command server.

For more information, see `dmpmqcfg`.

Changes to the `strmqm` command

From Version 9.0.4, the `strmqm` command checks the syntax of the CHANNELS and SSL stanzas in the `qm.ini` file before starting the queue manager fully, which makes it much easier to see what is wrong, and correct it quickly if `strmqm` finds that the `qm.ini` file contains any errors. For more information, see `strmqm`.



Changes to trace on AIX

From Version 9.0.4, enabling AIX system trace is separated from enabling IBM MQ trace.

You must specifically enable AIX system trace. For more information, see `strmqtrc`.

New environment variable `AMQ_SSL_LDAP_SERVER_VERSION`

From Version 9.0.4, you can set the environment variable `AMQ_SSL_LDAP_SERVER_VERSION` to ensure that IBM MQ passes the required lightweight directory access protocol (LDAP) version, that is LDAP v2 or LDAP v3, to GSKit when a certificate is validated against a higher version of LDAP Certificate Revocation List (CRL) server. For more information, see Environment variables.

Changes to `SYSTEM.DEFAULT.AUTHINFO.IDPWOS` and `SYSTEM.DEFAULT.AUTHINFO.IDPWLDAP` connection authentication information objects

In Version 9.0.3, and earlier, the value for the `adopt` context parameter of the system default connection authentication information objects was `N0`. That is, successful user ID and password validation did not change the security context of the connection. From Version 9.0.4, the `adopt` context parameters of the `SYSTEM.DEFAULT.AUTHINFO.IDPWOS` and `SYSTEM.DEFAULT.AUTHINFO.IDPWLDAP` objects is set to `YES` for new queue managers and when re-creating default objects. Queue managers that were created by an earlier version of IBM MQ that are started with Version 9.0.4 retain their existing object definitions.

For more information, see `DEFINE AUTHINFO`.

Changes to `qm.ini` `ChlauthEarlyAdopt` attribute for new queue managers

In Version 9.0.3, and earlier, the `ChlauthEarlyAdopt` attribute could be optionally added to the Channels: stanza of a queue manager's `qm.ini` configuration file. The `ChlauthEarlyAdopt` attribute controls the order in which connection authentication and channel authentication rules are evaluated. From Version 9.0.4, the `ChlauthEarlyAdopt` attribute is automatically added to the Channels: stanza of the `qm.ini` file for new queue managers with a value of `Y` that is connection authentication rules are evaluated before channel authentication rules. Queue managers that were created by an earlier version of IBM MQ that are started with Version 9.0.4 do not have this stanza added and retain any pre-existing `ChlauthEarlyAdopt` attribute values.

For more information, see Attributes of the channels stanza.



Changes to client channel authentication for z/OS queue managers

In Version 9.0.3, and earlier, channel authentication rules are evaluated before connection authentication validation of user ID and password credentials. From Version 9.0.4, channel authentication rules are evaluated after connection authentication validation of credentials, this change in ordering allows channel authentication blocking and mapping rules to consider any changes in the client security context from a successful user ID and password validation. The previous behavior can be restored by issuing the following MQSC command:

```
RECOVER QMGR(TUNE CHLAUTHEARLYADOPT OFF)
```

For more information, see [Securing remote connectivity to the queue manager](#).

Changes to user authentication with the Java client and IBM MQ Explorer

In Version 9.0.3, and earlier, user ID and passwords that are sent from Java client applications to queue managers use compatibility mode by default. From Version 9.0.4, the default setting for Java clients and IBM MQ Explorer is MQCSP authentication mode. Applications that need to use compatibility mode, for example for authenticating connections to queue managers older than Version 8.0.0 must override the authentication mode.

The compatibility mode of operation can be enabled on a connection-by-connection basis or globally:

- In IBM MQ classes for Java, set the property `MQConstants.USE_MQCSP_AUTHENTICATION_PROPERTY` to `false` in the properties hashtable that is passed to the `com.ibm.mq.MQQueueManager` constructor.
- In IBM MQ classes for JMS, set the property `JmsConstants.USER_AUTHENTICATION_MQCSP` to `false`, on the appropriate connection factory before creating the connection.
- Globally, specify the Java system property `-Dcom.ibm.mq.cfg.jmqi.useMQCSPauthentication=false` on the command line when starting your application, as shown in the following example:

```
java -Dcom.ibm.mq.cfg.jmqi.useMQCSPauthentication=false application_name
```

The check box to enable compatibility mode for remote queue manager connections in IBM MQ Explorer is cleared in Version 9.0.4.

For more information, see [Connection authentication with the Java client](#).

Changes to Bouncy Castle JAR files used by IBM MQ classes for Java and IBM MQ classes for JMS

In Version 9.0.4 the Bouncy Castle JAR files that are used to provide CMS support in IBM MQ classes for Java and IBM MQ classes for JMS have been updated to version 1.5.7. For more information about these files, see [Support for non-IBM JREs](#).

Comparison check of transfer identifier and value of groupId attribute restored for message-to-file transfers

From Version 9.0.4, Managed File Transfer is updated to restore the comparison check, previously removed by APAR IT18213, of the transfer identifier and the value of the groupId attribute within the transfer request XML payload. If these two identifiers are equivalent, the source agent uses the identifier as a message identifier match option (as opposed to a group identifier match option) for the first MQGET attempt that is made on the input queue for the message-to-file transfer.

Change to behavior of Managed File Transfer `fteDeleteAgent` command

From Version 9.0.4, the `fteDeleteAgent` command has been updated so that any resource monitor and scheduled transfers are removed when the agent is deleted.

For more information about the **fteDeleteAgent**, see fteDeleteAgent (delete a Managed File Transfer Agent).

Change to required permissions for Managed File Transfer agent authority queues

From Version 9.0.4, when user authority management is enabled by setting the agent property **authorityChecking=true**, inquire is a required permission on all of the agent authority queues.

For more information, see Restricting user authorities on MFT agent actions and The MFT agent.properties file.

Change to order of authority checks when a Managed File Transfer agent receives a request to cancel a file transfer

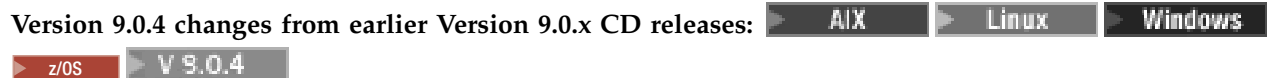
From Version 9.0.4, when user authority management is enabled by setting the agent property **authorityChecking=true**, the order in which authority checks are performed when an agent receives a request to cancel a file transfer is changed. The change to the order of checking avoids unexpected errors in agent and queue manager error logs when the user who requested the file transfer and the user who requested the cancellation are the same.

For more information, see Restricting user authorities on MFT agent actions.

Related concepts:

“What was new and changed in earlier versions” on page 89





Links to information about new features and changes to functions and resources, including stabilizations, deprecations and removals, that occurred in versions of the product before IBM MQ Version 9.0.

Version 9.0.4 changes from earlier Version 9.0.x CD releases: 

Changes to functions and resources introduced in earlier Version 9.0.x CD releases are described in this section.

Review this information if you are moving to Version 9.0.4 from an earlier Continuous Delivery release.

For additional information about system requirements for Version 9.0, see the System Requirements for IBM MQ Version 9.0 web page. You can use the Software Product Compatibility Reports (SPCR) tool to find information on supported operating systems, system requirements, prerequisites, and optional supported software. The web page includes more information about the SPCR tool and links to reports for each supported platform.

-  “Changes to error logging”
-  “New attribute added to the administrative Queue REST APIs” on page 50
-  “Changes to unit of work identifiers in the response body attributes for queues” on page 50
-  “Changes to the base URL for the administrative REST API” on page 50

Changes to error logging

There are various changes to error logging in Version 9.0.4:

- The change in log rollover behavior.
See Using error logs for more information.

- The increased default log size.
From Version 9.0.4, the default log size before rollover has changed from 2 MB to 32 MB.
You need to ensure that you have enough disk space to allow for the increase in size.
- All diagnostic messages, whether to the error logs or to the screen, now have the severity appended.
Attention: You need to modify any parsing tools appropriately.
The **AMQ_DIAGNOSTIC_MSG_SEVERITY** environment variable with a value of 0 is now used to turn off this feature. For more information, see Using error logs.
- New information in messages, including:
 - ArithInsert1
 - ArithInsert2
 - CommentInsert1
 - CommentInsert2
 - CommentInsert3
- The suppression of message amq8245.
Message amq8077 is put out as message amq8245 for display only.

New attribute added to the administrative Queue REST APIs

For Version 9.0.4, the attribute `enableMediaImageOperations` has been added to the Queue REST API. This attribute is in the extended object. For more information, see Request body attributes for queues.

Changes to unit of work identifiers in the response body attributes for queues

From Version 9.0.4, the `unitOfWorkId` and `qmgrTransactionId` attributes in the `applicationHandles` object are represented in hexadecimal. In earlier versions, these attributes are represented in Base64 format. For more information, see Response body attributes for queues.

Changes to the base URL for the administrative REST API

With the introduction of the new messaging REST API in Version 9.0.4, the existing base URLs for the administrative REST API have a new format to distinguish and separate the two:

For Version 9.0.3 and earlier, use the following URL format for the administrative REST API:

```
https://host:port/ibmmq/rest/v1/
```

For Version 9.0.4, and later, use the following URL format for the administrative REST API:

```
https://host:port/ibmmq/rest/v1/admin
```

The login URL stays at :

```
https://host:port/ibmmq/rest/v1/login
```


for Version 9.0.4.

For more information about the new Messaging REST API, see Messaging REST API reference.

New and changed messages in Version 9.0.4



A summary of the new messages that have been added for IBM MQ Version 9.0.4, with links to further information. Any messages that have been changed are also listed.

- “New and changed IBM MQ AMQ messages”
- “New and changed REST API messages” on page 52
- “New and changed IBM MQ Console messages” on page 53
- “New IBM MQ Bridge to blockchain messages” on page 53
- “New IBM MQ Bridge to Salesforce messages” on page 53
- “New and changed Managed File Transfer messages” on page 53
-  “New and changed IBM MQ for z/OS CSQ messages” on page 54



New and changed IBM MQ AMQ messages

The following IBM MQ AMQ messages are new for Version 9.0.4:

AMQ3xxx: Utilities and tools

AMQ3774 to AMQ3899

AMQ3944 to AMQ3960

AMQ4xxx: User interface messages (Windows and Linux systems)

AMQ4638

AMQ4639

AMQ5xxx: Installable services

AMQ5062

AMQ6xxx: Common Services

AMQ6323 (UNIX and Linux)

AMQ6323 (Windows)

AMQ6324

AMQ6610 (IBM MQ Appliance)

AMQ7xxx: IBM MQ product

AMQ7354 to AMQ7361

AMQ7734 to AMQ7737

AMQ8xxx: Administration

AMQ8124

AMQ8125

AMQ8245

AMQ8928

AMQ8929

AMQ9xxx: Remote

AMQ9097

The following AMQ messages are changed for Version 9.0.4:

AMQ5xxx: Installable services

AMQ5059
AMQ5060
AMQ5600 (IBM MQ Appliance)
AMQ5610 (AIX)
AMQ5613 (IBM MQ Appliance)
AMQ5703
AMQ5766

AMQ6xxx: Common Services

AMQ6052
AMQ6231 (HP NSS)
AMQ6509
AMQ6610
AMQ6620
AMQ6675
AMQ6680

AMQ7xxx: IBM MQ product

AMQ7701 to AMQ7704
AMQ7706 to AMQ7709
AMQ7711
AMQ7712
AMQ7714
AMQ7716 to AMQ7718 (IBM i)

AMQ8xxx: Administration

AMQ8121
AMQ8350
AMQ8882
AMQ8885
AMQ8886

New and changed REST API messages

From Version 9.0.4, the numbering pattern for REST API messages has changed. The prefix "REST" becomes "MQWB0", and the message type identifier is added to the end of each message number. For example, REST000 becomes MQWB0000E.

The following REST API messages are new for Version 9.0.4:

MQWB00xx: REST API messages MQWB0000 to 0099

MQWB0075E to MQWB0085E
MQWB0098E

MQWB01xx: REST API messages MQWB0100 to 0199

MQWB0115E to MQWB0118E

MQWB02xx: REST API messages MQWB0200 to 0299
MQWB0201E to MQWB0208E

MQWB03xx: REST API messages MQWB0300 to 0399
MQWB0300E to MQWB0314E

The following REST API messages are changed for Version 9.0.4:

MQWB00xx: REST API messages MQWB0000 to 0099
MQWB0007E
MQWB0026E
MQWB0028E
MQWB0042E
MQWB0046E
MQWB0049E
MQWB0060E
MQWB0070E
MQWB0072E

MQWB01xx: REST API messages MQWB0100 to 0199
MQWB0103E

New and changed IBM MQ Console messages

From Version 9.0.4, the numbering pattern for IBM MQ Console messages has changed. The prefix "WMQWB0" becomes "MQWB2". For example, WMQWB0001 becomes MQWB2001. There are no other changes.

There are no new IBM MQ Console messages for Version 9.0.4.

New IBM MQ Bridge to blockchain messages

The following IBM MQ Bridge to blockchain messages are new for Version 9.0.4:

IBM MQ Bridge to blockchain messages AMQBC000 to 999
AMQBC001 to AMQBC051

New IBM MQ Bridge to Salesforce messages

The following IBM MQ Bridge to Salesforce messages are new for Version 9.0.4:

IBM MQ Bridge to Salesforce messages AMQSF000 to 999
AMQSF001 to AMQSF037

New and changed Managed File Transfer messages

The following Managed File Transfer messages are new for Version 9.0.4:

BFGBR0001 - BFGBR9999
BFGBR0208E

BFGCH0001 - BFGCH9999
BFGCH0127I

BFGDB0001 - BFGDB9999
BFGDB0079E

BFGDB0080E

BFGMQ0001 - BFGMQ9999
BFGMQ1045I

The following Managed File Transfer messages are changed for Version 9.0.4:

BFGAG0001 - BFGAG9999
BFGAG0141E

BFGIO0001 - BFGIO9999
BFGIO0410E

BFGPR0001 - BFGPR9999
BFGPR0139E

BFGSS0001 - BFGSS9999
BFGSS0071E

z/OS

New and changed IBM MQ for z/OS CSQ messages

The following IBM MQ for z/OS CSQ messages are new for Version 9.0.4:

Message manager messages (CSQM...)

CSQM102E

CSQM578I

Distributed queuing messages (CSQX...)

CSQX674E

CSQX690I

CSQX691I

CSQX692I

CSQX693I

The following CSQ messages are changed for Version 9.0.4:

CICS adapter and Bridge messages (CSQC...)

All CSQC messages (CSQC100D to CSQC792I) replaced by a link into the DFHMQnnnn messages section of the CICS documentation.

Recovery log manager messages (CSQJ...)

CSQJ414I

Message manager messages (CSQM...)

CSQM574E

CSQM575E

Utilities messages (CSQU...)

CSQU586I

Advanced Message Security (CSQ0...)

CSQ0608E

CSQ0629E

Related information:



IBM MQ AMQ messages
REST API messages
IBM MQ Console messages
IBM MQ Bridge to blockchain messages
IBM MQ Bridge to Salesforce messages
Managed File Transfer diagnostic messages
Messages for IBM MQ for z/OS

What's new and changed in IBM MQ Version 9.0.3



IBM MQ Version 9.0.3, which is a Continuous Delivery (CD) release, delivers a number of new and enhanced features on Linux, Windows, and z/OS.

Related information:

-  [System Requirements for IBM MQ](#)
-  [IBM MQ, WebSphere MQ, and MQSeries product readmes web page](#)

What's new in IBM MQ Version 9.0.3



IBM MQ Version 9.0.3 delivers a number of new and enhanced features on Linux, Windows, and z/OS.

The individual features provided by a CD release are stand-alone functions but may increase in capability in subsequent CD releases, enhancing earlier CD features and forming a suite of related capabilities.



New support for IBM Cloud Private

- Installing IBM MQ into IBM Cloud Private

New features on all supported platforms

- “Enhancements for the administrative REST API” on page 56
- “Managed File Transfer resource monitor event logging” on page 56

New Linux and Windows only features

- “crtmqdir command” on page 56

New z/OS only features

- “Changes in z/OS product identifier (PID) recording” on page 56



New Connector Pack IBM MQ Advanced for z/OS, Value Unit Edition features

- “Monitoring of IBM MQ Advanced for z/OS, Value Unit Edition queue managers and usage metrics through the IBM Cloud Product Insights service in IBM Cloud (formerly Bluemix)” on page 57
- “Agent remote connections with IBM MQ Advanced for z/OS, Value Unit Edition” on page 57
- “Connectivity to IBM Blockchain service in IBM Cloud for IBM MQ Advanced for z/OS, Value Unit Edition queue managers” on page 57

New features on all supported platforms: 

IBM MQ Version 9.0.3 delivers a number of new features on all platforms that are supported by this release, that is on Linux, Windows, and z/OS.

- “Enhancements for the administrative REST API”
- “Managed File Transfer resource monitor event logging”

Enhancements for the administrative REST API

From Version 9.0.3, the following enhancements are available for the administrative REST API:

- You can request status information about a queue manager by using the **status** query parameter with the **qmgr** resource. For example, you can request information about the date and time at which the queue manager was started. For more information, see `/qmgr GET`.

Managed File Transfer resource monitor event logging

From Version 9.0.3, Managed File Transfer writes information to a log file, that is local to an agent, on what the resource monitors of that agent are doing.

See Overview of resource logging for general information, and the:

- `fteSetAgentLogLevel` command, for a description of the additional **logMonitor** parameter.
- `agent.properties` file for a description of the additional properties.

New Linux and Windows features: 

On Linux and Windows, IBM MQ Version 9.0.3 delivers a new command, `crtmqdir`.

`crtmqdir` command

From Version 9.0.3, the **amqicdir** tool to create or configure the IBM MQ data path has been improved and documented as the **crtmqdir** command.

Note, however, that the **amqicdir** tool is still shipped.

New z/OS features: 

On the z/OS platform, IBM MQ Version 9.0.3 delivers changes in z/OS product identifier (PID) recording.

Changes in z/OS product identifier (PID) recording

Version 9.0.3 introduces an improved method of including the correct PID for each of the following components:

- Base IBM MQ
- Advanced Message Security
- Managed File Transfer
- IBM MQ for z/OS Value Unit Edition (VUE)
- IBM MQ Advanced for z/OS, Value Unit Edition

For more information, see Product usage recording with IBM MQ for z/OS products.

New and enhanced capabilities for IBM MQ Advanced for z/OS, Value Unit Edition:

For IBM MQ Advanced for z/OS, Value Unit Edition, Version 9.0.3 delivers a Connector Pack that includes three new features that provide connectivity for queue managers running on IBM MQ Advanced for z/OS VUE.

- “Monitoring of IBM MQ Advanced for z/OS, Value Unit Edition queue managers and usage metrics through the IBM Cloud Product Insights service in IBM Cloud (formerly Bluemix)”
- “Agent remote connections with IBM MQ Advanced for z/OS, Value Unit Edition”
- “Connectivity to IBM Blockchain service in IBM Cloud for IBM MQ Advanced for z/OS, Value Unit Edition queue managers”

Monitoring of IBM MQ Advanced for z/OS, Value Unit Edition queue managers and usage metrics through the IBM Cloud Product Insights service in IBM Cloud (formerly Bluemix)

From Version 9.0.3, you can connect your existing queue managers to your own IBM Cloud Product Insights service instance in IBM Cloud to monitor the registration and usage metrics.

For more information, see [Configuring IBM MQ Advanced for z/OS, Value Unit Edition for use with IBM Cloud Product Insights service in IBM Cloud](#).

Agent remote connections with IBM MQ Advanced for z/OS, Value Unit Edition

Version 9.0.3 adds the ability for Managed File Transfer agents running on IBM MQ Advanced for z/OS, Value Unit Edition, to connect to a queue manager using a client connection. For more information, see [Enable agent remote connections with IBM MQ Advanced for z/OS, Value Unit Edition](#).

Connectivity to IBM Blockchain service in IBM Cloud for IBM MQ Advanced for z/OS, Value Unit Edition queue managers

From Version 9.0.3 you can use the IBM MQ Bridge to blockchain to send simple queries to and receive responses from your IBM Blockchain service in IBM Cloud. For more information, see [Configuring IBM MQ Advanced for z/OS VUE for use with the IBM Blockchain service in IBM Cloud](#).

Note: After 13 September 2017, the High Security Business Network vNext (Limited Beta) is no longer available. If you previously used the IBM MQ Bridge to blockchain to connect to an instance of the HSBN service in IBM Cloud, you will find that you are no longer able to do this as the limited beta for the service is ended.

What's changed in Version 9.0.3



Changes to functions and resources in IBM MQ Version 9.0.3 are described in this section. Review these changes before upgrading queue managers to the latest product version and decide whether you must plan to make changes to existing applications, scripts, and procedures before starting to migrate your systems.

For additional information about system requirements for Version 9.0, see the System Requirements for IBM MQ Version 9.0 web page. You can use the Software Product Compatibility Reports (SPCR) tool to find information on supported operating systems, system requirements, prerequisites, and optional supported software. The web page includes more information about the SPCR tool and links to reports for each supported platform.

- "Sample configuration file for IBM MQ classes for Java"
- "Changes to the strmqm command"
- "Error log enhancements"
- "Change to behavior of Managed File Transfer fteCleanAgent command"
- "Corrected behavior for messagePublicationFormat=mixed in the Managed File Transfer installation properties" on page 59
- "Enabling MQCSP authentication in Managed File Transfer" on page 59
- "Client reason code" on page 59

Sample configuration file for IBM MQ classes for Java

From Version 9.0.3, a sample configuration file, `mqjava.config`, is supplied in the `bin` subdirectory of the IBM MQ classes for Java installation directory. For more information, see The IBM MQ classes for Java configuration file.

Changes to the `strmqm` command

From Version 9.0.3, the security of `data path/log/qm`, on UNIX, has been changed from 2775 to 2770.

Error log enhancements

From Version 9.0.3, when IBM MQ processes write a message to an error log, the message time in ISO 8601 format, in Coordinated Universal Time (UTC), is included as a `Time()` attribute.

For more information, see Using error logs.

Change to behavior of Managed File Transfer `fteCleanAgent` command

From Version 9.0.3, the `fteCleanAgent` command has been updated so that you must specify which Managed File Transfer state to clear by passing the appropriate parameters to the command, as well as providing an agent name. This change in behavior ensures that, by default, `fteCleanAgent` does not clear all in-progress and pending transfers, resource monitor definitions and scheduled transfer definitions for the agent specified.

If required, you can revert to the previous behavior of `fteCleanAgent` by setting the new `failCleanAgentWithNoArguments` property in the `command.properties` file to `false`.

For more information, see `fteCleanAgent` (cleans up a Managed File Transfer Agent) and `The command.properties` file.

Corrected behavior for `messagePublicationFormat=mixed` in the Managed File Transfer installation properties

The `messagePublicationFormat` property was added to the `installation.properties` file at Version 9.0.2. In that version, when you specified `messagePublicationFormat=mixed`, the effect was the same as specifying `messagePublicationFormat=MQFMT_NONE`. The error is fixed in Version 9.0.3. For more information, see `The installation.properties` file.

Enabling MQCSP authentication in Managed File Transfer

From Version 9.0.3, you can disable the default compatibility mode and enable MQCSP authentication for connection authentication for:

- A Managed File Transfer agent connecting to a coordination queue manager, or command queue manager.
- The IBM MQ Explorer MFT Plugin connecting to a coordination queue manager or command queue manager.

For more information, see `Enabling MQCSP authentication mode`.

Client reason code

From Version 9.0.3, where multiple client connection attempts are made, for example with a channel with a comma separated connection name and the connection sequence fails, reason codes other than `MQRC_HOST_NOT_AVAILABLE` and `MQRC_STANDBY_Q_MGR` are given priority when being returned to an application. In previous versions the last reason code encountered in the connection sequence is always returned, this could lead to configuration errors such as `MQRC_NOT_AUTHORIZED` encountered earlier in the connection attempt to not be returned to the application.

New and changed messages in Version 9.0.3



A summary of the new messages that have been added for IBM MQ Version 9.0.3, with links to further information. Any messages that have been changed are also listed.

- “New and changed IBM MQ AMQ messages”
- “New and changed REST API messages” on page 60
- “New and changed Managed File Transfer messages” on page 61
- “New and changed IBM MQ for z/OS CSQ messages” on page 62



New and changed IBM MQ AMQ messages

The following IBM MQ AMQ messages are new for Version 9.0.3:

AMQ3xxx: Utilities and tools

AMQ3961

AMQ3962

AMQ3963

AMQ3964

AMQ3965

AMQ3966

AMQ3967

AMQ3968

AMQ3969

AMQ6xxx: Common Services

AMQ6321

AMQ6322

The following AMQ messages are changed for Version 9.0.3:

AMQ6xxx: Common Services

AMQ6231

AMQ6241

AMQ6620

AMQ6994

New and changed REST API messages

The following REST API messages are new for Version 9.0.3:

REST API messages

REST068

REST069

REST070

REST071

REST072

REST073

REST074

REST099

REST100

REST101

REST102

REST103

REST104

REST105

REST106

REST107

REST108

REST109

REST110

REST111

REST112

REST113

REST114

REST900

The following REST API messages are changed for Version 9.0.3:

REST API messages

REST003

REST019

REST022

REST035

REST038

REST046

REST049

REST055

REST057

REST059

REST064

New and changed Managed File Transfer messages

The following Managed File Transfer messages are new for Version 9.0.3:

BFGAG0001 - BFGAG9999

BFGAG0187E

BFGCI0001 - BFGCI9999

BFGCI0012E

BFGCI0013E

BFGCI0014E

BFGCI0015E

BFGCI0016E

BFGCL0001 - BFGCL9999

BFGCL0752E

BFGCL0753E

BFGCL0754I

BFGCL0755I

BFGCL0756E

BFGCL0757E

BFGCU0001 - BFGCU9999

BFGCU0056E

BFGMQ0001 - BFGMQ9999

BFGMQ1044E

BFGPR0001 - BFGPR9999

BFGPR0136E

BFGPR0137I
BFGPR0138W
BFGPR0139E
BFGPR0140I

BFGUT0001 - BFGUT9999

BFGUT0032I
BFGUT0033E
BFGUT0034E
BFGUT0035E
BFGUT0036I

The following Managed File Transfer messages are changed for Version 9.0.3:

BFGTR0001 - BFGTR9999

BFGTR0077I



New and changed IBM MQ for z/OS CSQ messages

The following IBM MQ for z/OS CSQ messages are new for Version 9.0.3:

Message manager messages (CSQM...)

CSQM572E
CSQM573E
CSQM574E
CSQM575E
CSQM576E
CSQM577E

Distributed queuing messages (CSQX...)

CSQX193I
CSQX194E
CSQX198E
CSQX967I
CSQX968I

Initialization procedure and general services messages (CSQY...)

CSQY036I
CSQY037I
CSQY038E
CSQY140I
CSQY141I
CSQY142I
CSQY0619I

The following CSQ messages are changed for Version 9.0.3:

Initialization procedure and general services messages (CSQY...) CSQY000I

Related information:

IBM MQ AMQ messages

REST API messages

Managed File Transfer diagnostic messages

Messages for IBM MQ for z/OS

What's new and changed in IBM MQ Version 9.0.2



IBM MQ Version 9.0.2, which is a Continuous Delivery (CD) release, delivers a number of new and enhanced features on Linux, Windows, and z/OS.

Related information:

 Video: IBM MQ V9.0.2 (YouTube)

 System Requirements for IBM MQ

 IBM MQ, WebSphere MQ, and MQSeries product readmes web page

What's new in IBM MQ Version 9.0.2





IBM MQ Version 9.0.2 delivers a number of new and enhanced features on Linux, Windows, and z/OS.

The individual features provided by a CD release are stand-alone functions but may increase in capability in subsequent CD releases, enhancing earlier CD features and forming a suite of related capabilities.

New features on all supported platforms

- “Enhancements for the administrative REST API” on page 64

New Linux and Windows only features

- “Optional installation of the Java Runtime Environment (JRE)” on page 64
- “Automatic management of linear log extents” on page 65
- “Automatic writing of media images” on page 65
- “Monitoring of IBM MQ queue managers and usage metrics through the IBM Cloud Product Insights service in IBM Cloud” on page 65
-  “Integration of Salesforce push topics and platform events with IBM MQ through the IBM MQ Bridge to Salesforce” on page 65
-  “Debian installer for Linux Ubuntu” on page 65

New features on all supported platforms:    

IBM MQ Version 9.0.2 delivers enhancements for the administrative REST API on all platforms that are supported by this release, that is on Linux, Windows, and z/OS.



Enhancements for the administrative REST API

From Version 9.0.2, the following enhancements are available for the administrative REST API:

- Security is enabled for the administrative REST API. You must configure security before you can use the administrative REST API. For more information, see IBM MQ Console and REST API security.
- You can create, update, delete, and view information about queues and queue status by using the REST API. For more information, see `/admin/qmgr/{qmgrName}/queue`.

New Linux and Windows features:   

On Linux and Windows, IBM MQ Version 9.0.2 delivers a number of new features.


- “Optional installation of the Java Runtime Environment (JRE)”
- “Automatic management of linear log extents” on page 65
- “Automatic writing of media images” on page 65
- “Monitoring of IBM MQ queue managers and usage metrics through the IBM Cloud Product Insights service in IBM Cloud” on page 65
-  “Integration of Salesforce push topics and platform events with IBM MQ through the IBM MQ Bridge to Salesforce” on page 65
-  “Debian installer for Linux Ubuntu” on page 65


Optional installation of the Java Runtime Environment (JRE)

From Version 9.0.2, if you are not using Java in your messaging applications, and you are not using portions of IBM MQ that are written in Java, you have the option to not install the JRE (or to remove the JRE if it was already installed).

Attention: If you choose not to install the JRE, or to remove the JRE if it was already installed:

- You must perform your key management by using the `runmqakm` command line tool rather than the `strmqikm` GUI tool, or the `runmqckm` command line tool.
- The IBM Key Management shortcut is still installed. Clicking the shortcut has no effect. You should use the `runmqakm` command line tool instead.
- Use of the `runmqras` command fails unless a JRE at version 7, or later, is available on the system path.

 On Linux, You can install IBM MQ without installing the MQSeriesJRE RPM, unless you are installing the portions of the product that require the presence of the JRE, in which case the RPM prerequisites test fails. From Version 9.0.2, you can now also install the MQSeriesGSKit RPM without the JRE.

 On Windows, before Version 9.0.2, the JRE was a hidden installation feature that was always installed. From Version 9.0.2, the JRE becomes an independently installable feature. The feature can be installed or omitted, either silently or through the GUI installer, as long as you are not installing other features that require the presence of the JRE. For more information, see Install features that require the Server or JRE.

Upgrading from Version 9.0.1 (or before) to Version 9.0.2 (or later) adds the separately installed JRE feature to the installed product.

For more information, see Using `runmqckm`, `runmqakm`, and `strmqikm` to manage digital certificates.

Automatic management of linear log extents

From Version 9.0.2, the queue manager has been extended to support automatic management of linear log extents that are no longer required for recovery. See Managing logs.

To enable this feature for migrated queue managers, you must specify a new attribute (**LogManagement**) either from the Log queue manager properties page of the IBM MQ Explorer, or the Log stanza in the `qm.ini` file. For more information, see Queue manager logs.

If you receive message AMQ7466: There is a problem with the size of the logfile, increase the number of primary log extents.

See Logging and the topics within this section for more information on:

- Types of logging
- Restart recovery
- Calculating the size of the log

Automatic writing of media images

From Version 9.0.2, the queue manager has been extended to support automatic writing of media images.

For more information, see Taking media images automatically.

Monitoring of IBM MQ queue managers and usage metrics through the IBM Cloud Product Insights service in IBM Cloud

From Version 9.0.2, you can connect your existing queue managers to your own IBM Cloud Product Insights service instance in IBM Cloud to monitor the registration and usage metrics.

For more information, see Configuring IBM MQ for use with IBM Cloud Product Insights service in IBM Cloud.



Integration of Salesforce push topics and platform events with IBM MQ through the IBM MQ Bridge to Salesforce

From Version 9.0.2, you can install and use the IBM MQ Bridge to Salesforce to configure the connections to Salesforce and IBM MQ, then run the command `runmqsfb` to subscribe to events from Salesforce and publish them to an IBM MQ network.

For more information, see Configuring IBM MQ for use with Salesforce push topics and platform events and `runmqsfb` (run IBM MQ Bridge to Salesforce).



Debian installer for Linux Ubuntu

From Version 9.0.2, IBM MQ provide packages in Debian installer format (`.deb` files) for installation by using the `dpkg` command, or a higher-level installation tool, on the Linux Ubuntu distribution on all three supported architectures.

For more information, see Installing IBM MQ on Linux Ubuntu using Debian.

What's changed in Version 9.0.2



Changes to functions and resources in IBM MQ Version 9.0.2, including any deprecations, are described in this section. Review these changes before upgrading queue managers to the latest product version and decide whether you must plan to make changes to existing applications, scripts, and procedures before starting to migrate your systems.

For additional information about system requirements for Version 9.0, see the System Requirements for IBM MQ Version 9.0 web page. You can use the Software Product Compatibility Reports (SPCR) tool to find information on supported operating systems, system requirements, prerequisites, and optional supported software. The web page includes more information about the SPCR tool and links to reports for each supported platform.

- “Changes to the administrative REST API /qmgr status attribute and query attribute ”
- “Changes to the administrative REST API security” on page 67
- “The administrative REST API enabled by default” on page 67
- “IBM MQ resource adapter IVT application updated to support WildFly V10 ” on page 67
- “New constant JMS_IBM_SUBSCRIPTION_USER_DATA added to the JmsConstants interface” on page 67
- “Proxy subscriptions are not modified to ADMIN when alterations are attempted” on page 67
- “Improved naming for subscriptions created by the IBM MQ Explorer MFT plugin” on page 67
- “Improved information about the MFT agent when the last reported status is UNKNOWN” on page 67
- “New messagePublicationFormat property for Managed File Transfer installation.properties file” on page 68
- “New agent property adminGroup for use with MFT agents on z/OS” on page 68
- “Change to logger checkpointing” on page 68
- “Change to the dmpmqlog command” on page 68
- “mqconfig change for nproc” on page 68
- “Deprecation: RPM support for Ubuntu” on page 69

Changes to the administrative REST API /qmgr status attribute and query attribute

At Version 9.0.1, the administrative REST API /qmgr object returned a JSON response with an attribute called **status**. The value of this attribute indicated the running state of the queue manager. From Version 9.0.2, this attribute is called **state**. For example, in Version 9.0.1 the JSON response includes "status" : "running". From Version 9.0.2 it includes "state" : "running".


Similarly, at Version 9.0.1 the administrative REST API /qmgr object included an optional query parameter called **status** that filtered the results returned based on the running state of the queue manager. From Version 9.0.2, this query parameter is called **state**.

For more information, see /qmgr GET.

Changes to the administrative REST API security

At Version 9.0.1, security was not available for the administrative REST API. From Version 9.0.2, security is enabled for the administrative REST API. If you used the administrative REST API feature in Version 9.0.1, and the `appSecurity-2.0` feature is enabled in the `mqwebuser.xml` file, at Version 9.0.2, the administrative REST API rejects your request with an HTTP status code of 401. To use the administrative REST API in Version 9.0.2, if security is enabled you must either use token based authentication, or HTTP basic authentication with each REST request. However, if security is not enabled, you do not need to use either approach. For more information, see IBM MQ Console and REST API security.

The administrative REST API enabled by default

At Version 9.0.1, the administrative REST API was disabled by default. From Version 9.0.2, the administrative REST API is enabled by default.  For more information about enabling or disabling the administrative REST API, see [Configuring autostart](#).

IBM MQ resource adapter IVT application updated to support WildFly V10

From Version 9.0.2, the IBM MQ Version 9.0 resource adapter installation verification test application has been updated so that the `WMQ_IVT_MDB.jar` file contains the file `META-INF\jboss-ejb3.xml`, which is used by WildFly V10. The file `WEB-INF\jboss-web.xml` within the `WMQ_IVT.war` file contains the correct resource references for WildFly V10.

For more information, see [Installing and testing the resource adapter in Wildfly](#).

New constant `JMS_IBM_SUBSCRIPTION_USER_DATA` added to the `JmsConstants` interface

From Version 9.0.2, the IBM MQ classes for JMS are updated so that when a message is consumed from a queue that contains an RFH2 header with the MQPS folder, the value associated with the `Sud` key, if it exists, is added as a String property to the JMS Message object returned to the IBM MQ classes for JMS application. To enable an application to retrieve this property from the message, a new constant, `JMS_IBM_SUBSCRIPTION_USER_DATA`, is added to the `JmsConstants` interface. This new property can be used with the method `javax.jms.Message.getStringProperty(java.lang.String)` to retrieve the subscription user data. For more information, see [Retrieval of user subscription data](#).

Proxy subscriptions are not modified to ADMIN when alterations are attempted

Subscriptions with a `SUBTYPE` of `PROXY` cannot be modified. From Version 9.0.2, if an attempt is made to modify a proxy subscription, an error message is reported and the `SUBTYPE` is not modified to `ADMIN`. See `DISPLAY SUB` and `ALTER SUB`.

Improved naming for subscriptions created by the IBM MQ Explorer MFT plugin

At Version 9.0.1, the name of the subscription created on the coordination queue manager by the IBM MQ Explorer MFT plug-in is prefixed with the host name of the system running the associated instance of IBM MQ Explorer. From Version 9.0.2, the prefix is changed to show that the subscription was created by the IBM MQ Explorer MFT plug-in and the name of the user. For more information, see [Ensuring that Managed File Transfer log messages are retained](#).

Improved information about the MFT agent when the last reported status is UNKNOWN

At Version 9.0.2 a new value, **Status Age**, is displayed for agents that are connecting to the coordination manager. The **Status Age** is output when you run `fteListAgents` and `fteShowAgentDetails` commands in the command line. The **Status Age** column can be seen in IBM MQ Explorer for agents that are listed against a coordination manager and in the individual properties for each agent. For more information, see

What to do if the **fteListAgents** command shows an agent status as UNKNOWN.

New messagePublicationFormat property for Managed File Transfer installation.properties file

From Version 9.0.2, a new property, **messagePublicationFormat**, is added to the `installation.properties` file. This property allows you to specify the message publication format used by MFT agents for their status XML messages. For more information, see The `installation.properties` file.

There is a coding error in Version 9.0.2. In this version, when you specify `messagePublicationFormat=mixed`, the effect is the same as specifying `messagePublicationFormat=MQFMT_NONE`. This error is fixed in Version 9.0.3.

z/OS

New agent property adminGroup for use with MFT agents on z/OS

Version 9.0.2 adds a new agent property **adminGroup** for use with Managed File Transfer agents on z/OS. This property defines the name of group of users who can:

- Start the agent by using the **fteStartAgent** command.
- Stop the agent by using the **fteStopAgent** command.
- Enable or disable trace for the agent by using the **fteSetAgentTraceLevel** command.
- Display agent details by using the **fteShowAgentDetails** command.

For more information, see The `agent.properties` file.

Change to logger checkpointing

From Version 9.0.2, the logger schedules checkpoints more frequently (so the next one is scheduled before the previous one has completed) because the logger is attempting to keep the active log in the primary log extents.

Change to the dmpmqlog command

The **dmpmqlog** command outputs a timestamp with each log record. The following text show an example of a timestamp:

```
LOG RECORD - LSN <0:0:4615:42406>
*****
```

```
HLG Header: lreclsize 212, version 1, rmid 0, eyecatcher HLRH
Creation Time: 2017-01-30 13:50:31.146 GMT Standard Time (UTC +0)
```

Linux

mqconfig change for nproc

On Linux, each thread is implemented as a light-weight process (LWP) and each LWP is counted as one process against the resource limit `nproc`. Therefore `nproc` needs to be set based on the number of threads.

From Version 9.0.2, the **mqconfig** script has been modified to display the number of processes based on number of threads running instead of the number of processes.

For more information about `mqconfig` and `nproc`, see [Configuring and tuning the operating system on Linux and mqconfig](#).

Deprecation: RPM support for Ubuntu

RPM support for Ubuntu is deprecated from Version 9.0.2. IBM intends to remove RPM support for Ubuntu in the next Long Term Support version of IBM MQ, and for versions of Ubuntu later than 16.04.

New, changed and removed messages in Version 9.0.2



A summary of the new messages that have been added for IBM MQ Version 9.0.2, with links to further information. Any messages that have been changed or removed are also listed.

- “New, changed and removed IBM MQ AMQ messages”
- “New and changed Managed File Transfer messages” on page 71



New, changed and removed IBM MQ AMQ messages

The following IBM MQ AMQ messages are new for Version 9.0.2:

AMQ3xxx: Utilities and tools

AMQ3970

AMQ3971

AMQ3972

AMQ3973

AMQ3974

AMQ3975

AMQ5xxx: Installable services

AMQ5058

AMQ5059

AMQ5060

AMQ5061_unix

AMQ5766

AMQ6xxx: Common services

AMQ6721

AMQ6723

AMQ6723_ibm

AMQ6762

AMQ6783

AMQ6784

AMQ7xxx: IBM MQ product

AMQ7489

AMQ7490

AMQ7491

AMQ8xxx: Administration

AMQ8599

AMQ8922

AMQ8923
AMQ8923_ibm
AMQ8924
AMQ8924_ibm
AMQ8925
AMQ8926
AMQ8927

AMQ9xxx: Remote

AMQ9183
AMQ9703

The following AMQ messages are changed for Version 9.0.2:

AMQ3xxx: Utilities and tools

AMQ3973 (changed from error to information)
AMQ3975 (changed from error to information)
AMQ3982 (changed from error to information)
AMQ3985 (changed from error to information)
AMQ3988 (changed from error to information)
AMQ3993

 **AMQ4xxx: User interface messages (Windows and Linux systems)**

AMQ4501
AMQ4635

AMQ5xxx: Installable services

AMQ5058
AMQ5059
AMQ5060
AMQ5600
AMQ5600_unix
AMQ5600_windows
AMQ5703

AMQ6xxx: Common Services

AMQ6271
AMQ6623
AMQ6641
AMQ6642
AMQ6693

AMQ8xxx: Administration

AMQ8350
AMQ8351
AMQ8449 (changed from severe error to information)

AMQ8576_windows

AMQ8602

AMQ8617

AMQ9xxx: Remote

AMQ9557

AMQ9616

The following AMQ messages are removed for Version 9.0.2:

AMQ6xxx: Common Services

AMQ6599

New and changed Managed File Transfer messages

The following Managed File Transfer messages are new for Version 9.0.2:

BFGAG0001 - BFGAG9999

BFGAG0184W

BFGAG0185W

BFGAG0186W

BFGIO0001 - BFGIO9999

BFGIO0410E

BFGJE0001 - BFGJE9999

BFGJE0006W

BFGPC0001 - BFGPC9999

BFGPC0053E

The following Managed File Transfer messages are changed for Version 9.0.2:

BFGAG0001 - BFGAG9999

BFGAG0114I

BFGJE0001 - BFGJE9999

BFGJE0001W

BFGTR0001 - BFGTR9999

BFGTR0081I

BFGTR0082I

Related information:

IBM MQ AMQ messages

Managed File Transfer diagnostic messages

What's new and changed in IBM MQ Version 9.0.1



IBM MQ Version 9.0.1, which is a Continuous Delivery (CD) release, delivers a number of new and enhanced features on Linux, Windows, and z/OS.


For all supported platforms, the new function includes updates to IBM MQ that deliver enhanced tooling with a new IBM MQ Console browser-based tool, and a new REST API, for administration.

New features for Linux and Windows also include functional and packaging changes for Managed File Transfer.

On z/OS, the new function also includes the provision of an IBM MQ Service Provider in support of z/OS Connect, and additional support for messaging within CICS Liberty Java virtual machine (JVM) Server.

Related information:

 [Video: IBM MQ V9.0.1 \(YouTube\)](#)

 [System Requirements for IBM MQ](#)

 [IBM MQ, WebSphere MQ, and MQSeries product readmes web page](#)

What's new in Version 9.0.1



IBM MQ Version 9.0.1 delivers a number of new and enhanced features on Linux, Windows, and z/OS.

The individual features provided by a CD release are stand-alone functions but may increase in capability in subsequent CD releases, enhancing earlier CD features and forming a suite of related capabilities.

New features on all supported platforms

- “The IBM MQ Console” on page 73
- “The administrative REST API” on page 73
- “IBM MQ Console and administrative REST API installation component” on page 73
- “Support for class name whitelisting in JMS ObjectMessage” on page 73

New Linux and Windows only features

- Windows C++ client libraries built by using the Microsoft Visual Studio 2015 compiler
- Managed File Transfer configurable recovery timeout for stalled transfers
- Managed File Transfer Agent redistributable package

New z/OS features

- IBM MQ for z/OS service provider for z/OS Connect
- “IBM MQ for z/OS Unix System Services Web Components feature ” on page 75
- “Additional support for messaging within CICS Liberty JVM server” on page 75
- “Display queue manager utility” on page 75

New features on all supported platforms:



IBM MQ Version 9.0.1 delivers a number of new features on all platforms that are supported by this release, that is on Linux, Windows, and z/OS.

- “The IBM MQ Console”
- “The administrative REST API”
- “IBM MQ Console and administrative REST API installation component”
- “Support for class name whitelisting in JMS ObjectMessage”

The IBM MQ Console

From Version 9.0.1, you can use a web-based user interface to administer IBM MQ. The IBM MQ Console runs in a browser and provides control over queue managers and IBM MQ objects.

 To get started quickly with the IBM MQ Console, see [Getting started with the IBM MQ Console](#).

For more information about the IBM MQ Console, see [Administration using the IBM MQ Console](#).

The administrative REST API

From Version 9.0.1, you can use the administrative REST API to request information about queue managers and installations.

To get started quickly with the administrative REST API, see [Getting started with the administrative REST API](#).

For more information about the administrative REST API, see [Administration using the REST API](#).

IBM MQ Console and administrative REST API installation component

If you want to use the IBM MQ Console, or the administrative REST API, you must install the web server that these features run on.

To install this function on Windows, you must install the Web Administration feature. For more information about installing this component on Windows, see [Windows installation tasks](#).

To install this function on Linux, you must install the REST API and Console component. For more information about installing this component on Linux, see [Linux installation tasks](#).

To install this function on z/OS, you must install the IBM MQ for z/OS Unix System Services Web Components feature. For more information about installing this feature, see [Create and customize the IBM WebSphere Application Server Liberty Profile server definition](#) and [Create a procedure for the IBM WebSphere Application Server Liberty Profile server](#).

Support for class name whitelisting in JMS ObjectMessage

From Version 9.0.1, IBM MQ classes for JMS supports whitelisting of classes in the implementation of the JMS ObjectMessage interface. The whitelist defines which Java classes might be serialized with `ObjectMessage.setObject()` and deserialized with `ObjectMessage.getObject()`.

For more information, see [Class name whitelisting in JMS ObjectMessage](#) and [Running IBM MQ classes for JMS applications under the Java Security Manager](#).

New Linux and Windows features:

Linux

Windows

V 9.0.1

On Linux and Windows, IBM MQ Version 9.0.1 delivers a number of new features.

- **Windows** C++ client libraries built by using the Microsoft Visual Studio 2015 compiler
- **Linux** **Windows** Managed File Transfer configurable recovery timeout for stalled transfers
- **Linux** **Windows** “Redistributable Managed File Transfer Agent package”

C++ client libraries built by using the Microsoft Visual Studio 2015 compiler

Windows From Version 9.0.1, IBM MQ provides C++ client libraries that are built with the Microsoft Visual Studio 2015 C++ compiler. Applications that are built by using a release of IBM MQ Version 9.0.1 or later can use these libraries. These libraries are provided in addition to the existing IBM MQ Version 9.0.1 C++ libraries that are built with the Microsoft Visual Studio 2012 C++ compiler.

In order to use the Microsoft Visual Studio 2015 C++ compiler, you must install the Microsoft Knowledge Base update KB3118401, if you are using a version of Windows before Windows 10.

For more information, see Building C++ programs on Windows.

Managed File Transfer configurable recovery timeout for stalled transfers

Linux **Windows** From Version 9.0.1, Managed File Transfer provides a new advanced property for setting a value for the amount of time a source agent continues to try to recover a stalled transfer.

For more information, see Timeout option for transfers in recovery.

Redistributable Managed File Transfer Agent package

Linux **Windows** From Version 9.0.1, Managed File Transfer Agent is optionally provided as an individual redistributable component, available to be downloaded as a zip package. This option enables developers to download, configure and test a Managed File Transfer Agent, to ensure it connects to an existing Managed File Transfer configuration and then make the configured agent bundle available to many users within their organization. Users who are unfamiliar with how Managed File Transfer works, can easily set up the pre-configured agent on their local environment and quickly connect to the relevant IBM MQ network. The users do not have to install IBM MQ to be able to transfer files.

For more information, see Configuring the Redistributable Managed File Transfer Agent.

New z/OS features:

On the z/OS platform, IBM MQ Version 9.0.1 delivers a number of new features.

- “IBM MQ for z/OS Service Provider for z/OS Connect”
- “IBM MQ for z/OS Unix System Services Web Components feature ”
- “Additional support for messaging within CICS Liberty JVM server”
- “Display queue manager utility”

IBM MQ for z/OS Service Provider for z/OS Connect

From Version 9.0.1, IBM MQ for z/OS Service Provider for z/OS Connect (MQ Service Provider) processes requests arriving through z/OS Connect. The MQ Service Provider allows REST aware applications to interact with z/OS assets, that are exposed using IBM MQ for z/OS queues and topics. You can achieve this without having to worry about the coding required to use asynchronous messaging.

For more information, see [IBM MQ for z/OS service provider for z/OS Connect](#).

IBM MQ for z/OS Unix System Services Web Components feature

From Version 9.0.1, if you want to use the IBM MQ Console, or the administrative REST API to IBM MQ for z/OS you need to install this feature.

Note that to install this feature, you must install the IBM MQ for z/OS Unix System Services Components feature.

See [Create and customize the IBM WebSphere Application Server Liberty Profile server definition and Create a procedure for the IBM WebSphere Application Server Liberty Profile server](#) for information on how you set up the feature.

Additional support for messaging within CICS Liberty JVM server

Version 9.0.1 provides an updated IBM MQ resource adapter, which has support to allow connections from the CICS JVM server Liberty profile to IBM MQ, using either client or bindings mode. This additional support for messaging that the resource adapter now provides has a dependency on the CICS Liberty JVM server profile environment. For more information, see [Using IBM MQ classes for JMS in a CICS Liberty JVM server](#).

Display queue manager utility

Version 9.0.1 provides the CSQUDSPM utility, which displays information about queue managers, and provides the equivalent function to **dspm** on Multiplatforms.

What's changed in Version 9.0.1



Changes to functions and resources in IBM MQ Version 9.0.1 are described in this section. Review these changes before upgrading queue managers to the latest product version and decide whether you must plan to make changes to existing applications, scripts, and procedures before starting to migrate your systems.

For additional information about system requirements for Version 9.0, see the System Requirements for IBM MQ Version 9.0 web page. You can use the Software Product Compatibility Reports (SPCR) tool to find information on supported operating systems, system requirements, prerequisites, and optional supported software. The web page includes more information about the SPCR tool and links to reports for each supported platform.

- “Changes to install feature names on Windows”
- “MQPROMPT environment variable for runmqsc”
- “Restriction on the use of topic alias queues in distribution lists”
- “Deprecated CipherSpecs”
- “Changes to DISPLAY CHSTATUS output values for remote Managed File Transfer Agents” on page 77
- “Changes to message to file transfers in Managed File Transfer” on page 77
- “New MFT agent property additionalWildcardSandboxChecking ” on page 77
- “Improvements to z/OSMF” on page 77
- “Migration changes” on page 78



Changes to install feature names on Windows

From Version 9.0.1, there are changes to the names of some of the installable features on Windows:

- The Java and .NET Messaging and Web Services feature is now named Extended Messaging APIs.
- The Windows Client is now named MQI Client.

MQPROMPT environment variable for runmqsc

From Version 9.0.1, you can make it easier to see that you are in an MQSC environment and see some details of the current environment by setting a prompt of your choice by using the MQPROMPT environment variable. For more information, see Administration using MQSC commands.







Restriction on the use of topic alias queues in distribution lists

Distribution lists do not support the use of alias queues that point to topic objects. From Version 9.0.1, if an alias queue points to a topic object in a distribution list, IBM MQ returns MQRC_ALIAS_BASE_Q_TYPE_ERROR.

Deprecated CipherSpecs

From Version 9.0.1, the following CipherSpecs are deprecated:

- FIPS_WITH_3DES_EDE_CBC_SHA
- TRIPLE_DES_SHA_US

- TLS_RSA_WITH_3DES_EDE_CBC_SHA
-    ECDHE_ECDSA_3DES_EDE_CBC_SHA256
-    ECDHE_RSA_3DES_EDE_CBC_SHA256

For more information, see [Deprecated cipherspecs](#).



Changes to **DISPLAY CHSTATUS** output values for remote Managed File Transfer Agents

From Version 9.0.1, output from the **DISPLAY CHSTATUS** command includes values that make it clear when the application that is running at the remote end of a channel is a Managed File Transfer Agent:

- The **RAPPLTAG** channel attribute value that displays the remote partner application name is updated to display *MFT Agent and agent name*.
- A new **RPRODUCT** product identifier *MQJF* is added to indicate that the product running at the remote end of the channel is a Managed File Transfer Agent.

For more information, see [Display CHSTATUS](#).



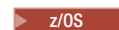
Changes to message to file transfers in Managed File Transfer

In a message to file transfer, the source agent now browses the messages from the source queue, unlike the destructive GET, in IBM MQ versions before Version 9.0.1. The messages are removed from the source queue after all messages (in a group if message grouping is used) have been browsed and data written to the destination file. This allows messages to remain in the source queue if a transfer fails or is canceled. Due to this change, authority to BROWSE must also be provided along with GET authority to run message to file transfers.

For more information about the message-to-file feature, see [Transferring data from messages to files](#).

New MFT agent property **additionalWildcardSandboxChecking**

From Version 9.0.1, if an agent has been configured with a user or agent sandbox in order to restrict the locations that the agent can transfer files to and from, you can specify that additional checks are to be made on wildcard transfers for that agent by setting the **additionalWildcardSandboxChecking** property to true. For more information, see [Additional checks for wildcard transfers](#) and [The agent.properties file](#).



Improvements to z/OSMF

From Version 9.0.1, the IBM z/OS Management Facility (z/OSMF) workflows for provisioning IBM MQ resources have been updated to take advantage of newer Cloud Provisioning capabilities that are being made available in z/OSMF V2.2.

Additionally, the workflow to provision local queues has been enhanced to include actions to display selected attributes of a queue, and to browse or destructively get a message from a queue.

For more information on the improvements to z/OSMF, see [Using IBM z/OSMF to automate IBM MQ](#).



Migration changes


On z/OS you can reverse queue manager migration, as long as you have not enabled new function, and your enterprise is using the Long Term Support (LTS) release model. For more information, see IBM MQ release types.

You enable new function for IBM MQ Version 9.0.0 LTS release, by setting the **OPMODE** parameter to (NEWFUNC, 900). For more information about the changes, see OPMODE.

New and changed messages in Version 9.0.1



A summary of the new messages that have been added for IBM MQ Version 9.0.1, with links to further information. Any messages that have been changed are also listed.

- “New and changed IBM MQ AMQ messages”
- “New and changed Managed File Transfer messages”
-  “New IBM MQ for z/OS CSQ messages” on page 80



New and changed IBM MQ AMQ messages

The following IBM MQ AMQ messages are new for Version 9.0.1:

AMQ3xxx: Utilities and tools
AMQ3976-99

AMQ6xxx: Common Services
AMQ6685

AMQ9xxx: Remote
AMQ9849

The following AMQ messages are changed for Version 9.0.1:

AMQ5xxx: Installable services
AMQ5600

AMQ6xxx: Common Services
AMQ6529,30,32,33,37,38
AMQ6620,23,42,48
AMQ6657-60
AMQ6662,63,66,68,72,74,75,93

AMQ8xxx: Administration
AMQ8151
AMQ8249

AMQ9xxx: Remote
AMQ9060,61,63
AMQ9785

New and changed Managed File Transfer messages

The following Managed File Transfer messages are new for Version 9.0.1:

BFGAG0001 - BFGAG9999

BFGAG0183I

BFGAI0001 - BFGAI9999

BFGAI0041E

BFGAP0001 - BFGAP9999

BFGAP0064E

BFGAP0065E

BFGCH0001 - BFGCH9999

BFGCH0122E

BFGCH0123E

BFGCH0124E

BFGCH0125E

BFGCH0126E

BFGAP0001 - BFGAP9999

BFGAP0064E

BFGAP0065E

BFGCL0001 - BFGCL9999

BFGCL0750E

BFGCL0751E

BFGIO0001 - BFGIO9999

BFGIO0410E

BFGNV0001 - BFGNV9999

BFGNV0175W

BFGNV0176I

BFGSS0001 - BFGSS9999

BFGSS0077E

BFGSS0078E

BFGSS0079E

BFGSS0080E

BFGSS0081E

BFGTR0001 - BFGTR9999

BFGTR0081I

BFGTR0082I

The following Managed File Transfer messages are changed for Version 9.0.1:

BFGCL0001 - BFGCL9999

BFGCL0561E

BFGTR0001 - BFGTR9999

BFGTR0062I

BFGTR0063I

BFGTR0068I

New IBM MQ for z/OS CSQ messages

The following IBM MQ for z/OS CSQ messages are new for Version 9.0.1:

- CSQZ - IBM MQ for z/OS service provider for z/OS Connect

Related information:

IBM MQ AMQ messages

Managed File Transfer diagnostic messages

Messages for IBM MQ for z/OS

What's changed in Version 9.0.0.x Long Term Support

LTS

The Long Term Support (LTS) release is a recommended product level for which support, including defect and security updates, are provided over a specified period of time. This version is intended for systems which demand a long term deployment and maximum stability.

Long term support for the IBM MQ Version 9.0.0 level of function is provided as PTFs for z/OS, or rolled up into fix packs on Multiplatforms. These Long Term Support (LTS) fixes contain only defect fixes and security updates, and are made available at regular intervals.

Long Term Support releases do not deliver new functional enhancements. These releases are intended for systems that demand maximum stability over a long term deployment period.

For further information see IBM MQ release types and IBM MQ FAQ for Long Term Support and Continuous Delivery releases.

V 9.0.0.3

Version 9.0.0, Fix Pack 3

- "Removal of JSON4J.jar file and com.ibm.msg.client.mqlight package" on page 82
- "Additional permission for java.security.policy file" on page 82
- "New attribute to allow TLS v1.0 to be optionally disabled on a queue manager" on page 82
- "Change to required permissions for Managed File Transfer agent authority queues" on page 82
- "Change to order of authority checks when a Managed File Transfer agent receives a request to cancel a file transfer" on page 83

V 9.0.0.2

Version 9.0.0, Fix Pack 2

- "New constant JMS_IBM_SUBSCRIPTION_USER_DATA added to the JmsConstants interface" on page 83
- "New environment variable AMQ_SSL_LDAP_SERVER_VERSION" on page 84
- "Sample configuration file for IBM MQ classes for Java" on page 84
- "Changes to the dmpmqcfg command" on page 84
- "Changes to the strmqm command" on page 84
- "Change to behavior of Managed File Transfer fteCleanAgent command" on page 84
- "Enabling MQCSP authentication in Managed File Transfer" on page 84
- "New messagePublicationFormat property for Managed File Transfer installation.properties file" on page 85
- "Comparison check of transfer identifier and value of groupId attribute restored for message-to-file transfers" on page 85

V 9.0.0.1

Version 9.0.0, Fix Pack 1

- “JMS exception listener updates” on page 85
- “Support for class name whitelisting in JMS ObjectMessage” on page 86
- “IBM MQ resource adapter IVT application updated to support WildFly V10 ” on page 86
- “Proxy subscriptions are not modified to ADMIN when alterations are attempted” on page 86
- “Restriction on the use of topic alias queues in distribution lists” on page 86
- “Deprecated CipherSpecs” on page 86
- “GSKit version updated” on page 87
- “Adopting other security contexts when you use the ADOPTCTX(YES) parameter” on page 87
- **Linux** “mqconfig change for nproc” on page 87
- “MQPROMPT environment variable for runmqsc” on page 87
- “fteMigrateAgent command changes” on page 88
- “New MFT agent property addCommandPathToSandbox” on page 88
- “New MFT agent property additionalWildcardSandboxChecking” on page 88
- **z/OS** “New agent property adminGroup for use with MFT agents on z/OS” on page 88

Related concepts:

V 9.0.0

“What's new and changed in IBM MQ Version 9.0.0” on page 10

IBM MQ Version 9.0 introduces a new continuous delivery and support model. The initial release under this new model delivers a number of new and enhanced capabilities.

CD

“What's new and changed in Version 9.0.x Continuous Delivery” on page 25

Following the initial release of IBM MQ Version 9.0, new function and enhancements are made available by incremental updates within the same version and release as modification releases, for example Version 9.0.3.

“What was new and changed in earlier versions” on page 89

Links to information about new features and changes to functions and resources, including stabilizations, deprecations and removals, that occurred in versions of the product before IBM MQ Version 9.0.





LTS

CD

“Release and platform icons in the product documentation” on page 94

In the IBM MQ Version 9.0 product documentation, icons are used to identify release at which a feature is made available. Icons are also used to identify information that applies to a specific platform or group of platforms only.

Related information:

-  [System Requirements for IBM MQ](#)
-  [IBM MQ, WebSphere MQ, and MQSeries product readmes web page](#)
-  [Recommended Fixes for WebSphere MQ](#)
-  [IBM MQ planned maintenance release dates](#)

What's changed in Version 9.0.0, Fix Pack 3

V 9.0.0.3

IBM MQ Version 9.0.0, Fix Pack 3 includes a number of changes to functions and resources.

Note: Fix Packs are delivered for Multiplatforms only. They are not delivered for the z/OS platform, or for the IBM MQ Appliance.

- “Removal of JSON4J.jar file and com.ibm.msg.client.mqlight package” on page 82
- “Additional permission for java.security.policy file” on page 82
- “New attribute to allow TLS v1.0 to be optionally disabled on a queue manager” on page 82

- “Change to required permissions for Managed File Transfer agent authority queues”
- “Change to order of authority checks when a Managed File Transfer agent receives a request to cancel a file transfer” on page 83

Removal of JS0N4J.jar file and com.ibm.msg.client.mqlight package

The JS0N4J.jar file and com.ibm.msg.client.mqlight package are not needed by the IBM MQ classes for Java and IBM MQ classes for JMS, therefore the following changes are made from Version 9.0.0, Fix Pack 3:

- The JS0N4J.jar file is removed from the *V.R.M.F-WS-MQ-Install-Java-All.jar* file, where *V.R.M.F* is the product version number, for example 9.0.0.3.
- The reference to JS0N4J.jar file is removed from the class path statement within the manifest file for the com.ibm.mq.allclient.jar file.
- The package com.ibm.msg.client.mqlight is no longer included inside the com.ibm.mq.allclient.jar file.

See Installing the IBM MQ classes for JMS separately, What is installed for IBM MQ classes for JMS, and What is installed for IBM MQ classes for Java.

Additional permission for java.security.policy file

From Version 9.0.0, Fix Pack 3, if your Java application uses the Java Security Manager, you must add a RuntimePermission to the java.security.policy file, otherwise, exceptions will be thrown to the application. This RuntimePermission is required by the client as part of managing the assignment and closure of multiplexed conversations over TCP/IP connections to queue managers.

For more information, see Running IBM MQ classes for Java applications under the Java Security Manager.

Change to required permissions for Managed File Transfer agent authority queues

From Version 9.0.0, Fix Pack 3, when user authority management is enabled by setting the agent property **authorityChecking=true**, inquire is a required permission on all of the agent authority queues.

For more information, see Restricting user authorities on MFT agent actions and The MFT agent.properties file.

New attribute to allow TLS v1.0 to be optionally disabled on a queue manager

From Version 9.0.0, Fix Pack 3, a new attribute is available in the qm.ini file, under the SSL stanza:

```
SSL:
  AllowTLSV1=NO
```

If this attribute is set in the qm.ini file before the queue manager is started, the queue manager does not accept inbound connections using the TLS v1.0 protocol. Similarly, if an LDAP connection is configured using an AUTHINFO object, only TLS 1.2 is used to communicate with the LDAP server if secure communication is enabled for the AUTHINFO object.

Alternatively, the **AMQ_TLS_V1_DISABLE** environment variable can be set for the environment used to start the queue manager, listener, and channel processes.

If either property is set, as well as disallowing TLS 1.0 connection attempts at the network layer, the queue manager's command server also rejects attempts to define or alter a channel definition to use a TLS 1.0 CipherSpec.

The default queue manager behavior is unchanged, such that TLS 1.0 connections continue to be accepted if the new attribute or environment variable is not set.

Change to order of authority checks when a Managed File Transfer agent receives a request to cancel a file transfer

From Version 9.0.0, Fix Pack 3, when user authority management is enabled by setting the agent property **authorityChecking=true**, the order in which authority checks are performed when an agent receives a request to cancel a file transfer is changed. The change to the order of checking avoids unexpected errors in agent and queue manager error logs when the user who requested the file transfer and the user who requested the cancellation are the same.

For more information, see Restricting user authorities on MFT agent actions.

Related information:

 System Requirements for IBM MQ

 IBM MQ, WebSphere MQ, and MQSeries product readmes web page

What's changed in Version 9.0.0, Fix Pack 2

V 9.0.0.2

IBM MQ Version 9.0.0, Fix Pack 2 includes a number of changes to functions and resources.

Note: Fix Packs are delivered for Multiplatforms only. They are not delivered for the z/OS platform, or for the IBM MQ Appliance.

- “New constant `JMS_IBM_SUBSCRIPTION_USER_DATA` added to the `JmsConstants` interface”
- “New environment variable `AMQ_SSL_LDAP_SERVER_VERSION`” on page 84
- “Sample configuration file for IBM MQ classes for Java” on page 84
- “Changes to the `dmpmqcfg` command” on page 84
- “Changes to the `strmqm` command” on page 84
- “Change to behavior of Managed File Transfer `fteCleanAgent` command” on page 84
- “Enabling MQCSP authentication in Managed File Transfer” on page 84
- “New `messagePublicationFormat` property for Managed File Transfer `installation.properties` file” on page 85
- “Comparison check of transfer identifier and value of `groupId` attribute restored for message-to-file transfers” on page 85

New constant `JMS_IBM_SUBSCRIPTION_USER_DATA` added to the `JmsConstants` interface

From Version 9.0.0, Fix Pack 2, the IBM MQ classes for JMS are updated so that when a message is consumed from a queue that contains an RFH2 header with the MQPS folder, the value associated with the `Sud` key, if it exists, is added as a `String` property to the JMS Message object returned to the IBM MQ classes for JMS application. To enable an application to retrieve this property from the message, a new constant, `JMS_IBM_SUBSCRIPTION_USER_DATA`, is added to the `JmsConstants` interface. This new property can be used with the method `javax.jms.Message.getStringProperty(java.lang.String)` to retrieve the subscription user data. For more information, see Retrieval of user subscription data and `DEFINE SUB`.

New environment variable **AMQ_SSL_LDAP_SERVER_VERSION**

From Version 9.0.0, Fix Pack 2, you can set the environment variable **AMQ_SSL_LDAP_SERVER_VERSION** to ensure that IBM MQ passes the required lightweight directory access protocol (LDAP) version, that is LDAP v2 or LDAP v3, to GSKit when a certificate is validated against a higher version of LDAP Certificate Revocation List (CRL) server. For more information, see Environment variables.

Sample configuration file for IBM MQ classes for Java

From Version 9.0.0, Fix Pack 2, a sample configuration file, `mqjava.config`, is supplied in the `bin` subdirectory of the IBM MQ classes for Java installation directory. For more information, see The IBM MQ classes for Java configuration file.

Changes to the **dmpmqcfg** command

From Version 9.0.0, Fix Pack 2, a command line switch **-w** is added to **dmpmqcfg**. This switch allows you increase the time, in seconds, that **dmpmqcfg** waits for a reply message from the command server. For more information, see `dmpmqcfg`.

Changes to the **strmqm** command

From Version 9.0.0, Fix Pack 2, the **strmqm** command checks the syntax of the CHANNELS and SSL stanzas in the `qm.ini` file before starting the queue manager fully, which makes it much easier to see what is wrong, and correct it quickly if **strmqm** finds that the `qm.ini` file contains any errors. For more information, see `strmqm`.

Change to behavior of Managed File Transfer **fteCleanAgent** command

From Version 9.0.0, Fix Pack 2, the **fteCleanAgent** command has been updated so that you must specify which Managed File Transfer state to clear by passing the appropriate parameters to the command, as well as providing an agent name. This change in behavior ensures that, by default, **fteCleanAgent** does not clear all in-progress and pending transfers, resource monitor definitions and scheduled transfer definitions for the agent specified.

If required, you can revert to the previous behavior of **fteCleanAgent** by setting the new `failCleanAgentWithNoArguments` property in the `command.properties` file to `false`.

For more information, see `fteCleanAgent` (cleans up a Managed File Transfer Agent) and The `command.properties` file.

Enabling MQCSP authentication in Managed File Transfer

From Version 9.0.0, Fix Pack 2, you can disable the default compatibility mode and enable MQCSP authentication for connection authentication for:

- A Managed File Transfer agent connecting to a coordination queue manager, or command queue manager.
- The IBM MQ Explorer MFT Plugin connecting to a coordination queue manager or command queue manager.

For more information, see Enabling MQCSP authentication mode.

New messagePublicationFormat property for Managed File Transfer installation.properties file

From Version 9.0.0, Fix Pack 2, a new property, **messagePublicationFormat**, is added to the `installation.properties` file. This property allows you to specify the message publication format used by MFT agents for their status XML messages. For more information, see `The installation.properties` file.

Comparison check of transfer identifier and value of groupId attribute restored for message-to-file transfers

From Version 9.0.0, Fix Pack 2, Managed File Transfer is updated to restore the comparison check, previously removed by APAR IT18213, of the transfer identifier and the value of the `groupId` attribute within the transfer request XML payload. If these two identifiers are equivalent, the source agent uses the identifier as a message identifier match option (as opposed to a group identifier match option) for the first MQGET attempt that is made on the input queue for the message-to-file transfer. See also `Transferring data from messages to files`.

Related information:

 [System Requirements for IBM MQ](#)



 [IBM MQ, WebSphere MQ, and MQSeries product readmes web page](#)

What's changed in Version 9.0.0, Fix Pack 1

V9.0.0.1

IBM MQ Version 9.0.0, Fix Pack 1 includes a number of changes to functions and resources.

Note: Fix Packs are delivered for Multiplatforms only. They are not delivered for the z/OS platform, or for the IBM MQ Appliance.

- "JMS exception listener updates"
- "Support for class name whitelisting in JMS ObjectMessage" on page 86
- "IBM MQ resource adapter IVT application updated to support WildFly V10 " on page 86
- "Proxy subscriptions are not modified to ADMIN when alterations are attempted" on page 86
- "Restriction on the use of topic alias queues in distribution lists" on page 86
- "Deprecated CipherSpecs" on page 86
- "GSKit version updated" on page 87
- "Adopting other security contexts when you use the ADOPTCTX(YES) parameter" on page 87
-  "mqconfig change for nproc" on page 87
- "MQPROMPT environment variable for runmqsc" on page 87
- "fteMigrateAgent command changes" on page 88
- "New MFT agent property addCommandPathToSandbox" on page 88
- "New MFT agent property additionalWildcardSandboxChecking" on page 88
-  "New agent property adminGroup for use with MFT agents on z/OS" on page 88

JMS exception listener updates

From Version 9.0.0, Fix Pack 1 IBM MQ classes for JMS are updated so that:

- An `ExceptionListener` registered by an application is invoked for any connection broken exceptions, regardless of whether the application is using synchronous or asynchronous message consumers.
- An `ExceptionListener` registered by an application is invoked if a TCP/IP socket used by a JMS Session is broken.
- Non-connection broken exceptions (for example `MQRC_GET_INHIBITED`) that arise during message delivery are delivered to an application's `ExceptionListener` when the application is using asynchronous

message consumers and the JMS ConnectionFactory used by the application has the ASYNC_EXCEPTIONS property set to the value ASYNC_EXCEPTIONS_ALL.

Note: An ExceptionListener is only invoked once for a connection broken exception, even if two TCP/IP connections (one used by a JMS Connection and one used by a JMS Session) are broken.

For more information, see Exceptions in IBM MQ classes for JMS.

Support for class name whitelisting in JMS ObjectMessage

With APAR IT14385, and from Version 9.0.0, Fix Pack 1, IBM MQ classes for JMS supports whitelisting of classes in the implementation of the JMS ObjectMessage interface. The whitelist defines which Java classes might be serialized with ObjectMessage.setObject() and deserialized with ObjectMessage.getObject().

For more information, see Class name whitelisting in JMS ObjectMessage and Running IBM MQ classes for JMS applications under the Java Security Manager.

IBM MQ resource adapter IVT application updated to support WildFly V10

The IBM MQ Version 9.0 Long Term Support release resource adapter installation verification test application has been updated so that the WMQ_IVT_MDB.jar file contains the file META-INF\jboss-ejb3.xml, which is used by WildFly V10. The file WEB-INF\jboss-web.xml within the WMQ_IVT.war file contains the correct resource references for WildFly V10.

For more information, see Installing and testing the resource adapter in Wildfly.

Proxy subscriptions are not modified to ADMIN when alterations are attempted













Subscriptions with a SUBTYPE of PROXY cannot be modified. From Version 9.0.0, Fix Pack 1, if an attempt is made to modify a proxy subscription, an error message is reported and the SUBTYPE is not modified to ADMIN. See DISPLAY SUB and ALTER SUB.







Restriction on the use of topic alias queues in distribution lists

Distribution lists do not support the use of alias queues that point to topic objects. From Version 9.0.0, Fix Pack 1, if an alias queue points to a topic object in a distribution list, IBM MQ returns MQRC_ALIAS_BASE_Q_TYPE_ERROR.

Deprecated CipherSpecs

From Version 9.0.0, Fix Pack 1, the following CipherSpecs are deprecated:

-    FIPS_WITH_3DES_EDE_CBC_SHA
- NULL_MD5
- NULL_SHA
- TRIPLE_DES_SHA_US
- TLS_RSA_WITH_NULL_MD5
- TLS_RSA_WITH_NULL_SHA
-    ECDHE_ECDSA_NULL_SHA256
-    ECDHE_RSA_NULL_SHA256
-    TLS_RSA_WITH_NULL_NULL
- TLS_RSA_WITH_NULL_SHA256

- TLS_RSA_WITH_3DES_EDE_CBC_SHA
-    ECDHE_ECDSA_3DES_EDE_CBC_SHA256
-    ECDHE_RSA_3DES_EDE_CBC_SHA256

For more information, see [Deprecated cipherspecs](#).

GSKit version updated

The GSKit version has been updated in Version 9.0.0, Fix Pack 1. The new version of GSKit alters the stash file format that is used when you generate an `.sth` file to stash the key database password. Stash files that are generated with this version of GSKit are not readable by earlier versions of GSKit.

To ensure that stash files that are generated with Version 9.0.0, Fix Pack 1, or later, are compatible with your applications and other IBM MQ installations, you must update to a version of IBM MQ that contains a compatible version of GSKit. The following fix packs contain a compatible version of GSKit:

- v7.1.0.8
- v7.5.0.8
- v8.0.0.6
- v9.0.0.1

If you cannot update your applications or other IBM MQ installations, you can request a stash file format that is compatible with an earlier version. When you use the `runmqakm` or `runmqckm` commands with the `-stash` or `-stashpw` option, include the `-v1stash` command line parameter. You cannot use the iKeyman GUI to generate a stash file that is compatible with an earlier version.

Adopting other security contexts when you use the `ADOPTCTX(YES)` parameter

When you use the `ADOPTCTX(YES)` parameter on an authentication information object, another security context cannot be adopted unless you set the `Ch1authEarlyAdopt` parameter in the channels stanza of the `qm.ini` file.

For more information, see [Attributes of the channels stanza](#).



mqconfig change for nproc

On Linux, each thread is implemented as a light-weight process (LWP) and each LWP is counted as one process against the resource limit `nproc`. Therefore `nproc` needs to be set based on the number of threads.

From Version 9.0.0, Fix Pack 1, the `mqconfig` script has been modified to display the number of processes based on number of threads running instead of the number of processes.

For more information about `mqconfig` and `nproc`, see [Configuring and tuning the operating system on Linux and mqconfig](#).

MQPROMPT environment variable for runmqsc

From Version 9.0.0, Fix Pack 1, you can make it easier to see that you are in an MQSC environment and see some details of the current environment by setting a prompt of your choice by using the `MQPROMPT` environment variable. For more information, see [Administration using MQSC commands](#).

fteMigrateAgent command changes

From Version 9.0.0, Fix Pack 1 the **fteMigrateAgent** command has been updated to ensure that the check to see if the user is an administrator is traced.

z/OS The command has also been updated to check that user satisfies (at least) one of these conditions in order to run the migrate command on z/OS:

- Be a member of the mqm group (if the mqm group exists).
- Be a member of the group named in the BFG_GROUP_NAME environment variable (if one is named).
- Have no value set in the BFG_GROUP_NAME environment variable.

For more information about the **fteMigrateAgent** command, see `fteMigrateAgent`.

New MFT agent property addCommandPathToSandbox

A new agent property **addCommandPathToSandbox** has been added to the Managed File Transfer component. This property is used to determine whether the directories specified by the `commandPath` property (and all of their subdirectories) should be added to the denied paths for both user sandboxes and the agent sandbox.

For more information, see `The commandPath` property and `The agent.properties` file.

New MFT agent property additionalWildcardSandboxChecking

From Version 9.0.0, Fix Pack 1, if an agent has been configured with a user or agent sandbox in order to restrict the locations that the agent can transfer files to and from, you can specify that additional checks are to be made on wildcard transfers for that agent by setting the `additionalWildcardSandboxChecking` property to true. For more information, see `Additional checks for wildcard transfers` and `The agent.properties` file.

z/OS

New agent property adminGroup for use with MFT agents on z/OS

Version 9.0.0, Fix Pack 1 adds a new agent property `adminGroup` for use with Managed File Transfer agents on z/OS. This property defines the name of group of users who can:

- Start the agent by using the **fteStartAgent** command.
- Stop the agent by using the **fteStopAgent** command.
- Enable or disable trace for the agent by using the **fteSetAgentTraceLevel** command.
- Display agent details by using the **fteShowAgentDetails** command.

For more information, see `The agent.properties` file.

Related information:

 System Requirements for IBM MQ

 IBM MQ, WebSphere MQ, and MQSeries product readmes web page

What was new and changed in earlier versions

Links to information about new features and changes to functions and resources, including stabilizations, deprecations and removals, that occurred in versions of the product before IBM MQ Version 9.0.

For information about what was new and what changed in an earlier version of the product, see the appropriate section in the product documentation for that version.

IBM MQ Version 8.0

- What's new in IBM MQ Version 8.0
- What's changed in IBM MQ Version 8.0
- What's new and changed in IBM MQ Version 8.0 Fix Packs

IBM WebSphere MQ Version 7.5

- What's new in IBM WebSphere MQ Version 7.5
- What's changed in IBM WebSphere MQ Version 7.5
- What's changed in IBM WebSphere MQ Version 7.5 Fix Packs


IBM WebSphere MQ Version 7.1



- What's new in IBM WebSphere MQ Version 7.1
- What's changed in IBM WebSphere MQ Version 7.1
- What's changed in IBM WebSphere MQ Version 7.1 Fix Packs

IBM WebSphere MQ Version 7.0.1

- What's new in IBM WebSphere MQ Version 7.0.1
- What's changed in IBM WebSphere MQ Version 7.0.1

Attention:

 On IBM MQ for Multiplatforms, you cannot reverse queue manager migration to remove the effect of changes. This restriction applies whether your enterprise uses the Long Term Support (LTS) release or Continuous Delivery (CD) release model.

  On IBM MQ for z/OS, you can reverse queue manager migration as long as you have not enabled new function, and are using the LTS release model. You enable new function, for a queue manager on the LTS release model, by setting the **OPMODE** parameter to (NEWFUNC, 900).

See IBM MQ release types for further information.

Related concepts:

V 9.0.0 “What's new and changed in IBM MQ Version 9.0.0” on page 10

IBM MQ Version 9.0 introduces a new continuous delivery and support model. The initial release under this new model delivers a number of new and enhanced capabilities.

CD “What's new and changed in Version 9.0.x Continuous Delivery” on page 25


Following the initial release of IBM MQ Version 9.0, new function and enhancements are made available by incremental updates within the same version and release as modification releases, for example Version 9.0.3.

LTS “What's changed in Version 9.0.0.x Long Term Support” on page 80

The Long Term Support (LTS) release is a recommended product level for which support, including defect and security updates, are provided over a specified period of time. This version is intended for systems which demand a long term deployment and maximum stability.

Related information:

Migration methods and concepts

 System Requirements for IBM MQ Version 9.0

 IBM MQ, WebSphere MQ, and MQSeries product readmes web page

Notices

This information was developed for products and services that are offered in the USA.

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*

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

*Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan*

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: 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 states 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 supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
J46A/G4
555 Bailey Avenue
San Jose, CA 95141-1003
USA

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

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.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

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 the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

Portions of this code are derived from IBM Corp. Sample Programs.

© Copyright IBM Corp. 1993,2017. All rights reserved.

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 www.ibm.com/legal/copytrade.shtml.

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

Java and all Java-based trademarks and logos are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

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

The Oracle Outside In Technology included herein is subject to a restricted use license and can only be used in conjunction with this application.

Intel is a trademark or registered trademark 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.

Terms and conditions for product documentation

Permissions for the use of these publications are granted subject to the following terms and conditions.

Applicability

These terms and conditions are in addition to any terms of use for the IBM website.

Personal use

You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative work of these publications, or any portion thereof, without the express consent of IBM.

Commercial use

You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of IBM.

Rights

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

Accessibility features for IBM MQ

Accessibility features assist users who have a disability, such as restricted mobility or limited vision, to use information technology content successfully.

Accessibility features

IBM MQ includes the following major accessibility features:

- Keyboard-only operation
- Operations that use a screen reader

IBM MQ uses the latest W3C Standard, WAI-ARIA 1.0 (<http://www.w3.org/TR/wai-aria/>), to ensure compliance to US Section 508 (<http://www.access-board.gov/guidelines-and-standards/communications-and-it/about-the-section-508-standards/section-508-standards>), and Web Content Accessibility Guidelines (WCAG) 2.0 (<http://www.w3.org/TR/WCAG20/>). To take advantage of accessibility features, use the latest release of your screen reader in combination with the latest web browser that is supported by this product.

The IBM MQ online product documentation in IBM Knowledge Center is enabled for accessibility. The accessibility features of IBM Knowledge Center are described at <http://www.ibm.com/support/knowledgecenter/about/releasenotes.html>.

Keyboard navigation

This product uses standard navigation keys.

Interface information

The fully accessible way of using IBM MQ is to use the command line interface. For more information about using commands, see [How to use IBM MQ control commands](#) and [Administration using MQSC commands](#).

For Windows, the accessible way to install IBM MQ is by using a non interactive installation. For further information, see [Advanced installation using msiexec](#).

The IBM MQ user interfaces do not have content that flashes 2 - 55 times per second.

The IBM MQ web user interface does not rely on cascading style sheets to render content properly and to provide a usable experience. However, the product documentation does rely on cascading style sheets. IBM MQ provides an equivalent way for low-vision users to use a user's system display settings, including high-contrast mode. You can control font size by using the device or browser settings.

Related accessibility information

In addition to standard IBM help desk and support websites, IBM has established a TTY telephone service for use by deaf or hard of hearing customers to access sales and support services:

TTY service
800-IBM-3383 (800-426-3383)
(within North America)

IBM and accessibility

For more information about the commitment that IBM has to accessibility, see [IBM Accessibility](#) (www.ibm.com/able).

Release and platform icons in the product documentation





In the IBM MQ Version 9.0 product documentation, icons are used to identify release at which a feature is made available. Icons are also used to identify information that applies to a specific platform or group of platforms only.





The product documentation for the IBM MQ Version 9.0 Long Term Support (LTS) release, as well as for IBM MQ Version 9.0 Continuous Delivery (CD) releases is in the same IBM MQ Version 9.0.x section in IBM Knowledge Center.

If you prefer to use the IBM MQ Version 9.0 product documentation offline, you can download it through the links on the IBM MQ documentation library web page, either as downloadable package or as a set of PDF files.

Long Term Support release, Continuous Delivery release, and release number icons

In the IBM MQ Version 9.0 documentation, rectangular icons are used as follows to identify the following information relating to releases:











- Information that applies to the Long Term Support release in general is indicated by a dark blue icon with the text LTS: .
- Information that applies to the Continuous Delivery release in general is indicated by a dark blue icon with the text CD: .

- Information about features that are made available at the initial release of IBM MQ Version 9.0 is indicated by a dark blue icon that contains the initial release number: 
- For Long Term Support (LTS) release content, the icon is dark blue and contains the LTS release and Fix Pack number. For example .
- Information about features that are made available at an IBM MQ Version 9.0.x Continuous Delivery (CD) release is indicated by a light blue icon that contains the CD release number. For example,  or .

Platform icons

Platform icons are used to identify information that relates to a specific platform, or group of platforms.

Table 2. Platform icons

Icon	Platform	
	AIX	
	HP-UX	
	Solaris	
	All UNIX platforms. That is, all platforms listed previously in this table.	
	Linux	
	Windows	
	All UNIX, Linux, and Windows platforms. That is, all platforms listed previously in this table.	
	IBM i	
	Multiplatforms. That is, all platforms except z/OS.	
	z/OS	

Note: For IBM MQ Version 9.0.1, Version 9.0.2, and Version 9.0.3 Continuous Delivery releases, the supported platforms are Linux, Windows, and z/OS.

For IBM MQ Version 9.0.4 and later Continuous Delivery releases, the supported platforms are AIX, Linux, Windows, and z/OS.

Related concepts:

V 9.0.0 “What's new and changed in IBM MQ Version 9.0.0” on page 10

IBM MQ Version 9.0 introduces a new continuous delivery and support model. The initial release under this new model delivers a number of new and enhanced capabilities.

CD “What's new and changed in Version 9.0.x Continuous Delivery” on page 25

Following the initial release of IBM MQ Version 9.0, new function and enhancements are made available by incremental updates within the same version and release as modification releases, for example Version 9.0.3.

LTS “What's changed in Version 9.0.0.x Long Term Support” on page 80

The Long Term Support (LTS) release is a recommended product level for which support, including defect and security updates, are provided over a specified period of time. This version is intended for systems which demand a long term deployment and maximum stability.

“What was new and changed in earlier versions” on page 89

Links to information about new features and changes to functions and resources, including stabilizations, deprecations and removals, that occurred in versions of the product before IBM MQ Version 9.0.

Related information:

[IBM MQ release types](#)

IBM MQ information roadmap

The information roadmap contains links to a variety of IBM MQ resources.

This roadmap brings together information from different sources to help you find out more about a particular area of IBM MQ. Click the links to each section in the roadmap to see what resources are available.

- [Product overview](#)
- [Technical overview](#)
- [Scenarios](#)
- [Planning](#)
- [Migrating and upgrading](#)
- [Installing](#)
- [Security](#)
- [Configuring](#)
- [Administering](#)
- [Developing applications](#)
- [Monitoring and performance](#)
- [Troubleshooting and support](#)
- [Reference](#)

Table 3. IBM MQ information roadmap table







Category	Information resources
About IBM MQ	<p>Overview of the overall purpose, capabilities, and new features of IBM MQ.</p> <p> “About IBM MQ” on page 1 Introductory information to help you get started with IBM MQ Version 9.0.x, including an introduction to the product and an overview of what is new and what is changed for this release.</p> <p>IBM MQ product web page Product web page with links to resources and additional information.</p> <p></p> <p>Video: IBM MQ Version 9.0 overview (YouTube) An introduction to the new features included in IBM MQ Version 9.0.</p> <p></p> <p>Video: IBM MQ Version 9.0.1 (YouTube) An overview of IBM MQ Version 9.0.1 Continuous Delivery release.</p> <p></p> <p>Video: IBM MQ Version 9.0.2 (YouTube) An overview of IBM MQ Version 9.0.2 Continuous Delivery release.</p> <p></p> <p>Video: IBM MQ Advanced (YouTube) Meet Jerry, and see how IBM MQ Advanced helps him..</p> <p>IBM MQ system requirements Web page with links to the system requirements for the different releases of IBM MQ. For IBM MQ Version 8.0 or later, you can use the Software Product Compatibility Reports (SPCR) tool to find information on supported operating systems, system requirements, prerequisites, and optional supported software.</p> <p>“IBM MQ customer-installable Knowledge Center” on page 100 You can download IBM MQ messaging documentation into an offline version of IBM Knowledge Center that you install locally.</p> <p>IBM MQ Version 9.0.x PDF documentation You can download IBM MQ Version 9.0.x product documentation as a series of PDF files.</p>
Technical overview	<p> “IBM MQ Technical overview” on page 147</p> <p>Information to help you to find out about message queuing and other features that IBM MQ provides.</p>

Table 3. IBM MQ information roadmap table (continued)














Category	Information resources
Scenarios	<p>Each scenario takes you through a significant set of tasks, and helps you to configure a major product feature. The scenarios include useful links to other content to help you to gain a better understanding of the area in which you are interested.</p> <ul style="list-style-type: none"> <li data-bbox="367 363 1414 531">   IBM MQ Scenarios The <i>Getting started</i> scenario explains how to get started with IBM MQ. Use this scenario if you have not used IBM MQ before and want to get started quickly. Further scenarios help you to configure or use product features by taking you through the appropriate task steps. <li data-bbox="367 552 1414 678">  Connecting WebSphere Application Server to IBM MQ Contains information that leads you through the key tasks required to connect WebSphere Application Server to IBM MQ in a variety of scenarios. <li data-bbox="367 699 1414 825">  Connecting WebSphere Application Server Liberty profile to IBM MQ Contains information that leads you through the key tasks required to connect WebSphere Application Server Liberty profile to IBM MQ in a variety of scenarios. <li data-bbox="367 846 1414 968">  Connecting IBM MessageSight to IBM MQ and WebSphere Application Server Contains information that leads you through the key tasks required to connect IBM MessageSight to IBM MQ and WebSphere Application Server in a variety of scenarios.
Planning	<ul style="list-style-type: none"> <li data-bbox="367 999 1414 1188">  Planning When planning your IBM MQ environment, consider the support that IBM MQ provides for single and multiple queue manager architectures, and for point-to-point and publish/subscribe messaging styles. Also plan your resource requirements, and your use of logging and backup facilities.
Migrating and upgrading	<ul style="list-style-type: none"> <li data-bbox="367 1220 1414 1440">  Maintaining and migrating Migration is the process of updating queue managers, and other objects, such as applications or administrative procedures. To migrate a queue manager to run on a new level of code, you must first upgrade IBM MQ to install the new code level. When you have verified that the upgrade is successful, migrate the queue manager and all the applications and resources that are associated with it. Before you start this process, create a migration plan, based on the information in this documentation. <li data-bbox="367 1451 1414 1650">  <ul style="list-style-type: none"> <li data-bbox="367 1503 1414 1650">   IBM WebSphere MQ / IBM MQ Migration Guide This guide provides information to help you plan the process of migrating from an older version to a new version of IBM MQ for Multiplatforms. You can either view the guide in your web browser or download it as a PDF file. <li data-bbox="367 1661 1414 1839">  <ul style="list-style-type: none"> <li data-bbox="367 1713 1414 1839">   IBM WebSphere MQ / IBM MQ for z/OS Migration Guide This guide provides information to help you plan the process of migrating from an older version to a new version on z/OS. You can either view the guide in your web browser or download it as a PDF file.

Table 3. IBM MQ information roadmap table (continued)

















Category	Information resources
Installing	<p> Installing and uninstalling Information to help you to prepare for installation, install the product, and verify the installation. There is also information to help you to uninstall the product.</p> <p> Video: Installing and verifying IBM MQ V9.0 (YouTube) Installing and verifying IBM MQ Version 9.0 on Windows</p> <p>  </p> <p>Video: Redistributable Managed File Transfer Agent (YouTube) How to install and configure the Redistributable Managed File Transfer Agent.</p>
Security	<p> Securing Aspects of security to consider in your IBM MQ installation including identification and authentication, authorization, auditing, confidentiality, and data integrity.</p>
Configuring	<p> Configuring Create one or more queue managers on one or more computers, and configure them and their related resources on your development, test, and production systems to process messages that contain your business data.</p> <p></p> <p>Video: IBM MQ to Product Insights Service in IBM Cloud (YouTube) A tutorial showing how to connect an IBM MQ queue manager to the IBM Cloud Product Insights service in IBM Cloud (formerly Bluemix).</p>
Administering	<p> Administering IBM MQ Administer your queue managers and associated resources.</p> <p> </p> <p>Video: MQ Service Provider for z/OS Connect (YouTube) A tutorial, showing how to configure a one way IBM MQ Service in z/OS Connect. A one way service allows an application to send a message to a queue or a topic using an HTTP POST, browse a message on a queue using an HTTP GET or destructively get a message from a queue using an HTTP DELETE.</p>
Developing applications	<p> Developing applications Develop applications to send and receive messages, and to manage your queue managers and related resources. IBM MQ support applications written in procedural languages, and object oriented languages and frameworks.</p>
Monitoring and performance	<p> Monitoring and performance Monitoring information and guidance to help improve the performance of your queue manager network and tuning tips to help improve the performance of your queue manager network.</p>

Table 3. IBM MQ information roadmap table (continued)

Category	Information resources
Troubleshooting and support	 <p>Troubleshooting and support Techniques to help you diagnose and solve problems with your queue manager network or IBM MQ applications.</p> <p>IBM SupportAssistant web page The IBM Support Assistant (ISA) helps you to resolve questions and problems with IBM software products by providing access to support-related information and troubleshooting tools.</p> <p>IBM Support Portal web page IBM Support Portal for IBM MQ.</p>  <p>dw Answers Search for questions tagged with "messaging".</p>
Reference	 <p>Reference Reference information for configuration, administration, developing applications, telemetry, security, monitoring, troubleshooting and support, and diagnostic messages.</p>

IBM MQ customer-installable Knowledge Center

You can download IBM MQ messaging documentation into an offline version of IBM Knowledge Center that you install locally.

Overview

The IBM Knowledge Center for IBM MQ products (MQ KC) has two components:

- The MQ KC framework. This is a locally-installable offline version of IBM Knowledge Center.
- The documentation packages that you install into the framework. These packages contain the same documentation as is published online in IBM Knowledge Center.

You can run the MQ KC on AIX, Windows, or Linux, or under an application server (WebSphere Application Server Liberty profile).

There are two steps to installing the MQ KC:

1. Installing the MQ KC framework.
You download the framework for your platform from Fix Central. To do this, you must have a license for a current IBM MQ product. After your initial installation, you need to do this again only if there is a new version of the framework from IBM and you want to migrate to that version.
2. Installing messaging documentation into the framework.
You get the documentation packages from the IBM MQ documentation download site. You can install documentation for multiple products and versions into the same MQ KC framework. After your initial installation, you need to do this again only if IBM makes updates to the documentation and you want to pick up those updates.

Detailed installation instructions are provided in a readme file at the root of each framework or documentation package that you download.

Links to the available downloads are given in the following sections.

MQ KC framework packages

The MQ KC framework packages are available to licensed users for download from Fix Central. Each package is around 160-230MB, depending on platform. The packages are called `MQ_KC_version_number.v_date_and_time-fix_pack_level.platform`.

Either search Fix Central for all available MQ KC framework packages, or click an individual package from the following list.

Table 4. MQ KC framework packages: Links, file names and file descriptions

Fix name and framework package name	File description
IBM-MQ-KC-AixPPC64.151fp6.1 (which contains MQ_KC_1.5.1.v201711290558-FP6.1.aix.ppc64.tar.gz)	MQ KC framework package for AIX
IBM-MQ-KC-LinuxIA32.151fp6.1 (which contains MQ_KC_1.5.1.v201711290558-FP6.1.linux.i386.tar.gz)	MQ KC framework package for Linux on IBM i
IBM-MQ-KC-LinuxPPC.151fp6.1 (which contains MQ_KC_1.5.1.v201711290558-FP6.1.linux.ppc64.tar.gz)	MQ KC framework package for Linux on LE Power
IBM-MQ-KC-LinuxX64.151fp6.1 (which contains MQ_KC_1.5.1.v201711290558-FP6.1.linux.x86_64.tar.gz)	MQ KC framework package for Linux on x86 64-bit
IBM-MQ-KC-Java.151fp6.1 (which contains MQ_KC_1.5.1.v201711290558-FP6.1.war)	MQ KC framework package for WebSphere Application Server Liberty profile
IBM-MQ-KC-Win32.151fp6.1 (which contains MQ_KC_1.5.1.v201711290558-FP6.1.win32.zip)	MQ KC framework package for Windows 32-bit
IBM-MQ-KC-Win64.151fp6.1 (which contains MQ_KC_1.5.1.v201711290558-FP6.1.win64.zip)	MQ KC framework package for Windows 64-bit

IBM MQ messaging documentation packages

The documentation packages are available for download from the IBM MQ documentation download site.

- mq90_docs_for_MQ_KC.zip
- mq80_docs_for_MQ_KC.zip
- mqfte704_docs_for_MQ_KC.zip (WebSphere MQ File Transfer Edition Version 7.0.4)

IBM MQ Version 9.0.x PDF documentation





You can download the IBM MQ Version 9.0.x product documentation as a series of PDF files.

IBM MQ product documentation PDF files

The following PDF files contain the same information as the HTML version of the IBM MQ Version 9.0.x product documentation published in IBM Knowledge Center.

With the exception of the Quick Start Guide, these PDF files are for IBM MQ Version 9.0.0 Long Term Support release and its maintenance, as well as for IBM MQ Version 9.0.x Continuous Delivery releases. There are separate versions of the Quick Start Guide for IBM MQ Version 9.0.0 Long Term Support release and IBM MQ Version 9.0.x Continuous Delivery releases.

Table 5. Mapping of PDF files to product documentation sections

PDF file name and download link	Product documentation section
 mq90_qsg_en.pdf	IBM MQ Version 9.0 Quick Start Guide (Long Term Support release)
 mq90cdr_qsg_en.pdf	IBM MQ Version 9.0.x Quick Start Guide (Continuous Delivery releases)
mq90.overview.pdf	Product overview Technical overview
mq90.scenarios.pdf	Scenarios
mq90.installconfig.pdf	Planning Installing Migrating and upgrading Configuring
mq90.administer.pdf	Administering Security Monitoring and performance Troubleshooting and support
mq90.develop.pdf	Developing applications
mq90.reference.pdf	Reference
mq90.mft.pdf	Managed File Transfer
 MQ_Migration_Guide.pdf	Migration guide for IBM WebSphere MQ and IBM MQ on Multiplatforms
 WMQ_zOS_Migration.pdf	Migration guide for IBM WebSphere MQ and IBM MQ for z/OS

Note: The PDF files must be in the same folder for links between PDF files to function correctly.

 z/OS

IBM MQ for z/OS Program Directory PDF files

 LTS

The following IBM MQ for z/OS Version 9.0.0 Program Directory documents are available for download as PDF files from the IBM Publications Center:

- Program directory for IBM MQ for z/OS Long Term Support Release V9.0.0 (GI13-3386)
- Program directory for IBM MQ for z/OS Value Unit Edition Long Term Support Release V9.0.0 (GI13-3387)
- Program directory for Advanced Message Security for z/OS Long Term Support Release V9.0.0 (GI13-3388)
- Program directory for Managed File Transfer for z/OS Long Term Support Release V9.0.0 (GI13-3389)
- Program directory for IBM MQ Advanced for z/OS Long Term Support Release V9.0.0 (GI13-3390)

 CD

The following IBM MQ for z/OS Version 9.0.x Program Directory documents are available for download as PDF files from the IBM Publications Center:

- Program directory for IBM MQ for z/OS Continuous Delivery Release V9.0.x (GI13-3391)
- Program directory for IBM MQ for z/OS Value Unit Edition Continuous Delivery Release V9.0.x (GI13-3395)
- Program directory for Advanced Message Security for z/OS Continuous Delivery Release V9.0.x (GI13-3398)
- Program directory for Managed File Transfer for z/OS Continuous Delivery Release V9.0.x (GI13-3392)

- Program directory for IBM MQ Advanced for z/OS Continuous Delivery Release V9.0.x (GI13-3396)
- Program directory for IBM MQ Advanced for z/OS Value Unit Edition Continuous Delivery Release V9.0.x (GI13-3397)

Glossary

This glossary provides terms and definitions for the IBM MQ software and products.

The following cross-references are used in this glossary:

- *See* refers you from a nonpreferred term to the preferred term or from an abbreviation to the spelled-out form.
- *See also* refers you to a related or contrasting term.

For other terms and definitions, see the IBM Terminology website (opens in new window).

“A” “B” on page 106 “C” on page 107 “D” on page 112 “E” on page 114 “F” on page 115 “G” on page 117 “H” on page 117 “I” on page 118 “J” on page 121 “K” on page 121 “L” on page 122 “M” on page 123 “N” on page 127 “O” on page 128 “P” on page 130 “Q” on page 133 “R” on page 134 “S” on page 137 “T” on page 142 “U” on page 144 “V” on page 145 “W” on page 145 “X” on page 145

A

abend reason code

A 4-byte hexadecimal code that uniquely identifies a problem with a program that runs on the z/OS operating system.

abstract class

In object-oriented programming, a class that represents a concept; classes derived from it represent implementations of the concept. An object cannot be constructed from an abstract class; that is, it cannot be instantiated. See also parent class.

Abstract Syntax Notation One (ASN.1)

The international standard for defining the syntax of information data. It defines a number of simple data types and specifies a notation for referencing these types and for specifying values of these types. The ASN.1 notations can be applied whenever it is necessary to define the abstract syntax of information without constraining in any way how the information is encoded for transmission.

access control

In computer security, the process of ensuring that users can access only those resources of a computer system for which they are authorized.

access control list (ACL)

In computer security, a list associated with an object that identifies all the subjects that can access the object and their access rights.

ACL See access control list.

active log

A data set with a fixed size where recovery events are recorded as they occur. When the active log is full, the contents of the active log are copied to the archive log.

active queue manager instance

The instance of a running multi-instance queue manager that is processing requests. There is only one active instance of a multi-instance queue manager.

adapter

An intermediary software component that allows two other software components to communicate with one another.

address space

The range of addresses available to a computer program or process. Address space can refer to physical storage, virtual storage, or both. See also allied address space, buffer pool.

administration bag

In the IBM MQ Administration Interface (MQAI), a type of data bag that is created for administering IBM MQ by implying that it can change the order of data items, create lists, and check selectors within a message.

administrative topic object

An object that allows you to assign specific, non-default attributes to topics.

administrator command

A command used to manage IBM MQ objects, such as queues, processes, and namelists.

Advanced Message Queuing Protocol (AMQP)

An open-source wire protocol that is used to receive, queue, route, and deliver messages.

Advanced Program-to-Program Communication (APPC)

An implementation of the SNA LU 6.2 protocol that allows interconnected systems to communicate and share the processing of programs.

affinity

An association between objects that have some relationship or dependency upon each other.

alert A message or other indication that signals an event or an impending event that meets a set of specified criteria.

alert monitor

In IBM MQ for z/OS, a component of the CICS adapter that handles unscheduled events occurring as a result of connection requests to IBM MQ for z/OS.

alias queue

An IBM MQ object, the name of which is an alias for a base queue or topic that is defined to the local queue manager. When an application or a queue manager uses an alias queue, the alias name is resolved and the requested operation is performed on the associated base object. For further information, see the topic Alias queues.

alias queue object

An IBM MQ object, the name of which is an alias for a base queue defined to the local queue manager. When an application or a queue manager uses an alias queue, the alias name is resolved and the requested operation is performed on the associated base queue.

allied address space

A z/OS address space that is connected to IBM MQ for z/OS.

ally See allied address space.

alternate user authority

The ability of a user ID to supply a different user ID for security checks. When an application opens an IBM MQ object, it can supply a user ID on the MQOPEN, MQPUT1, or MQSUB call that the queue manager uses for authority checks instead of the one associated with the application. For further information, see the topic Alternate user authority.

alternate user security

On z/OS, the authority checks that are performed when an application requests alternate user authority when opening an IBM MQ object.

AMQP

See Advanced Message Queuing Protocol.

AMQP channel

A type of channel that provides a level of support for AMQP 1.0-compliant applications. MQ Light clients or other AMQP 1.0 compatible clients can be connected to an IBM MQ AMQP channel.

APAR See authorized program analysis report.

APF See authorized program facility.

API-crossing exit

A user written program that is similar in concept to an API exit. It is supported only for CICS applications on IBM MQ for z/OS.

API exit

A user-written program that monitors or modifies the function of an MQI call. For each MQI call issued by an application, the API exit is called before the queue manager starts to process the call and again after the queue manager has completed processing the call. The API exit can inspect and modify any of the parameters on the MQI call.

APPC See Advanced Program-to-Program Communication.

application-defined format

Application data in a message for which the user application defines the meaning. See also built-in format.

application environment

The environment that includes the software and the server or network infrastructure that supports it.

application level security

The security services that are started when an application issues an MQI call.

application log

In Windows systems, a log that records significant application events.

application queue

A local queue that is used by applications for messaging, through the Message Queue Interface (MQI). Application queues are often set up as triggered queues.

archive log

A data set on a storage device to which IBM MQ copies the contents of each active log data set when the active log reaches its size limit. See also recovery log.

ARM See automatic restart manager.

ASN.1 See Abstract Syntax Notation One.

asymmetric key cryptography

A system of cryptography that uses two keys: a public key known to everyone and a private key known only to the receiver or sender of the message. See also symmetric key cryptography. For further information, see the topic Cryptography.

asynchronous consumption

A process that uses a set of MQI calls that allow an application to consume messages from a set of queues. Messages are delivered to the application by using a unit of code identified by the application, passing either the message or a token representing the message.

asynchronous messaging

A method of communication between programs in which a program places a message on a message queue, then proceeds with its own processing without waiting for a reply to its message. See also synchronous messaging.

asynchronous put

A put of a message by an application, without waiting for a response from the queue manager.

attribute

1. A characteristic or trait of an entity that describes the entity; for example, the telephone number of an employee is one of the employee attributes. See also entity.
2. In object-oriented programming, a property of an object or class that can be distinguished distinctly from any other properties. Attributes often describe state information.

authentication

A security service that provides proof that a user of a computer system is genuinely who that person claims to be. Common mechanisms for implementing this service are passwords and digital signatures.

authentication information object

An object that provides the definitions needed to check certificate revocation lists (CRLs) using LDAP servers, in support for Secure Sockets Layer (SSL) security.

authority check

See authorization check.

authorization

The process of granting a user, system, or process either complete or restricted access to an object, resource, or function.

authorization check

A security check that is performed when a user or application attempts to access a system resource; for example, when an administrator attempts to issue a command to administer IBM MQ or when an application attempts to connect to a queue manager.

authorization file

A file that provides security definitions for an object, a class of objects, or all classes of objects.

authorization service

In IBM MQ on UNIX and Linux systems and IBM MQ for Windows, a service that provides authority checking of commands and MQI calls for the user identifier associated with the command or call.

authorized program analysis report (APAR)

A request for correction of a defect in a supported release of a program supplied by IBM.

authorized program facility (APF)

In a z/OS environment, a facility that permits the identification of programs that are authorized to use restricted functions.

automatic restart manager (ARM)

A z/OS recovery function that can automatically restart batch jobs and started tasks after they or the system on which they are running end unexpectedly.

B**backout**

An operation that reverses all changes to resources made during the current unit of work. See also commit.

bag See data bag.

bar A z/OS memory limit, which in 64-bit systems is set at 2 GB. The bar separates storage below the 2-gigabyte address from storage above the 2 GB address. The area above the bar is intended for data; no programs run above the bar.

basic mapping support (BMS)

An interface between CICS and application programs that formats input and output display data and routes multiple-page output messages without regard for control characters used by various terminals.

behavior

In object-oriented programming, the functionality embodied within a method.

BMS See basic mapping support.

Booch methodology

An object-oriented methodology that helps users design systems using the object-oriented paradigm.

bootstrap data set (BSDS)

A VSAM data set that contains an inventory of all active and archived log data sets known to IBM MQ for z/OS, and a wrap-around inventory of all recent IBM MQ for z/OS activity. The BSDS is required to restart the IBM MQ for z/OS subsystem.

browse

In message queuing, to copy a message without removing it from the queue. See also get, put.

browse cursor

In message queuing, an indicator used when browsing a queue to identify the message that is next in sequence.

BSDS See bootstrap data set.

buffer pool

An area of memory into which data pages are read and in which they are modified and held during processing. See also address space.

built-in format

Application data in a message for which the queue manager defines the meaning. See also application-defined format.

C

CA See certificate authority.

CAF See client attachment feature.

callback

A message consumer or an event handler routine.

CCDT See client channel definition table.

CCF See channel control function.

CCSID

See coded character set identifier.

CDF See channel definition file.

certificate authority (CA)

A trusted third-party organization or company that issues the digital certificates. The certificate authority typically verifies the identity of the individuals who are granted the unique certificate. See also Secure Sockets Layer. For further information, see the topic Certificate Authorities.

certificate chain

A hierarchy of certificates that are cryptographically related to one another, starting with the personal certificate and ending with root at the top of the chain.

certificate expiration

A digital certificate contains a date range when the certificate is valid. Outside the valid date range, the certificate is said to be "expired".

certificate request (CR)

Synonym for certificate signing request.

certificate revocation list (CRL)

A list of certificates that have been revoked before their scheduled expiration date. Certificate revocation lists are maintained by the certificate authority and used, during a Secure Sockets Layer (SSL) handshake to ensure that the certificates involved have not been revoked. For further information, see the topic Certificate management.

certificate signing request (CSR)

An electronic message that an organization sends to a certificate authority (CA) to obtain a certificate. The request includes a public key and is signed with a private key; the CA returns the certificate after signing with its own private key.

certificate store

The Windows name for a key repository. For further information, see the topic Personal certificate store.

CF See coupling facility.

CFSTRUCT

An IBM MQ object used to describe the queue manager's use of a Coupling Facility list structure

channel

An IBM MQ object that defines a communication link between two queue managers (message channel) or between a client and a queue manager (MQI channel). See also message channel, MQI channel, queue manager.

channel callback

A mechanism that ensures that the channel connection is established to the correct machine. In a channel callback, a sender channel calls back the original requester channel using the sender's definition.

channel control function (CCF)

A program to move messages from a transmission queue to a communication link, and from a communication link to a local queue, together with an operator panel interface to allow the setup and control of channels.

channel definition file (CDF)

A file containing communication channel definitions that associate transmission queues with communication links.

channel event

An event reporting conditions detected during channel operations, such as when a channel instance is started or stopped. Channel events are generated on the queue managers at both ends of the channel.

channel exit program

A user-written program that is called from one of a defined number of places in the processing sequence of a message channel agent (MCA).

channel initiator

A component of IBM MQ distributed queuing that monitors the initiation queue and starts the sender channel when triggering criteria are met.

channel listener

A component of IBM MQ distributed queuing that monitors the network for a startup request and then starts the receiving channel.

checkpoint

A place in a program at which a check is made, or at which a recording of data is made to allow the program to be restarted in case of interruption.

CI See control interval.

CipherSpec

The combination of encryption algorithm and hash function applied to an SSL message after authentication completes.

cipher suite

The combination of authentication, key exchange algorithm, and the Secure Sockets Layer (SSL) cipher specification used for the secure exchange of data.

ciphertext

Data that is encrypted. Ciphertext is unreadable until it is converted into plaintext (decrypted) with a key. See also cleartext.

circular logging

In IBM MQ on UNIX and Linux systems and IBM MQ for Windows, the process of keeping all restart data in a ring of log files. See also linear logging. For further information, see the topic Types of logging.

CL See Command Language.

class In object-oriented design or programming, a model or template that can be used to create objects with a common definition and common properties, operations, and behavior. An object is an instance of a class.

class hierarchy

The relationships between classes that share a single inheritance.

class library

In object-oriented programming, a collection of prewritten classes or coded templates, any of which can be specified and used by a programmer when developing an application.

cleartext

A string of characters sent over a network in readable form. It might be encoded for the purposes of compression, but it can easily be decoded. See also ciphertext.

client A runtime component that provides access to queuing services on a server for local user applications. The queues used by the applications reside on the server. See also IBM MQ fully managed .NET client, IBM MQ Java client, IBM MQ MQI client.

client application

An application, running on a workstation and linked to a client, that gives the application access to queuing services on a server.

client attachment feature (CAF)

An option that supports the attachment of clients to z/OS.

client channel definition table (CCDT)

A file that contains one or more client-connection channel definitions. For further information, see the topic Client channel definition table.

client-connection channel type

The type of MQI channel definition associated with a IBM MQ client. See also server-connection channel type.

CLUSRCVR

See cluster-receiver channel.

CLUSSDR

See cluster-sender channel.

cluster

In IBM MQ, a group of two or more queue managers on one or more computers, providing automatic interconnection, and allowing queues and topics to be advertised among them for load balancing and redundancy.

cluster queue

A local queue that is hosted by a cluster queue manager, and defined as a target for messages being put from an application connected to any queue manager within the cluster. All applications retrieving messages must be locally connected.

cluster queue manager

A queue manager that is a member of a cluster. A queue manager can be a member of more than one cluster.

cluster-receiver channel (CLUSRCVR)

A channel on which a cluster queue manager can receive messages from other queue managers in the cluster, and cluster information from the repository queue managers.

cluster-sender channel (CLUSDR)

A channel on which a cluster queue manager can send messages to other queue managers in the cluster, and cluster information to the repository queue managers.

cluster topic

An administrative topic that is defined on a cluster queue manager and made available to other queue managers in the cluster.

cluster transmission queue

A transmission queue that holds all messages from a queue manager destined for another queue manager that is in the same cluster. The queue is called `SYSTEM.CLUSTER.TRANSMIT.QUEUE`.

CMS key database

A CMS key database is the format of the Database supported by Windows systems, UNIX systems, Linux, and the clients of those platforms. Files ending with ".kdb" are CMS format. The ".kdb" files contain the certificates and the keys.

coded character set identifier (CCSID)

A 16-bit number that includes a specific set of encoding scheme identifiers, character set identifiers, code page identifiers, and other information that uniquely identifies the coded graphic-character representation.

coexistence

The ability of two or more different versions of IBM MQ to function on the same computer.

command

A statement used to initiate an action or start a service. A command consists of the command name abbreviation, and its parameters and flags if applicable.

command bag

In the MQAI, a type of bag that is created for administering IBM MQ objects, but cannot change the order of data items or create lists within a message.

command event

A notification that an MQSC or PCF command has run successfully.

Command Language (CL)

In IBM MQ for IBM i, a language that can be used to issue commands, either at the command line or by writing a CL program.

command prefix (CPF)

1. In IBM MQ for z/OS, a character string that identifies the queue manager to which IBM MQ for z/OS commands are directed, and from which IBM MQ for z/OS operator messages are received.
2. A 1-character to 8-character command identifier. The command prefix distinguishes the command as belonging to an application or subsystem rather than to z/OS.

command server

The IBM MQ component that reads commands from the system-command input queue, verifies them, and passes valid commands to the command processor.

commit

To apply all the changes made during the current unit of recovery (UR) or unit of work (UOW). After the operation is complete, a new UR or UOW can begin.

common name (CN)

The component in a Distinguished Name (DN) attribute of an X.509 certificate that represents the name normally associated with the owner of the certificate. For people, the CN is usually their actual name. For web servers, the CN is the fully qualified host and domain name of the server. For IBM MQ there are no specific requirements on this field, however many administrators use the name of the queue manager. See also "Distinguished Name"

completion code

A return code indicating how a message queue interface (MQI) call has ended.

confidentiality

The security service that protects sensitive information from unauthorized disclosure. Encryption is a common mechanism for implementing this service.

configuration event

Notifications about the attributes of an object. The notifications are generated when the object is created, changed, or deleted and also by explicit requests.

connection affinity

A channel attribute that specifies the client channel definition that client applications use to connect to the queue manager, if multiple connections are available.

connection factory

A set of configuration values that produces connections that enable a Java EE component to access a resource. Connection factories provide on-demand connections from an application to an enterprise information system (EIS) and allow an application server to enroll the EIS in a distributed transaction.

connection handle

The identifier or token by which a program accesses the queue manager to which it is connected.

constructor

In object-oriented programming, a special method used to initialize an object.

consume

To remove a message from a queue and return its contents to the calling application.

consumer

An application that receives and processes messages. See also message consumer.

context security

On z/OS, the authority checks that are performed when an application opens a queue and specifies that it will set the context in messages that it puts on the queue, or pass the context from messages that it has received to messages that it puts on the queue.

control command

In IBM MQ on UNIX and Linux systems and IBM MQ for Windows, a command that can be entered interactively from the operating system command line. Such a command requires only that the IBM MQ product be installed; it does not require a special utility or program to run it.

control interval (CI)

A fixed-length area of direct access storage in which VSAM stores records and creates distributed free space. The control interval is the unit of information that VSAM transmits to or from direct-access storage. A control interval always includes an integral number of physical records.

controlled shutdown

See quiesced shutdown.

correlation identifier

A field in a message that provides a means of identifying related messages. Correlation identifiers are used, for example, to match request messages with their corresponding reply message.

coupling facility (CF)

A special logical partition that provides high-speed caching, list processing, and locking functions in a sysplex.

CPF See command prefix.

CRL See certificate revocation list.

cross-system coupling facility (XCF)

A component of z/OS that provides functions to support cooperation between authorized programs running within a sysplex.

cryptography

A method for protecting information by transforming it (encrypting it) into an unreadable format, called ciphertext. Only users who possess a secret key can decipher (or decrypt) the message into plaintext.

D

DAE See dump analysis and elimination.

daemon

A program that runs unattended to perform continuous or periodic functions, such as network control.

data bag

A container of object properties that the MQAI uses in administering queue managers. There are three types of data bag: user (for user data), administration (for administration with assumed options), and command (for administration with no options assumed).

data-conversion interface (DCI)

The IBM MQ interface to which customer-written or vendor-written programs that convert application data between different machine encodings and CCSIDs must conform.

data-conversion service

A service that converts application data to the character set and encoding that are required by applications on other platforms.

datagram

A form of asynchronous messaging in which an application sends a message, but does not require a response. See also request/reply.

data integrity

The security service that detects whether there has been unauthorized modification of data, or tampering. The service detects only whether data has been modified; it does not restore data to its original state if it has been modified.

data item

In the MQAI, an integer or character-string item that is contained within a data bag. A data item can be either a user item or a system item.

DCE See Distributed Computing Environment.

DCE principal

A user ID that uses the distributed computing environment.

DCI See data-conversion interface.

DCM See Digital Certificate Manager.

dead-letter queue (DLQ)

A queue to which a queue manager or application sends messages that cannot be delivered to their correct destination.

dead-letter queue handler

A utility that monitors a dead-letter queue (DLQ) and processes messages on the queue in accordance with a user-written rules table. A sample dead letter queue handler is provided by IBM MQ.

decryption

The process of decoding data that has been encrypted into a secret format. Decryption requires a secret key or password.

default object

A definition of an object (for example, a queue) with all attributes defined. If a user defines an object but does not specify all possible attributes for that object, the queue manager uses default attributes in place of any that were not specified.

deferred connection

A pending event that is activated when a CICS subsystem tries to connect to IBM MQ for z/OS before it has started.

derivation

In object-oriented programming, the refinement or extension of one class from another.

destination

1. An end point to which messages are sent, such as a queue or topic.
2. In JMS, an object that specifies where and how messages should be sent and received.

Diffie-Hellman key exchange

A public, key-exchange algorithm that is used for securely establishing a shared secret over an insecure channel.

digital certificate

An electronic document used to identify an individual, a system, a server, a company, or some other entity, and to associate a public key with the entity. A digital certificate is issued by a certification authority and is digitally signed by that authority.

Digital Certificate Manager (DCM)

On IBM i systems, the method of managing digital certificates and using them in secure applications on the IBM i server. Digital Certificate Manager requests and processes digital certificates from certification authorities (CAs) or other third-parties.

digital signature

Information that is encrypted with a private key and is appended to a message or object to assure the recipient of the authenticity and integrity of the message or object. The digital signature proves that the message or object was signed by the entity that owns, or has access to, the private key or shared-secret symmetric key.

direct routing

An option for routing publications in a publish/subscribe cluster. With direct routing, every queue manager in the cluster sends publications from any publishing queue manager direct to any other queue manager in the cluster with a matching subscription.

disconnect

To break the connection between an application and a queue manager.

distinguished name (DN)

A set of name-value pairs (such as CN=person name and C=country or region) that uniquely identifies an entity in a digital certificate.

distributed application

In message queuing, a set of application programs that can each be connected to a different queue manager, but that collectively comprise a single application.

Distributed Computing Environment (DCE)

In network computing, a set of services and tools that supports the creation, use, and maintenance of distributed applications across heterogeneous operating systems and networks.

distributed publish/subscribe

Publish/subscribe messaging performed in a multiple queue manager environment.

distributed queue management

In message queuing, the setup and control of message channels between distributed queue managers.

distributed queuing

Sending messages from one queue manager to another. The receiving queue manager could be on the same machine or on a remote one.

distribution list

A list of queues to which a message can be put with a single statement.

DLQ See dead-letter queue.

DN See distinguished name.

dual logging

A method of recording IBM MQ for z/OS activity, where each change is recorded on two data sets, so that if a restart is necessary and one data set is unreadable, the other can be used. See also single logging.

dual mode

See dual logging.

dump analysis and elimination (DAE)

A z/OS service that enables an installation to suppress SVC dumps and ABEND SYSUDUMP dumps that are not needed because they duplicate previously written dumps.

durable subscription

A subscription that is retained when the connection from a subscribing application to the queue manager is closed. When the subscribing application disconnects, the durable subscription remains in place and publications continue to be delivered. When the application reconnects, it can use the same subscription by specifying the unique subscription name. See also nondurable subscription.

dynamic queue

A local queue created when a program opens a model queue object.

E**eavesdropping**

A breach of communication security in which the information remains intact, but its privacy is compromised. See also impersonation, tampering.

Eclipse

An open-source initiative that provides independent software vendors (ISVs) and other tool developers with a standard platform for developing plug-compatible application development tools.

encapsulation

In object-oriented programming, the technique that is used to hide the inherent details of an object, function, or class from client programs.

encryption

In computer security, the process of transforming data into an unintelligible form in such a way that the original data either cannot be obtained or can be obtained only by using a decryption process.

enqueue

To put a message or item in a queue.

entity A user, group, or resource that is defined to a security service, such as RACF®.

environment variable

A variable that specifies how an operating system or another program runs, or the devices that the operating system recognizes.

ESM See external security manager.

ESTAE

See extended specify task abnormal exit.

event data

In an event message, the part of the message data that contains information about the event (such as the queue manager name, and the application that gave rise to the event). See also event header.

event header

In an event message, the part of the message data that identifies the event type of the reason code for the event. See also event data.

event message

A message that contains information (such as the category of event, the name of the application that caused the event, and queue manager statistics) relating to the origin of an instrumentation event in a network of IBM MQ systems.

event queue

The queue onto which the queue manager puts an event message after it detects an event. Each category of event (queue manager, performance, configuration, instrumentation, or channel event) has its own event queue.

Event Viewer

A tool provided by Windows systems to examine and manage log files.

exception listener

An instance of a class that can be registered by an application and for which the onException() method is called to pass a JMS exception to the application asynchronously.

exclusive method

In object-oriented programming, a method that is not intended to exhibit polymorphism; one with specific effect.

extended specify task abnormal exit (ESTAE)

A z/OS macro that provides recovery capability and gives control to the user-specified exit routine for processing, diagnosing an abend, or specifying a retry address.

external security manager (ESM)

A security product that performs security checking on users and resources. RACF is an example of an ESM.

F**failover**

An automatic operation that switches to a redundant or standby system or node in the event of a software, hardware, or network interruption.

FAP See Formats and Protocols.

Federal Information Processing Standard

A standard produced by the National Institute of Standards and Technology when national and international standards are nonexistent or inadequate to satisfy the U.S. government requirements.

FFDC See first-failure data capture.

FFST™ See First Failure Support Technology™.

FFST file

See First Failure Support Technology file.

FIFO See first-in first-out.

FIPS See Federal Information Processing Standard.

first-failure data capture (FFDC)

1. The i5/OS implementation of the FFST architecture providing problem recognition, selective dump of diagnostic data, symptom string generation, and problem log entry.
2. A problem diagnosis aid that identifies errors, gathers and logs information about these errors, and returns control to the affected runtime software.

First Failure Support Technology (FFST)

An IBM architecture that defines a single approach to error detection through defensive programming techniques. These techniques provide proactive (passive until required) problem recognition and a description of diagnostic output required to debug a software problem.

First Failure Support Technology file (FFST file)

A file containing information for use in detecting and diagnosing software problems. In IBM MQ, FFST files have a file type of FDC.

first-in first-out (FIFO)

A queuing technique in which the next item to be retrieved is the item that has been in the queue for the longest time.

forced shutdown

A type of shutdown of the CICS adapter where the adapter immediately disconnects from IBM MQ for z/OS, regardless of the state of any currently active tasks. See also quiesced shutdown.

format

In message queuing, a term used to identify the nature of application data in a message.

Formats and Protocols (FAP)

In message queuing, a definition of how queue managers communicate with each other, and of how clients communicate with server queue managers.

Framework

In IBM MQ, a collection of programming interfaces that allow customers or vendors to write programs that extend or replace certain functions provided in IBM MQ products. The interfaces are the following: data conversion interface (DCI), message channel interface (MCI), name service interface (NSI), security enabling interface (SEI), trigger monitor interface (TMI).

friend class

A class in which all member functions are granted access to the private and protected members of another class. It is named in the declaration of another class and uses the keyword friend as a prefix to the class.

FRR See functional recovery routine.

full repository

A complete set of information about every queue manager in a cluster. This set of information is called the repository or sometimes the full repository and is usually held by two of the queue managers in the cluster. See also partial repository.

function

A named group of statements that can be called and evaluated and can return a value to the calling statement.

functional recovery routine (FRR)

A z/OS recovery and termination manager that enables a recovery routine to gain control in the event of a program interrupt.

G**gateway queue manager**

A cluster queue manager that is used to route messages from an application to other queue managers in the cluster.

generalized trace facility (GTF)

A z/OS service program that records significant system events such as I/O interrupts, SVC interrupts, program interrupts, and external interrupts.

Generic Security Services API

See Generic Security Services application programming interface.

Generic Security Services application programming interface (Generic Security Services API, GSS API)

A common application programming interface (API) for accessing security services.

get

In message queuing, to use the MQGET call to remove a message from a queue and return its contents to the calling application. See also browse, put.

globally defined object

On z/OS, an object whose definition is stored in the shared repository. The object is available to all queue managers in the queue-sharing group. See also locally defined object.

global trace

An IBM MQ for z/OS trace option where the trace data comes from the entire IBM MQ for z/OS subsystem.

global transaction

A recoverable unit of work performed by one or more resource managers in a distributed transaction environment and coordinated by an external transaction manager.

GSS API

See Generic Security Services application programming interface.

GTF See generalized trace facility.**H****handshake**

The exchange of messages at the start of a Secure Sockets Layer session that allows the client to authenticate the server using public key techniques (and, optionally, for the server to authenticate the client) and then allows the client and server to cooperate in creating symmetric keys for encryption, decryption, and detection of tampering.

hardened message

A message that is written to auxiliary (disk) storage so that the message is not lost in the event of a system failure.

header

See message header.

heartbeat

A signal that one entity sends to another to convey that it is still active.

heartbeat flow

A pulse that is passed from a sending message channel agent (MCA) to a receiving MCA when there are no messages to send. The pulse unblocks the receiving MCA, which would otherwise remain in a wait state until a message arrived or the disconnect interval expired.

heartbeat interval

The time, in seconds, that is to elapse between heartbeat flows.

hierarchy

In publish/subscribe messaging topology, a local queue manager connected to a parent queue manager.

HTTP See Hypertext Transfer Protocol.

Hypertext Transfer Protocol (HTTP)

An Internet protocol that is used to transfer and display hypertext and XML documents on the web.

I**IBM MQ**

A family of IBM licensed programs that provides message queuing services.

IBM MQ Administration Interface (MQAI)

A programming interface that performs administration tasks on an IBM MQ queue manager through the use of data bags. Data bags allow the user to handle properties (or parameters) of IBM MQ objects.

IBM MQ classes for .NET

A set of classes that allow a program written in the .NET programming framework to connect to IBM MQ as an IBM MQ client or to connect directly to an IBM MQ server.

IBM MQ classes for C++

A set of classes that encapsulate the IBM MQ Message Queue Interface (MQI) in the C++ programming language.

IBM MQ classes for Java

A set of classes that encapsulate the IBM MQ Message Queue Interface (MQI) in the Java programming language.

IBM MQ fully managed .NET client

Part of an IBM MQ product that can be installed on a system without installing the full queue manager. The IBM MQ .NET client is used by fully-managed .NET applications and communicates with a queue manager on a server system. A .NET application that is not fully managed uses the IBM MQ MQI client. See also client, IBM MQ Java client, IBM MQ MQI client.

IBM MQ Java client

Part of an IBM MQ product that can be installed on a system without installing the full queue manager. The IBM MQ Java client is used by Java applications (both IBM MQ classes for Java and IBM MQ classes for JMS) and communicates with a queue manager on a server system. See also client, IBM MQ fully managed .NET client, IBM MQ MQI client.

IBM MQ MQI client

Part of an IBM MQ product that can be installed on a system without installing the full queue manager. The IBM MQ MQI client accepts MQI calls from applications and communicates with a queue manager on a server system. See also client, IBM MQ fully managed .NET client, IBM MQ Java client.

IBM MQ script commands (MQSC)

Human readable commands, uniform across all platforms, that are used to manipulate IBM MQ objects. See also programmable command format.

IBM MQ server

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 system, that is, on the MQI server machine. A server can support normal local MQI applications as well.

IBM MQ Telemetry

Supports small client libraries that can be embedded into smart devices running on a number of different device platforms. Applications built with the clients use the MQ Telemetry Transport (MQTT) protocol and the IBM MQ telemetry service to publish and subscribe messages reliably with IBM MQ.

IBM MQ Telemetry daemon for devices

See MQTT daemon for devices.

identification

The security service that enables each user of a computer system to be identified uniquely. A common mechanism for implementing this service is to associate a user ID with each user.

identity context

Information that identifies the user of the application that first puts the message on a queue

IFCID See instrumentation facility component identifier.

ILE See Integrated Language Environment®.

immediate shutdown

In IBM MQ, a shutdown of a queue manager that does not wait for applications to disconnect. Current message queue interface (MQI) calls are allowed to complete, but new MQI calls fail after an immediate shutdown has been requested. See also preemptive shutdown, quiesced shutdown.

impersonation

A breach of communication security in which the information is passed to a person posing as the intended receiver or information is sent by a person posing as someone else. See also eavesdropping, tampering.

inbound channel

A channel that receives messages from another queue manager.

in-built format

See built-in format.

index In the IBM MQ Administration Interface (MQAI), a means of referencing data items.

indoubt unit of recovery

The status of a unit of recovery for which a sync point has been requested but not yet confirmed.

inflight

The state of a resource or unit of recovery that has not yet completed the prepare phase of the commit process.

inheritance

An object-oriented programming technique in which existing classes are used as a basis for creating other classes. Through inheritance, more specific elements incorporate the structure and behavior of more general elements.

initialization input data set

A data set that is used by IBM MQ for z/OS when it starts.

initiation queue

A local queue on which the queue manager puts trigger messages.

initiator

In distributed queuing, a program that requests network connections on another system. See also responder.

input parameter

A parameter of an MQI call in which information is supplied.

insertion order

In the IBM MQ Administration Interface (MQAI), the order that data items are placed into a data bag.

installable service

In IBM MQ on UNIX and Linux systems and IBM MQ for Windows, additional functionality provided as independent component. The installation of each component is optional: in-house or third-party components can be used instead.

instance

A specific occurrence of an object that belongs to a class. See also object.

instance data

In object-oriented programming, state information associated with an object.

intermediate certificate

A signer certificate that is not the root certificate. It is issued by the trusted root specifically to issue end-entity server certificates. The result is a certificate chain that begins at the trusted root CA, through a number of intermediates, and ends with the SSL certificate issued to the organization.

instrumentation event

A way of monitoring queue manager resource definitions, performance conditions, and channel conditions in a network of IBM MQ systems.

instrumentation facility component identifier (IFCID)

In Db2 for z/OS, a value that names and identifies a trace record of an event. As a parameter on the START TRACE and MODIFY TRACE commands, it specifies that the corresponding event is to be traced.

Integrated Language Environment (ILE)

A set of constructs and interfaces that provides a common runtime environment and runtime bindable application programming interfaces (APIs) for all ILE-conforming high-level languages.

Interactive Problem Control System (IPCS)

A component of MVS™ and z/OS that permits online problem management, interactive problem diagnosis, online debugging for disk-resident abend dumps, problem tracking, and problem reporting.

Interactive System Productivity Facility (ISPF)

An IBM licensed program that serves as a full-screen editor and dialog manager. Used for writing application programs, it provides a means of generating standard screen panels and interactive dialogs between the application programmer and the terminal user.

interface

In object-oriented programming, an abstract model of behavior; a collection of functions or methods.

Internet Protocol (IP)

A protocol that routes data through a network or interconnected networks. This protocol acts as an intermediary between the higher protocol layers and the physical network. See also Transmission Control Protocol.

interprocess communication (IPC)

The process by which programs send messages to each other. Sockets, semaphores, signals, and internal message queues are common methods of interprocess communication. See also client.

intersystem communication (ISC)

A CICS facility that provides inbound and outbound support for communication from other computer systems.

- IP** See Internet Protocol.
- IPC** See interprocess communication.
- IPCS** See Interactive Problem Control System.
- ISC** See intersystem communication.
- ISPF** See Interactive System Productivity Facility.

J

JAAS See Java Authentication and Authorization Service.

Java Authentication and Authorization Service (JAAS)

In Java EE technology, a standard API for performing security-based operations. Through JAAS, services can authenticate and authorize users while enabling the applications to remain independent from underlying technologies.

Java Message Service (JMS)

An application programming interface that provides Java language functions for handling messages. See also Message Queue Interface.

Java runtime environment (JRE)

A subset of a Java developer kit that contains the core executable programs and files that constitute the standard Java platform. The JRE includes the Java virtual machine (JVM), core classes, and supporting files.

JMS See Java Message Service.

JMSAdmin

An administration tool that enables administrators to define the properties of JMS objects and to store them within a JNDI namespace

journal

A feature of OS/400® that IBM MQ for IBM i uses to control updates to local objects. Each queue manager library contains a journal for that queue manager.

JRE See Java runtime environment.

K

keepalive

A TCP/IP mechanism where a small packet is sent across the network at predefined intervals to determine whether the socket is still working correctly.

Kerberos

A network authentication protocol that is based on symmetric key cryptography. Kerberos assigns a unique key, called a ticket, to each user who logs on to the network. The ticket is embedded in messages that are sent over the network. The receiver of a message uses the ticket to authenticate the sender.

key authentication

See authentication.

key repository

A store for digital certificates and their associated private keys.

key ring

In computer security, a file that contains public keys, private keys, trusted roots, and certificates.

keystore

In security, a file or a hardware cryptographic card where identities and private keys are stored, for authentication and encryption purposes. Some keystores also contain trusted or public keys. See also truststore.

L**last will and testament**

An object that is registered with a monitor by a client, and used by the monitor if the client ends unexpectedly.

LDAP See Lightweight Directory Access Protocol.

Lightweight Directory Access Protocol (LDAP)

An open protocol that uses TCP/IP to provide access to directories that support an X.500 model and that does not incur the resource requirements of the more complex X.500 Directory Access Protocol (DAP). For example, LDAP can be used to locate people, organizations, and other resources in an Internet or intranet directory.

linear logging

In IBM MQ on UNIX and Linux systems, and IBM MQ for Windows, the process of keeping restart data in a sequence of files. New files are added to the sequence as necessary. The space in which the data is written is not reused. See also circular logging. For further information, see the topic Types of logging.

link level security

The security services that are started, directly or indirectly, by a message channel agent (MCA), the communications subsystem, or a combination of the two working together.

listener

A program that detects incoming requests and starts the associated channel.

local definition of a remote queue

An IBM MQ object belonging to a local queue manager that defines the attributes of a queue that is owned by another queue manager. In addition, it is used for queue manager aliasing and reply-to-queue aliasing.

locale A setting that identifies language or geography and determines formatting conventions such as collation, case conversion, character classification, the language of messages, date and time representation, and numeric representation.

locally defined object

On z/OS, an object whose definition is stored on page set zero. The definition can be accessed only by the queue manager that defined it. See also globally defined object.

local queue

A queue that belongs to the local queue manager. A local queue can contain a list of messages waiting to be processed. See also remote queue.

local queue manager

The queue manager to which the program is connected and that provides message queuing services to the program. See also remote queue manager.

log

In IBM MQ, a file recording the work done by queue managers while they receive, transmit, and deliver messages, to enable them to recover in the event of failure.

log control file

In IBM MQ on UNIX and Linux systems, and IBM MQ for Windows, the file containing information needed to monitor the use of log files (for example, their size and location, and the name of the next available file).

log file

In IBM MQ on UNIX and Linux systems, and IBM MQ for Windows, a file in which all

significant changes to the data controlled by a queue manager are recorded. If the primary log files become full, IBM MQ allocates secondary log files.

logical unit (LU)

An access point through which a user or application program accesses the SNA network to communicate with another user or application program.

logical unit 6.2 (LU 6.2)

An SNA logical unit that supports general communication between programs in a distributed processing environment.

logical unit of work identifier (LUWID)

A name that uniquely identifies a thread within a network. This name consists of a fully qualified logical unit network name, a logical unit of work instance number, and a logical unit of work sequence number.

log record

A set of data that is treated as a single unit in a log file.

log record sequence number (LRSN)

A unique identifier for a log record that is associated with a data sharing member. Db2 for z/OS uses the LRSN for recovery in the data sharing environment.

LRSN See log record sequence number.

LU See logical unit.

LU 6.2 See logical unit 6.2.

LU 6.2 conversation

In SNA, a logical connection between two transaction programs over an LU 6.2 session that enables them to communicate with each other.

LU 6.2 conversation level security

In SNA, a conversation level security protocol that enables a partner transaction program to authenticate the transaction program that initiated the conversation.

LU 6.2 session

In SNA, a session between two logical units (LUs) of type 6.2.

LU name

The name by which VTAM[®] refers to a node in a network.

LUWID

See logical unit of work identifier.

M

managed destination

A queue that is provided by the queue manager, as the destination to which published messages are to be sent, for an application that elects to use a managed subscription. See also managed subscription.

managed handle

An identifier that is returned by the MQSUB call when a queue manager is specified to manage the storage of messages that are sent to the subscription.

managed subscription

A subscription for which the queue manager creates a subscriber queue to receive publications because the application does not require a specific queue to be used. See also managed destination.

marshalling

See serialization.

MCA See message channel agent.

MCI See message channel interface.

media image

In IBM MQ on UNIX and Linux systems and IBM MQ for Windows, the sequence of log records that contain an image of an object. The object can be re-created from this image.

message

1. In system programming, information intended for the terminal operator or system administrator.
2. A string of bytes that is passed from one application to another. Messages typically comprise a message header (used for message routing and identification) and a payload (containing the application data being sent). The data has a format that is compatible with both the sending and receiving application.

message affinity

The relationship between conversational messages that are exchanged between two applications, where the messages must be processed by a particular queue manager or in a particular sequence.

message channel

In distributed message queuing, a mechanism for moving messages from one queue manager to another. A message channel comprises two message channel agents (a sender at one end and a receiver at the other end) and a communication link. See also channel.

message channel agent (MCA)

A program that transmits prepared messages from a transmission queue to a communication link, or from a communication link to a destination queue. See also Message Queue Interface.

message channel interface (MCI)

The IBM MQ interface to which customer-written or vendor-written programs that transmit messages between a IBM MQ queue manager and another messaging system must conform. See also Message Queue Interface.

message consumer

1. In JMS, an object that is created within a session to receive messages from a destination.
2. A program, function, or organization that gets and processes messages. See also consumer.

message context

Information about the originator of a message that is held in fields in the message descriptor. There are two categories of context information: identity context and origin context.

message descriptor

Control information describing the message format and presentation that is carried as part of a IBM MQ message. The format of the message descriptor is defined by the MQMD structure.

message exit

A type of channel exit program that is used to modify the contents of a message. Message exits usually work in pairs, one at each end of a channel. At the sending end of a channel, a message exit is called after the message channel agent (MCA) has got a message from the transmission queue. At the receiving end of a channel, a message exit is called before the message channel agent (MCA) puts a message on its destination queue.

message flow control

A distributed queue management task that involves setting up and maintaining message routes between queue managers.

Message Format Service (MFS)

An IMS editing facility that allows application programs to deal with simple logical messages instead of device-dependent data, thus simplifying the application development process.

message group

A logical group of related messages. The relationship is defined by the application putting the messages, and ensures that the messages are retrieved in the sequence put if both the producer and consumer honor the grouping.

message handle

A reference to a message. The handle can be used to obtain access to the message properties of the message.

message header

The part of a message that contains control information such as a unique message ID, the sender and receiver of the message, the message priority, and the type of message.

message input descriptor (MID)

The Message Format Service (MFS) control block that describes the format of the data presented to the application program. See also message output descriptor.

message listener

An object that acts as an asynchronous message consumer.

message output descriptor (MOD)

The Message Format Service (MFS) control block that describes the format of the output data produced by the application program. See also message input descriptor.

message priority

In IBM MQ, an attribute of a message that can affect the order in which messages on a queue are retrieved, and whether a trigger event is generated.

message producer

In JMS, an object that is created by a session and that is used to send messages to a destination. See also producer.

message property

Data associated with a message, in name-value pair format. Message properties can be used as message selectors to filter publications or to selectively get messages from queues. Message properties can be used to include business data or state information about processing without having to alter the message body.

Message Queue Interface (MQI)

The programming interface provided by IBM MQ queue managers. The programming interface allows application programs to access message queuing services. See also Java Message Service, message channel agent, message channel interface.

message queue management (MQM)

In IBM MQ for HP NonStop Server, a facility that provides access to PCF command formats and control commands to manage queue managers, queues, and channels.

message queuing

A programming technique in which each program within an application communicates with the other programs by putting messages on queues.

message-retry

An option available to an MCA that is unable to put a message. The MCA can wait for a predefined amount of time and then try to put the message again.

message segment

One of a number of segments of a message that is too large either for the application or for the queue manager to handle.

message selector

In application programming, a variable-length string that is used by an application to register its

interest in only those messages whose properties satisfy the Structured Query Language (SQL) query that the selection string represents. The syntax of a message selector is based on a subset of the SQL92 conditional expression syntax.

message sequence numbering

A programming technique in which messages are given unique numbers during transmission over a communication link. This number enables the receiving process to check whether all messages are received, to place them in a queue in the original order, and to discard duplicate messages.

message token

A unique identifier of a message within an active queue manager.

method

In object-oriented design or programming, the software that implements the behavior specified by an operation.

MFS See Message Format Service.

MGAS

See mostly global address space.

Microsoft Cluster Server (MSCS)

A technology that provides high availability by grouping computers running Windows into MSCS clusters. If one of the computers in the cluster hits any one of a range of problems, MSCS shuts down the disrupted application in an orderly manner, transfers its state data to another computer in the cluster, and re-initiates the application there. For further information, see the topic Supporting the Microsoft Cluster Service (MSCS).

Microsoft Transaction Server (MTS)

A facility that helps Windows users run business logic applications in a middle tier server. MTS divides work up into activities, which are short independent chunks of business logic.

MID See message input descriptor.

MOD See message output descriptor.

model queue object

A set of queue attributes that act as a template when a program creates a dynamic queue.

mostly global address space (MGAS)

A flexible virtual address space model, used in systems such as HP-UX, that preserves most of the address space for shared applications. This model can enhance performance for processes that share a lot of data. See also mostly private address space.

mostly private address space (MPAS)

A flexible virtual address space model, used in systems such as HP-UX, that can allocate larger address space blocks to processes. This model can enhance performance for processes that require a lot of data space. See also mostly global address space.

MPAS See mostly private address space.

MQAI See IBM MQ Administration Interface.

MQI See Message Queue Interface.

MQI channel

A connection between an IBM MQ client and a queue manager on a server system. An MQI channel transfers only MQI calls and responses in a bidirectional manner. See also channel.

MQM See message queue management.

MQSC

See IBM MQ script commands.

MQSeries

A previous name for IBM MQ and IBM WebSphere MQ.

MQ Telemetry Transport (MQTT)

An open, lightweight publish/subscribe messaging protocol flowing over TCP/IP to connect large numbers of devices such as servos, actuators, smart phones, vehicles, homes, health, remote sensors, and control devices. MQTT is designed for constrained devices and low-bandwidth, high-latency or unreliable networks. The protocol minimises network bandwidth and device resource requirements, while ensuring some degree of reliability and assured delivery. It is used increasingly in the "machine-to-machine" (M2M) or "Internet of Things" world of connected devices, and for mobile applications where bandwidth and battery power are at a premium.

MQTT

See MQ Telemetry Transport.

MQTT client

An MQTT client application connects to MQTT capable servers such as IBM MQ Telemetry channels. You can write your own clients to use the published protocol, or download the free Paho clients. A typical client is responsible for collecting information from a telemetry device and publishing the information to the server. It can also subscribe to topics, receive messages, and use this information to control the telemetry device.

MQTT daemon for devices

The MQTT daemon for devices was an advanced MQTT V3 client. It was a very small footprint MQTT server designed for embedded systems. Its primary use was to store and forward messages from telemetry devices and other MQTT clients, including other MQTT daemons for devices.

MQTT server

A messaging server that supports the MQ Telemetry Transport protocol. It enables mobile apps and devices, supported by MQTT clients, to exchange messages. It typically allows many MQTT clients to connect to it at the same time, and provides a hub for messages distribution to the MQTT clients. MQTT servers are available from IBM and others. IBM MessageSight and IBM MQ Telemetry are MQTT servers from IBM.

MQXR service

See telemetry service.

MSCS See Microsoft Cluster Server. For further information, see the topic Supporting the Microsoft Cluster Service (MSCS).

MTS See Microsoft Transaction Server.

multi-hop

To pass through one or more intermediate queue managers when there is no direct communication link between a source queue manager and the target queue manager.

multi-instance queue manager

A queue manager that is configured to share the use of queue manager data with other queue manager instances. One instance of a running multi-instance queue manager is active, other instances are on standby ready to take over from the active instance. See also queue manager, single instance queue manager.

N**namelist**

An IBM MQ object that contains a list of object names, for example, queue names.

name service

In IBM MQ on UNIX and Linux systems and IBM MQ for Windows, the facility that determines which queue manager owns a specified queue.

name service interface (NSI)

The IBM MQ interface to which customer-written or vendor-written programs that resolve queue-name ownership must conform.

name transformation

In IBM MQ on UNIX and Linux systems and IBM MQ for Windows, an internal process that changes a queue manager name so that it is unique and valid for the system being used. Externally, the queue manager name remains unchanged.

nested bag

In the IBM MQ Administration Interface (MQAI), a system bag that is inserted into another data bag

nesting

In the IBM MQ Administration Interface (MQAI), a means of grouping information returned from IBM MQ.

NetBIOS (Network Basic Input/Output System)

A standard interface to networks and personal computers that is used on local area networks to provide message, print-server, and file-server functions. Application programs that use NetBIOS do not have to handle the details of LAN data link control (DLC) protocols.

Network Basic Input/Output System

See NetBIOS.

New Technology File System (NTFS)

One of the native file systems in Windows operating environments.

node In Microsoft Cluster Server (MSCS), each computer in the cluster.

nondurable subscription

A subscription that exists only while the connection from the subscribing application to the queue manager remains open. The subscription is removed when the subscribing application disconnects from the queue manager either deliberately or by loss of connection. See also durable subscription.

nonpersistent message

A message that does not survive a restart of the queue manager. See also persistent message.

NSI See name service interface.

NTFS See New Technology File System.

NUL See null character.

null character (NUL)

A control character with the value of X'00' that represents the absence of a displayed or printed character.

O

OAM See object authority manager.

object

1. In IBM MQ, a queue manager, queue, process definition, channel, namelist, authentication information object, administrative topic object, listener, service object, or (on z/OS only) a CF structure object or storage class.
2. In object-oriented design or programming, a concrete realization (instance) of a class that consists of data and the operations associated with that data. An object contains the instance data that is defined by the class, but the class owns the operations that are associated with the data.

object authority manager (OAM)

In IBM MQ on UNIX and Linux systems, IBM MQ for IBM i, and IBM MQ for Windows, the default authorization service for command and object management. The OAM can be replaced by, or run in combination with, a customer-supplied security service.

object descriptor

A data structure that identifies a particular IBM MQ object. Included in the descriptor are the name of the object and the object type.

object handle

The identifier or token by which a program accesses the IBM MQ object with which it is working.

object-oriented programming

A programming approach based on the concepts of data abstraction and inheritance. Unlike procedural programming techniques, object-oriented programming concentrates not on how something is accomplished but instead on what data objects compose the problem and how they are manipulated.

OCSP See Online Certificate Status Protocol.

offloading

In IBM MQ for z/OS, an automatic process whereby a queue manager's active log is transferred to its archive log.

Online Certificate Status Protocol

A method of checking if a certificate is revoked.

one way authentication

In this method of authentication, the queue manager presents the certificate to the client, but the authentication is not checked from the client to the queue manager.

open To establish access to an object, such as a queue, topic, or hyperlink.

open systems interconnection (OSI)

The interconnection of open systems in accordance with standards of the International Organization for Standardization (ISO) for the exchange of information.

Open Transaction Manager Access (OTMA)

A component of IMS that implements a transaction-based, connectionless client/server protocol in an MVS sysplex environment. The domain of the protocol is restricted to the domain of the z/OS Cross-System Coupling Facility (XCF). OTMA connects clients to servers so that the client can support a large network (or a large number of sessions) while maintaining high performance.

OPM See original program model.

original program model (OPM)

The set of functions for compiling source code and creating high-level language programs before the Integrated Language Environment (ILE) model was introduced.

OSGi Alliance

A consortium of more than 20 companies, including IBM, that creates specifications to outline open standards for the management of voice, data and multimedia wireless and wired networks.

OSI See open systems interconnection.

OSI directory standard

The standard, known as X.500, that defines a comprehensive directory service, including an information model, a namespace, a functional model, and an authentication framework. X.500 also defines the Directory Access Protocol (DAP) used by clients to access the directory. The Lightweight Directory Access Protocol (LDAP) removes some of the burden of X.500 access from directory clients, making the directory available to a wider variety of machines and applications.

OTMA

See Open Transaction Manager Access.

outbound channel

A channel that takes messages from a transmission queue and sends them to another queue manager.

output log-buffer

In IBM MQ for z/OS, a buffer that holds recovery log records before they are written to the archive log.

output parameter

A parameter of an MQI call in which the queue manager returns information when the call completes or fails.

overloading

In object-oriented programming, the capability of an operator or method to have different meanings depending on the context. For example, in C++, a user can redefine functions and most standard operators when the functions and operators are used with class types. The method name or operator remains the same, but the method parameters differ in type, number, or both. This difference is collectively called the function's or the operator's signature and each signature requires a separate implementation.

P**page set**

A VSAM data set used when IBM MQ for z/OS moves data (for example, queues and messages) from buffers in main storage to permanent backing storage (DASD).

parent class

A class from which another class inherits instance methods, attributes, and instance variables. See also abstract class.

partial repository

A partial set of information about queue managers in a cluster. A partial repository is maintained by all cluster queue managers that do not host a full repository. See also full repository.

partner queue manager

See remote queue manager.

PassTicket

In RACF secured sign-on, a dynamically generated, random, one-time-use, password substitute that a workstation or other client can use to sign on to the host rather than sending a RACF password across the network.

PCF See programmable command format.

pending event

An unscheduled event that occurs as a result of a connect request from a CICS adapter.

percolation

In error recovery, the passing along a preestablished path of control from a recovery routine to a higher-level recovery routine.

performance event

A category of event indicating that a limit condition has occurred.

performance trace

An IBM MQ trace option where the trace data is to be used for performance analysis and tuning.

permanent dynamic queue

A dynamic queue that is deleted when it is closed only if deletion is explicitly requested. Permanent dynamic queues are recovered if the queue manager fails, so they can contain persistent messages. See also temporary dynamic queue.

persistent message

A message that survives a restart of the queue manager. See also nonpersistent message.

personal certificate

Certificate for which you own the corresponding private key. Associated with queue managers or applications.

PGM See Pragmatic General Multicast.

PID See process ID.

ping The command that sends an Internet Control Message Protocol (ICMP) echo-request packet to a gateway, router, or host with the expectation of receiving a reply.

PKCS Public Key Cryptography Standards. A set of standards for cryptography, of which:

7 is for messages

11 is for hardware security modules

12 is for the file format used in the key repository

PKI See public key infrastructure.

plain text

See cleartext.

point of recovery

In IBM MQ for z/OS, a set of backup copies of IBM MQ for z/OS page sets and the corresponding log data sets required to recover these page sets. These backup copies provide a potential restart point in the event of page set loss (for example, page set I/O error).

poison message

In a queue, an incorrectly formatted message that the receiving application cannot process. The message can be repeatedly delivered to the input queue and repeatedly backed out by the application.

polymorphism

An object-oriented programming characteristic that allows a method to perform differently, depending on the class that implements it. Polymorphism allows a subclass to override an inherited method without affecting the method of the parent class. Polymorphism also enables a client to access two or more implementations of an object from a single interface.

Pragmatic General Multicast (PGM)

A reliable multicast transport protocol that provides a reliable sequence of packets to multiple recipients simultaneously.

preemptive shutdown

In IBM MQ, a shutdown of a queue manager that does not wait for connected applications to disconnect, or for current MQI calls to complete. See also immediate shutdown, quiesced shutdown.

preferred computer

The primary computer used by an application running under Microsoft Cluster Server control. After a failover to another computer, MSCS monitors the preferred computer until it is repaired, and as soon as it is running correctly again, moves the application back to it.

principal

An entity that can communicate securely with another entity. A principal is identified by its associated security context, which defines its access rights.

privately defined object

See locally defined object.

private methods and instance data

In object-oriented programming, methods and instance data that are only accessible to the implementation of the same class.

process definition object

An IBM MQ object that contains the definition of an IBM MQ application. For example, a queue manager uses the definition when it works with trigger messages.

process ID (PID)

The unique identifier that represents a process. A process ID is a positive integer and is not reused until the process lifetime ends.

producer

An application that creates and sends messages. See also message producer, publisher.

programmable command format (PCF)

A type of IBM MQ message used by the following applications: user administration applications, to put PCF commands onto the system command input queue of a specified queue manager, user administration applications, to get the results of a PCF command from a specified queue manager, and a queue manager, as a notification that an event has occurred. See also IBM MQ script commands.

program temporary fix (PTF)

For System i[®], System p, and IBM Z Systems products, a package containing individual or multiple fixes that is made available to all licensed customers. A PTF resolves defects and might provide enhancements.

property

A characteristic of an object that describes the object. A property can be changed or modified. Properties can describe an object name, type, value, or behavior, among other things.

protected methods and instance data

In object-oriented programming, methods and instance data that are only accessible to the implementations of the same or derived classes, or from friend classes.

proxy subscription

A proxy subscription is a subscription made by one queue manager for topics published on another queue manager. A proxy subscription flows between queue managers for each individual topic string that is subscribed to by a subscription. You do not create proxy subscriptions explicitly, the queue manager does so on your behalf.

PTF See program temporary fix.

public key cryptography

A cryptography system that uses two keys: a public key known to everyone and a private or secret key known only to the recipient of the message. The public and private keys are related in such a way that only the public key can be used to encrypt messages and only the corresponding private key can be used to decrypt them.

public key infrastructure (PKI)

A system of digital certificates, certification authorities, and other registration authorities that verify and authenticate the validity of each party involved in a network transaction.

public methods and instance data

In object oriented programming, methods and instance data that are accessible to all classes.

public-private key cryptography

See public key cryptography.

publish

To make information about a specified topic available to a queue manager in a publish/subscribe system.

publisher

An application that makes information about a specified topic available to a broker in a publish/subscribe system. See also producer.

publish/subscribe

A type of messaging interaction in which information, provided by publishing applications, is delivered by an infrastructure to all subscribing applications that registered to receive that type of information.

publish/subscribe cluster

A set of queue managers that are fully interconnected and that form part of a multi queue manager network for publish/subscribe applications.

put In message queuing, to use the MQPUT or MQPUT1 calls to place messages on a queue. See also browse, get.

Q

queue An object that holds messages for message-queuing applications. A queue is owned and maintained by a queue manager.

queue index

In IBM MQ for z/OS, a list of message identifiers or a list of correlation identifiers that can be used to increase the speed of MQGET operations on the queue.

queue-manager

A component of a message queuing system that provides queuing services to applications. See also channel, multi-instance queue manager.

queue-manager-event

An event that indicates that an error condition has occurred in relation to the resources used by a queue manager (for example, a queue is unavailable), or a significant change has occurred in the queue manager (for example, a queue manager has stopped or started).

queue-manager-group

In a client channel definition table (CCDT), the group of queue managers a client tries to connect to when a connection is established to a server.

queue-manager-level-security

In IBM MQ for z/OS, the authorization checks that are performed using RACF profiles specific to a queue manager.

queue-manager-set

A grouping of queue managers in IBM MQ Explorer that allows a user to perform actions on all of the queue managers in the group.

queue-sharing group

In IBM MQ for z/OS, a group of queue managers in the same sysplex that can access a single set of object definitions stored in the shared repository, and a single set of shared queues stored in the coupling facility. See also shared queue.

queue-sharing group level security

In IBM MQ for z/OS, the authorization checks that are performed using RACF profiles that are shared by all queue managers in a queue-sharing group.

quiesce

To end a process or shut down a system after allowing normal completion of active operations.

quiesced shutdown

1. In IBM MQ, a shutdown of a queue manager that allows all connected applications to disconnect. See also immediate shutdown, preemptive shutdown.

2. A type of shutdown of the CICS adapter where the adapter disconnects from IBM MQ, but only after all the currently active tasks have been completed. See also forced shutdown.

quiescing

In IBM MQ, the state of a queue manager before it stops. In this state, programs are allowed to finish processing, but no new programs are allowed to start.

quorum disk

The disk accessed exclusively by Microsoft Cluster Server to store the cluster recovery log, and to determine whether a server is up or down. Only one server can own the quorum disk at a time. Servers in the cluster can negotiate for the ownership.

R

RACF See Resource Access Control Facility.

RAID See Redundant Array of Independent Disks.

RBA See relative byte address.

RC See return code.

read ahead

An option that allows messages to be sent to a client before an application requests them.

reason code

A return code that describes the reason for the failure or partial success of a Message Queue Interface (MQI) call.

receive exit

A type of channel exit program that is called just after the message channel agent (MCA) has regained control following a communications receive and has received a unit of data from a communications connection. See also send exit.

receiver channel

In message queuing, a channel that responds to a sender channel, takes messages from a communication link, and puts them on a local queue.

recovery log

In IBM MQ for z/OS, data sets containing information needed to recover messages, queues, and the IBM MQ subsystem. See also archive log.

recovery termination manager (RTM)

A program that handles all normal and abnormal termination of tasks by passing control to a recovery routine associated with the terminating function.

Redundant Array of Independent Disks (RAID)

A collection of two or more physical disk drives that present to the host an image of one or more logical disk drives. In the event of a physical device failure, the data can be read or regenerated from the other disk drives in the array due to data redundancy.

reference message

A message that refers to a piece of data that is to be transmitted. The reference message is handled by message exit programs, which attach and detach the data from the message so allowing the data to be transmitted without having to be stored on any queues.

registry

A repository that contains access and configuration information for users, systems, and software.

Registry Editor

In Windows systems, the program that allows the user to edit the registry.

registry hive

In Windows systems, the structure of the data stored in the registry.

relative byte address (RBA)

The offset of a data record or control interval from the beginning of the storage space that is allocated to the data set or file to which it belongs.

reliable multicast messaging (RMM)

A high-throughput low-latency transport fabric designed for one-to-many data delivery or many-to-many data exchange, in a message-oriented middleware publish/subscribe fashion. RMM uses the IP multicast infrastructure to ensure scalable resource conservation and timely information distribution.

remote queue

A queue that belongs to a remote queue manager. Programs can put messages on remote queues, but they cannot get messages from remote queues. See also local queue.

remote queue manager

A queue manager to which a program is not connected, even if it is running on the same system as the program. See also local queue manager.

remote queue object

An IBM MQ object belonging to a local queue manager. This object defines the attributes of a queue that is owned by another queue manager. In addition, it is used for queue manager aliasing and reply-to-queue aliasing.

remote queuing

In message queuing, the provision of services to enable applications to put messages on queues belonging to other queue managers.

reply message

A type of message used for replies to request messages. See also report message, request message.

reply-to queue

The name of a queue to which the program that issued an MQPUT call wants a reply message or report message sent.

report message

A type of message that gives information about another message. A report message can indicate that a message has been delivered, has arrived at its destination, has expired, or could not be processed for some reason. See also reply message, request message.

repository

A collection of information about the queue managers that are members of a cluster. This information includes queue manager names, their locations, their channels, and what queues they host.

repository queue manager

A queue manager that hosts the full repository of information about a cluster.

requester channel

In message queuing, a channel that can be started locally to initiate operation of a server channel. See also server channel.

request message

A type of message used to request a reply from another program. See also reply message, report message.

request/reply

A type of messaging application in which a request message is used to request a reply from another application. See also datagram.

RESLEVEL

In IBM MQ for z/OS, an option that controls the number of user IDs checked for API-resource security.

resolution path

The set of queues that are opened when an application specifies an alias or a remote queue on input to an MQOPEN call.

resource

A facility of a computing system or operating system required by a job, task, or running program. Resources include main storage, input/output devices, the processing unit, data sets, files, libraries, folders, application servers, and control or processing programs.

Resource Access Control Facility (RACF)

An IBM licensed program that provides access control by identifying users to the system; verifying users of the system; authorizing access to protected resources; logging unauthorized attempts to enter the system; and logging accesses to protected resources.

resource adapter

An implementation of the Java Enterprise Edition Connector Architecture that allows JMS applications and message driven beans, running in an application server, to access the resources of an IBM MQ queue manager.

resource manager

An application, program, or transaction that manages and controls access to shared resources such as memory buffers and data sets. IBM MQ, CICS, and IMS are resource managers.

Resource Recovery Services (RRS)

A component of z/OS that uses a sync point manager to coordinate changes among participating resource managers.

responder

In distributed queuing, a program that replies to network connection requests from another system. See also initiator.

resynch

In IBM MQ, an option to direct a channel to start and resolve any in-doubt status messages, but without restarting message transfer.

return code (RC)

A value returned by a program to indicate the result of its processing. Completion codes and reason codes are examples of return codes.

return-to-sender

An option available to an MCA that is unable to deliver a message. The MCA can send the message back to the originator.

Rivest-Shamir-Adleman algorithm (RSA)

A public-key encryption technology developed by RSA Data Security, Inc, and used in the IBM implementation of SSL.

RMM See reliable multicast messaging.

rollback

See backout.

root certificate

The top certificate in the chain. If this is a self-signed certificate, it is used only for signing other certificates. See also self-signed certificate.

RRS See Resource Recovery Services.

RSA See Rivest-Shamir-Adleman algorithm.

RTM See recovery termination manager.

rules table

A control file containing one or more rules that the dead-letter queue handler applies to messages on the dead letter queue (DLQ).

S

SAF See store and forward.

Scalable Parallel 2 (SP2)

IBM's parallel UNIX system: effectively parallel AIX systems on a high-speed network.

SDK See software development kit.

SDWA

See system diagnostic work area.

SECMEC

See security mechanism.

Secure Sockets Layer (SSL)

A security protocol that provides communication privacy. With SSL, client/server applications can communicate in a way that is designed to prevent eavesdropping, tampering, and message forgery. See also certificate authority.

security enabling interface (SEI)

The IBM MQ interface to which customer-written or vendor-written programs that check authorization, supply a user identifier, or perform authentication must conform.

security exit

A channel exit program that is called immediately after the initial data negotiation has completed on channel startup. Security exits normally work in pairs and can be called on both message channels and MQI channels. The primary purpose of the security exit is to enable the message channel agent (MCA) at each end of a channel to authenticate its partner.

security identifier (SID)

On Windows systems, a supplement to the user ID that identifies the full user account details on the Windows security account manager database where the user is defined.

security mechanism (SECMEC)

A technical tool or technique that is used to implement a security service. A mechanism might operate by itself, or in conjunction with others, to provide a particular service. Examples of security mechanisms include access control lists, cryptography, and digital signatures.

security message

One of the messages, sent by security exits that are called at both ends of a channel, to communicate with each other. The format of a security message is not defined and is determined by the user.

security service

A service within a computer system that protect its resources. Access control is an example of a security service.

Security Support Provider Interface (SSI)

The means for networked applications to call one of several security support providers (SSPs) to establish authenticated connections and to exchange data securely over those connections. It is available for use on Windows systems.

segmentation

The division of a message that is too large for a queue manager, queue, or application, into a number of smaller physical messages, which are then reassembled by the receiving queue manager or application.

SEI See security enabling interface.

selector

An identifier for a data item. In the IBM MQ Administration Interface (MQAI), there are two types of selector: a user selector and a system selector.

self-signed certificate

A certificate that is constructed like a digital certificate, but is signed by its subject. Unlike a digital certificate, a self-signed certificate cannot be used in a trustworthy manner to authenticate a public key to other parties.

semaphore

In UNIX and Linux systems, a general method of communication between two processes that extends the features of signals.

sender channel

In message queuing, a channel that initiates transfers, removes messages from a transmission queue, and moves them over a communication link to a receiver or requester channel.

send exit

A type of channel exit program that is called just before a message channel agent (MCA) issues a communications send to send a unit of data over a communications connection. See also receive exit.

Sequenced Packet Exchange protocol (SPX)

A session-oriented network protocol that provides connection-oriented services between two nodes on the network, and is used primarily by client/server applications. It relies on the Internet Packet Exchange (IPX) protocol, provides flow control and error recovery, and guarantees reliability of the physical network.

sequence number wrap value

In IBM MQ, a method of ensuring that both ends of a communication link reset their current message sequence numbers at the same time. Transmitting messages with a sequence number ensures that the receiving channel can reestablish the message sequence when storing the messages.

serialization

In object-oriented programming, the writing of data in sequential fashion to a communications medium from program memory.

server

1. A software program or a computer that provides services to other software programs or other computers. See also client.
2. A queue manager that provides queue services to client applications running on a remote workstation.

server channel

In message queuing, a channel that responds to a requester channel, removes messages from a transmission queue, and moves them over a communication link to the requester channel. See also requester channel.

server-connection channel type

The type of MQI channel definition associated with the server that runs a queue manager. See also client-connection channel type.

service interval

A time interval, against which the elapsed time between a put or a get and a subsequent get is compared by the queue manager in deciding whether the conditions for a service interval event have been met. The service interval for a queue is specified by a queue attribute.

service interval event

An event related to the service interval.

service object

An object that can start additional processes when the queue manager starts and can stop the processes when the queue manager stops.

session

A logical or virtual connection between two stations, software programs, or devices on a network that allows the two elements to communicate and exchange data for the duration of the session.

session ID

See session identifier.

session identifier (session ID)

In IBM MQ for z/OS, the identifier, unique to CICS, that defines the communication link to be used by a message channel agent when moving messages from a transmission queue to a link.

session-level authentication

In Systems Network Architecture (SNA), a session level security protocol that enables two logical units (LUs) to authenticate each other while they are activating a session. Session level authentication is also known as LU-LU verification.

session-level cryptography

In Systems Network Architecture (SNA), a method of encrypting and decrypting data that flows on a session between two logical units (LUs).

shared inbound channel

In IBM MQ for z/OS, a channel that was started by a listener using the group port. The channel definition of a shared channel can be stored either on page set zero (private) or in the shared repository (global).

shared outbound channel

In IBM MQ for z/OS, a channel that moves messages from a shared transmission queue. The channel definition of a shared channel can be stored either on page set zero (private) or in the shared repository (global).

shared queue

In IBM MQ for z/OS, a type of local queue. The messages on the queue are stored in the coupling facility and can be accessed by one or more queue managers in a queue-sharing group. The definition of the queue is stored in the shared repository. See also queue-sharing group.

shared repository

In IBM MQ for z/OS, a shared Db2 database that is used to hold object definitions that have been defined globally.

sharing conversations

The facility for more than one conversation to share a channel instance, or the conversations that share a channel instance.

shell A software interface between users and an operating system. Shells generally fall into one of two categories: a command line shell, which provides a command line interface to the operating system; and a graphical shell, which provides a graphical user interface (GUI).

SID See security identifier.

signal A mechanism by which a process can be notified of, or affected by, an event occurring in the system. Examples of such events include hardware exceptions and specific actions by processes.

signaling

In IBM MQ for z/OS and IBM MQ, a feature that allows the operating system to notify a program when an expected message arrives on a queue.

signature

The collection of types associated with a method. The signature includes the type of the return value, if any, as well as the number, order, and type of each of the method's arguments.

signer certificate

The digital certificate that validates the issuer of a certificate. For a CA, the signer certificate is the root CA certificate. For a user who creates a self-signed certificate for testing purposes, the signer certificate is the user's personal certificate.

single instance queue manager

A queue manager that does not have multiple instances. See also multi-instance queue manager.

single logging

A method of recording IBM MQ for z/OS activity where each change is recorded on one data set only. See also dual logging.

single-phase backout

A method in which an action in progress must not be allowed to finish, and all changes that are part of that action must be undone.

single-phase commit

A method in which a program can commit updates to a commitment resource without coordinating those updates with updates the program has made to resources controlled by another resource manager.

SIT See system initialization table.

SMF See System Management Facilities.

SNA See Systems Network Architecture.

software development kit (SDK)

A set of tools, APIs, and documentation to assist with the development of software in a specific computer language or for a particular operating environment.

source queue manager

See local queue manager.

SP2 See Scalable Parallel 2.

SPX See Sequenced Packet Exchange protocol.

SSI See Security Support Provider Interface.

SSL See Secure Sockets Layer.

SSLPeer

The value in the issuer represents the distinguished name of the remote personal certificate.

SSL or TLS client

The initiating end of the connection. One outbound channel from a queue manager is also an SSL or TLS client.

standby queue manager instance

An instance of a running multi-instance queue manager ready to take over from the active instance. There are one or more standby instances of a multi-instance queue manager.

stanza A group of lines in a file that together have a common function or define a part of the system. Stanzas are usually separated by blank lines or colons, and each stanza has a name.

star-connected communications network

A network in which all nodes are connected to a central node.

storage class

In IBM MQ for z/OS, the page set that is to hold the messages for a particular queue. The storage class is specified when the queue is defined.

store and forward (SAF)

The temporary storing of packets, messages, or frames in a data network before they are retransmitted toward their destination.

streaming

In object-oriented programming, the serialization of class information and object instance data.

subscribe

To request information about a topic.

subsystem

In z/OS, a service provider that performs one or many functions but does nothing until a request is made. For example, each IBM MQ for z/OS queue manager or instance of a Db2 for z/OS database management system is a z/OS subsystem.

supervisor call (SVC)

An instruction that interrupts the program being run and passes control to the supervisor so that it can perform the specific service indicated by the instruction.

SVC See supervisor call.

switchover

The change from the active multi-instance queue manager instance to a standby instance. A switchover results from an operator intentionally stopping the active multi-instance queue manager instance.

switch profile

In IBM MQ for z/OS, a RACF profile used when IBM MQ starts up or when a refresh security command is issued. Each switch profile that IBM MQ detects turns off checking for the specified resource.

symmetric key cryptography

A system of cryptography in which the sender and receiver of a message share a single, common, secret key that is used to encrypt and decrypt the message. See also asymmetric key cryptography.

symptom string

Diagnostic information displayed in a structured format designed for searching the IBM software support database.

synchronous messaging

A method of communication between programs in which a program places a message on a message queue and then waits for a reply to its message before resuming its own processing. See also asynchronous messaging.

sync point

A point during the processing of a transaction at which protected resources are consistent.

sysplex

A set of z/OS systems that communicate with each other through certain multisystem hardware components and software services.

system bag

A type of data bag that is created by the MQAI.

system control commands

Commands used to manipulate platform-specific entities such as buffer pools, storage classes, and page sets.

system diagnostic work area (SDWA)

In a z/OS environment, the data that is recorded in a SYS1.LOGREC entry that describes a program or hardware error.

system initialization table (SIT)

A table containing parameters used by CICS at startup.

system item

A type of data item that is created by the MQAI.

System Management Facilities (SMF)

A component of z/OS that collects and records a variety of system and job-related information.

system selector

In the IBM MQ Administration Interface (MQAI), a system item identifier that is included in the data bag when it is created.

Systems Network Architecture (SNA)

The description of the logical structure, formats, protocols, and operational sequences for transmitting information through and controlling the configuration and operation of networks.

T**tampering**

A breach of communication security in which information in transit is changed or replaced and then sent on to the recipient. See also eavesdropping, impersonation.

target library high-level qualifier (thlqual)

A high-level qualifier for z/OS target data set names.

target queue manager

See remote queue manager.

task control block (TCB)

A z/OS control block that is used to communicate information about tasks within an address space that is connected to a subsystem.

task switching

The overlapping of I/O operations and processing between several tasks.

TCB See task control block.

TCP See Transmission Control Protocol.

TCP/IP

See Transmission Control Protocol/Internet Protocol.

technote

A short document about a single topic.

telemetry channel

A communication link between a queue manager on IBM MQ, and MQTT clients. Each channel might have one or more telemetry devices connected to it.

telemetry service

The telemetry service is a IBM MQ service that handles the server side of the MQTT protocol (see "MQTT Server"). The telemetry service hosts telemetry channels. It is sometimes referred to as the MQXR ("MQ extended reach") service.

temporary dynamic queue

A dynamic queue that is deleted when it is closed. Temporary dynamic queues are not recovered if the queue manager fails, so they can contain nonpersistent messages only. See also permanent dynamic queue.

teraspace

A one terabyte temporary storage area that provides storage that is private to a process.

termination notification

A pending event that is activated when a CICS subsystem successfully connects to IBM MQ for z/OS.

thlqual

See target library high-level qualifier.

thread A stream of computer instructions that is in control of a process. In some operating systems, a thread is the smallest unit of operation in a process. Several threads can run concurrently, performing different jobs.

TID See transaction identifier.

time-independent messaging

See asynchronous messaging.

TLS See Transport Layer Security.

TMF See Transaction Manager Facility.

TMI See trigger monitor interface.

topic host routing

An option for routing publications in a publish/subscribe cluster. With topic host routing, only selected cluster queue managers host the topic definitions. Publications from non-hosting queue managers are routed through the hosting queue managers to any queue manager in the cluster with a matching subscription.

TP See transaction program.

trace A record of the processing of a computer program or transaction. The information collected from a trace can be used to assess problems and performance.

transaction ID

See transaction identifier.

transaction identifier (TID, transaction ID, XID)

A unique name that is assigned to a transaction and is used to identify the actions associated with that transaction.

transaction manager

A software unit that coordinates the activities of resource managers by managing global transactions and coordinating the decision to commit them or roll them back.

Transaction Manager Facility (TMF)

In IBM MQ for HP NonStop Server, a subsystem to protect business transactions and the integrity of databases.

transaction program (TP)

A program that processes transactions in an SNA network.

Transmission Control Protocol (TCP)

A communication protocol used in the Internet and in any network that follows the Internet Engineering Task Force (IETF) standards for internetwork protocol. TCP provides a reliable host-to-host protocol in packet-switched communication networks and in interconnected systems of such networks. See also Internet Protocol.

Transmission Control Protocol/Internet Protocol (TCP/IP)

An industry-standard, nonproprietary set of communication protocols that provides reliable end-to-end connections between applications over interconnected networks of different types.

transmission program

See message channel agent.

transmission queue

A local queue on which prepared messages destined for a remote queue manager are temporarily stored.

Transport Layer Security

A set of encryption rules that uses verified certificates and encryption keys to secure communications over the Internet. TLS is an update to the SSL protocol.

triggered queue

A local queue, usually an application queue, that has triggering enabled so that a message is written when a trigger event occurs. The trigger message is often written to an initiation queue.

trigger event

An event, such as a message arriving on a queue, that causes a queue manager to create a trigger message on an initiation queue.

triggering

In IBM MQ, a facility that allows a queue manager to start an application automatically when predetermined conditions on a queue are satisfied.

trigger message

A message that contains information about the program that a trigger monitor is to start.

trigger monitor

A continuously running application that serves one or more initiation queues. When a trigger message arrives on an initiation queue, the trigger monitor retrieves the message. It uses the information in the trigger message to start a process that serves the queue on which a trigger event occurred.

trigger monitor interface (TMI)

The IBM MQ interface to which customer-written or vendor-written trigger monitor programs must conform.

truststore

In security, a storage object, either a file or a hardware cryptographic card, where public keys are stored in the form of trusted certificates, for authentication purposes. In some applications, these trusted certificates are moved into the application keystore to be stored with the private keys.

two-phase commit

A two-step process by which recoverable resources and an external subsystem are committed. During the first step, the database manager subsystems are polled to ensure that they are ready to commit. If all subsystems respond positively, the database manager instructs them to commit.

two way authentication

In this method of authentication, the queue manager and the client, present the certificate to each other. Also known as mutual authentication.

type A characteristic that specifies the internal format of data and determines how the data can be used.

U

UDP See User Datagram Protocol.

unauthorized access

Gaining access to resources within a computer system without permission.

undelivered message queue

See dead-letter queue.

undo/redo record

A log record used in recovery. The redo part of the record describes a change to be made to an IBM MQ object. The undo part describes how to back out the change if the work is not committed.

unit of recovery

A recoverable sequence of operations within a single resource manager, such as an instance of Db2 for z/OS. See also unit of work.

unit of work (UOW)

A recoverable sequence of operations performed by an application between two points of

consistency. A unit of work begins when a transaction starts or at a user-requested sync point. It ends either at a user-requested sync point or at the end of a transaction.

UOW See unit of work.

user bag

In the MQAI, a type of data bag that is created by the user.

User Datagram Protocol (UDP)

An Internet protocol that provides unreliable, connectionless datagram service. It enables an application program on one machine or process to send a datagram to an application program on another machine or process.

user item

In the MQAI, a type of data item that is created by the user.

user selector

In the IBM MQ Administration Interface (MQAI), the identifier that is placed with a data item into a data bag to identify the data item. IBM MQ provides predefined user selectors for IBM MQ objects.

user token (UTOKEN)

The RACF security token that encapsulates or represents the security characteristics of a user. RACF assigns a UTOKEN to each user in the system.

utility In IBM MQ, a supplied set of programs that provide the system operator or system administrator with facilities in addition to those provided by the IBM MQ commands.

UTOKEN

See user token.

V

value The content of a data item. The value can be an integer, a string, or the handle of another data bag.

virtual method

In object-oriented programming, a method that exhibits polymorphism.

W

IBM WebSphere MQ

A previous name for IBM MQ.

wiretapping

The act of gaining access to information that is flowing along a wire or any other type of conductor used in communications. The objective of wiretapping is to gain unauthorized access to information without being detected.

X

X509 International Telecommunications Union standard for PKI. Specifies the format of the public key certificate and the public key cryptography.

XCF See cross-system coupling facility.

XID See transaction identifier.

X/Open XA

The X/Open Distributed Transaction Processing XA interface. A proposed standard for distributed transaction communication. The standard specifies a bidirectional interface between resource managers that provide access to shared resources within transactions, and between a transaction service that monitors and resolves transactions.

IBM MQ Technical overview

Use IBM MQ to connect your applications and manage the distribution of information across your organization.

IBM MQ enables programs to communicate with one another across a network of unlike components (processors, operating systems, subsystems, and communication protocols) using a consistent application programming interface. Applications designed and written using this interface are known as message queuing applications.

Use the following subtopics to find out about message queuing and other features provided by IBM MQ.

Related concepts:

“Introduction to IBM MQ” on page 1

You can use IBM MQ to enable applications to communicate at different times and in many diverse computing environments.

Related reference:

“Main features and benefits of message queuing” on page 149

This information highlights some features and benefits of message queuing. It describes features such as security and data integrity of message queuing.

Related information:

Planning an IBM MQ architecture

Finding product requirements and updated support information

Managed File Transfer

Introduction to message queuing

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

Applications designed and written using this interface are known as *message queuing* applications, because they use the *messaging* and *queuing* style:

- Messaging means that programs communicate by sending each other data in messages rather than calling each other directly.
- Queuing means that messages are placed on queues in storage, allowing programs to run independently of each other, at different speeds and times, in different locations, and without having a logical connection between them.

Message queuing has been used in data processing for many years. It is most commonly used today in electronic mail. Without queuing, sending an electronic message over long distances requires every node on the route to be available for forwarding messages, and the addressees to be logged on and conscious of the fact that you are trying to send them a message. In a queuing system, messages are stored at intermediate nodes until the system is ready to forward them. At their final destination they are stored in an electronic mailbox until the addressee is ready to read them.

Even so, many complex business transactions are processed today without queuing. In a large network, the system might be maintaining many thousands of connections in a ready-to-use state. If one part of the system suffers a problem, many parts of the system become unusable.

You can think of message queuing as being electronic mail for programs. In a message queuing environment, each program that makes up part of an application suite performs a well-defined,

self-contained function in response to a specific request. To communicate with another program, a program must put a message on a predefined queue. The other program retrieves the message from the queue, and processes the requests and information contained in the message. So message queuing is a style of program-to-program communication.

Queuing is the mechanism by which messages are held until an application is ready to process them. Queuing allows you to:

- Communicate between programs (which might each be running in different environments) without having to write the communication code.
- Select the order in which a program processes messages.
- Balance loads on a system by arranging for more than one program to service a queue when the number of messages exceeds a threshold.
- Increase the availability of your applications by arranging for an alternative system to service the queues if your primary system is unavailable.

What is a message queue?

A message queue, known simply as a 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.

Queues reside in, and are managed by, a queue manager, (see “Message queuing terminology” on page 151). The physical nature of a queue depends on the operating system on which the queue manager is running. A queue can either be a volatile buffer area in the memory of a computer, or a data set on a permanent storage device (such as a disk). The physical management of queues is the responsibility of the queue manager and is not made apparent to the participating application programs.

Programs access queues only through the external services of the queue manager. They can open a queue, put messages on it, get messages from it, and close the queue. They can also set, and inquire about, the attributes of queues.

Different styles of message queuing

Point-to-point

One message is placed on the queue and one application receives that message.

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

Publish/Subscribe

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

Publish/subscribe messaging allows you to decouple the provider of information from the consumers of that information. The sending application and receiving application do not need to know as much about each other for the information to be sent and received. For more information, see “Publish/subscribe messaging” on page 203.

Benefits of message queuing to the application designer and developer

IBM MQ allows application programs to use *message queuing* to participate in message-driven processing. Application programs can communicate across different platforms by using the appropriate message

queuing software products. For example, z/OS applications can communicate through IBM MQ for z/OS. The applications are shielded from the mechanics of the underlying communications. Some of the other benefits of message queuing are:

- You can design applications using small programs that you can share between many applications.
- You can quickly build new applications by reusing these building blocks.
- Applications written to use message queuing techniques are not affected by changes in the way that queue managers work.
- You do not need to use any communication protocols. The queue manager deals with all aspects of communication for you.
- Programs that receive messages need not be running at the time that messages are sent to them. The messages are retained on queues.

Designers can reduce the cost of their applications because development is faster, fewer developers are needed, and demands on programming skill are lower than those for applications that do not use message queuing.

IBM MQ implements a common application programming interface known as the *message queue interface* (or MQI) wherever the applications run. This makes it easier for you to port application programs from one platform to another.

For details about the MQI, see The Message Queue Interface overview.

Main features and benefits of message queuing

This information highlights some features and benefits of message queuing. It describes features such as security and data integrity of message queuing.

The main features of applications that use message queuing techniques are:

- There are no direct connections between programs.
- Communication between programs can be independent of time.
- Work can be carried out by small, self-contained programs.
- Communication can be driven by events.
- Applications can assign a priority to a message.
- Security.
- Data integrity.
- Recovery support.

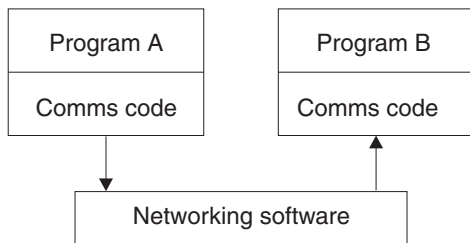
No direct connections between programs

Message queuing is a technique for indirect program-to-program communication. It can be used within any application where programs communicate with each other. Communication occurs by one program putting messages on a queue (owned by a queue manager) and another program getting the messages from the queue.

Programs can get messages that were put on a queue by other programs. The other programs can be connected to the same queue manager as the receiving program, or to another queue manager. This other queue manager might be on another system, a different computer system, or even within a different business or enterprise.

There are no physical connections between programs that communicate using message queues. A program sends messages to a queue owned by a queue manager, and another program retrieves messages from the queue (see Figure 1 on page 150).

Traditional communication between programs



Communication by message queuing

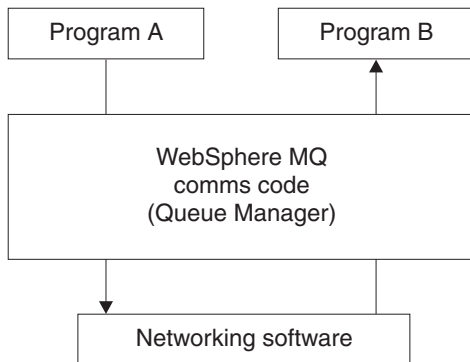


Figure 1. Message queuing compared with traditional communication

As with electronic mail, the individual messages that are part of a transaction travel through a network on a store-and-forward basis. If a link between nodes fails, the message is kept until the link is restored, or the operator or program redirects the message.

The mechanism by which a message moves from queue to queue is hidden from the programs. Therefore the programs are simpler.

Time-independent communication

Programs requesting others to do work do not have to wait for the reply to a request. They can do other work, and process the reply either when it arrives or at a later time. When writing a messaging application, you need not know (or be concerned) when a program sends a message, or when the target is able to receive the message. The message is not lost; it is retained by the queue manager until the target is ready to process it. The message stays on the queue until it is removed by a program. This means that the sending and receiving application programs are decoupled; the sender can continue processing without waiting for the receiver to acknowledge receipt of the message. The target application does not even have to be running when the message is sent. It can retrieve the message after it is has been started.

Small programs

Message queuing allows you to use the advantages of using small, self-contained programs. Instead of a single, large program performing all the parts of a job sequentially, you can spread the job over several smaller, independent programs. The requesting program sends messages to each of the separate programs, asking them to perform their function; when each program is complete, the results are sent back as one or more messages.

Message-driven processing

When messages arrive on a queue, they can automatically start an application using *triggering*. If necessary, the applications can be stopped when the message (or messages) have been processed.

Event-driven processing

Programs can be controlled according to the state of queues. For example, you can arrange for a

program to start as soon as a message arrives on a queue, or you can specify that the program does not start until there are, for example, 10 messages above a certain priority on the queue, or 10 messages of any priority on the queue.

Message priority

A program can assign a priority to a message when it puts the message on a queue. This determines the position in the queue at which the new message is added.

Programs can get messages from a queue either in the order in which the messages are in the queue, or by getting a specific message. (A program might want to get a specific message if it is looking for the reply to a request that it sent earlier.)

Security

Security facilities are provided, including authentication of applications when they use a queue manager, authorization checks when they use resources such as a queue on the queue manager, and encryption of message data as it travels over the network, and as it resides on queues. For more information about security, see Security Overview.

Data integrity

Data integrity is provided by units of work. The synchronization of the start and end of units of work is fully supported as an option on each MQGET or MQPUT, allowing the results of the unit of work to be committed or rolled back. Sync point support operates either internally or externally to IBM MQ depending on the form of sync point coordination selected for the application.

Recovery support



For recovery to be possible, all persistent IBM MQ updates are logged. If recovery is necessary, all persistent messages are restored, all in-flight transactions are rolled back, and any sync point commit and backouts are handled in the normal way of the sync point manager in control. For more information about persistent messages, see Message persistence.

Note: When considering IBM MQ clients and servers, you do not have to change a server application to support additional IBM MQ MQI clients on new platforms. Similarly, the IBM MQ MQI client can, without change, function with additional types of servers.

Message queuing terminology

This information gives an insight into some terms used in message queuing.

They include:

- Message
- Message descriptor
- Queue
- Queue manager
- Channels
- Message channel agent
- Cluster
-  Shared queue
-  Queue-sharing group
- Intra-group queuing
- IBM MQ MQI client
- Point-to-point
- Publish/subscribe
- Topic
- Subscription

Message

In message queuing, a message is a collection of data sent by one program and intended for another program. See IBM MQ messages. For information about message types, see Types of message.

Message descriptor

An IBM MQ message consists of control information and application data.

The control information is defined in a message descriptor structure (MQMD) and contains such things as:

- The type of the message
- An identifier for the message
- The priority for delivery of the message

The structure and content of the application data is determined by the participating programs, not by IBM MQ.

Queue

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

Queue manager

A *queue manager* is a system program that provides queuing services to applications.

It provides an application programming interface so that programs can put messages on, and get messages from, queues. A queue manager provides additional functions so that administrators can create new queues, alter the properties of existing queues, and control the operation of the queue manager.

For IBM MQ message queuing services to be available on a system, there must be a queue manager running. You can have more than one queue manager running on a single system (for example, to separate a test system from a *live* system). To an application, each queue manager is identified by a *connection handle* (*Hconn*).

Many different applications can use the services of the queue manager at the same time and these applications can be entirely unrelated. For a program to use the services of a queue manager, it must establish a connection to that queue manager.

For applications to send messages to applications that are connected to other queue managers, the queue managers must be able to communicate among themselves. IBM MQ implements a *store-and-forward* protocol to ensure the safe delivery of messages between such applications.

Channels

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.

Message channel agent

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 and move messages from one queue manager to another.

Cluster

A *cluster* is a network of queue managers that are logically associated in some way.

In an IBM MQ network using distributed queuing without clustering, every queue manager is independent. If one queue manager needs to send messages to another, it must have defined a transmission queue and a channel to the remote queue manager.

There are two different reasons for using clusters: to reduce system administration and to improve availability and workload balancing.

As soon as you establish even the smallest cluster, you benefit from simplified system administration. Queue managers that are part of a cluster need fewer definitions and so the risk of making an error in your definitions is reduced.

For more information about clustering, see [Clusters](#).

z/OS Shared queue

A *shared queue* is a type of local queue with messages that can be accessed by one or more queue managers that are in a sysplex. This is not the same as a queue being shared by more than one application, using the same queue manager. This applies only to IBM MQ for z/OS.

z/OS Queue-sharing group

The queue managers that can access the same set of shared queues form a group called a *queue-sharing group* (QSG). They communicate with each other with a coupling facility (CF) that stores the shared queues. This applies only to IBM MQ for z/OS. See “Shared queues and queue-sharing groups” on page 322 for more information about queue-sharing groups.








z/OS Intra-group queuing

Queue managers in a queue-sharing group can communicate using normal channels or you can use a technique called *intra-group queuing* (IGQ), which lets you perform fast message transfer without defining channels. This applies only to IBM MQ for z/OS.

IBM MQ MQI client

IBM MQ MQI *clients* are independently installable components of IBM MQ. An MQI client allows you to run IBM MQ applications with a communications protocol, to interact with one or more Message Queue Interface (MQI) servers on other platforms and to connect to their queue managers.

For full details on how to install and use IBM MQ MQI client components, see the following topics:

-  Installing an IBM MQ client on AIX
-  Installing IBM MQ on HP Integrity NonStop Server
-  Installing an IBM MQ client on HP-UX
-  Installing an IBM MQ client on Linux
-  Installing an IBM MQ client on Solaris
-  Installing an IBM MQ client on Windows
-  Installing an IBM MQ client on IBM i

and Configuring connections between the server and client.

Point-to-point messaging

In point-to-point messaging, each message travels from one producing application to one consuming application. Messages are transferred through the producing application putting messages onto a queue and the consuming application gets them from that queue.

Publish/subscribe messaging

In publish/subscribe messaging, a copy of each message published by a publishing application is delivered to every interested application. There might be many, one or no interested applications.

In publish/subscribe an interested application is known as a subscriber and the messages are queued on a queue identified by a subscription. For more information, see “Publish/subscribe messaging” on page 203.

Topic

A topic is a character string that describes the subject of the information that is published in a publish/subscribe message.

Topics are key to the successful delivery of messages in a publish/subscribe system. Instead of including a specific destination address in each message, a publisher assigns a topic to each message. The queue manager matches the topic with a list of subscribers who have subscribed to that topic, and delivers the message to each of those subscribers.

Subscription

A publish/subscribe application can register an interest in messages about specific topics. When an application does this it is known as a subscriber and the term subscription defines how matching messages are queued for processing.

A subscription contains information about the identity of the subscriber and the identity of the destination queue on to which publications are to be placed. It also contains information about how a publication is to be placed on the destination queue.

Messages and queues

Messages and queues are the basic components of a message queuing system.

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.

An IBM MQ message is made up of:

- *The application data.* The content and structure of the application data is defined by the application programs that use it.
- *A message descriptor.* The message descriptor identifies the message and contains additional control information, such as the type of message and the priority assigned to the message by the sending application.

The format of the message descriptor is defined by IBM MQ. For a complete description of the message descriptor, see MQMD - Message descriptor.

- *Message properties.* Meta-data about the message. The content of the message properties are defined by the application programs that use them. For more information, see Message properties.

Message lengths

The default maximum message length is 4 MB, although you can increase this to a maximum length of 100 MB (where 1 MB equals 1 048 576 bytes). In practice, the message length might be limited by:

- The maximum message length defined for the receiving queue
- The maximum message length defined for the queue manager
- The maximum message length defined by the queue
- The maximum message length defined by either the sending or receiving application
- The amount of storage available for the message

It might take several messages to send all the information that an application requires.

How do applications send and receive messages?

Application programs send and receive messages using **MQI calls**.

For example, to put a message onto a queue, an application:

1. Opens the required queue by issuing an MQI MQOPEN call
2. Issues an MQI MQPUT call to put the message onto the queue

Another application can retrieve the message from the same queue by issuing an MQI MQGET call

For more information about MQI calls, see MQI calls.

What is a queue?

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

Each queue is owned by a *queue manager*. The queue manager is responsible for maintaining the queues 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.

Predefined queues and dynamic queues

Queues can be characterized by the way they are created:

- **Predefined queues** are created by an administrator using the appropriate MQSC or PCF commands. Predefined queues are permanent; they exist independently of the applications that use them and survive IBM MQ restarts.
- **Dynamic queues** are created when an application issues an MQOPEN request specifying the name of a *model queue*. The queue created is based on a *template queue definition*, which is called a model queue. You can create a model queue using the MQSC command DEFINE QMODEL. The attributes of a model queue (for example, the maximum number of messages that can be stored on it) are inherited by any dynamic queue that is created from it.

Model queues have an attribute that specifies whether the dynamic queue is to be permanent or temporary. Permanent queues survive application and queue manager restarts; temporary queues are lost on restart.

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.

The MQGET request from the application determines the method used.

IBM MQ objects

Queue managers define the properties of IBM MQ objects. The values of these properties affect the way in which IBM MQ processes these objects. You create and manage objects using IBM MQ commands and interfaces. From your applications, you use the Message Queue Interface (MQI) to control objects. Objects are identified by an IBM MQ *object descriptor* (MQOD) when addressed from a program.




The administration of objects includes the following tasks:

- Starting and stopping queue managers.
- Creating objects, particularly queues, for applications.
- Displaying or altering the attributes of objects.
- Deleting objects.
- Working with channels to create communication paths to queue managers on other (remote) systems.
- Creating *clusters* of queue managers to simplify the overall administration process, and to balance workload.


With the exception of dynamic queues, objects must be defined to the queue manager before you can work with them.


When you use an IBM MQ command to carry out an object administration operation, the queue manager checks that you have the required level of authority to perform the operation. Similarly, when an application uses the MQOPEN call to open an object, the queue manager checks that the application has the required level of authority before it allows access to that object. The checks are made on the name of the object being opened.

You can define and manage objects by using the following methods:

- The PCF commands described in Programmable command formats reference and Automating administration tasks
- The MQSC commands described in The MQSC commands
-  The IBM MQ for z/OS operations and control panels, described in Operating IBM MQ for z/OS
-   The IBM MQ Explorer (Windows and Linux for Intel systems only). For more information, see Introduction to MQ Explorer.

You can also manage objects by using the following methods:

- Control commands, which are typed in from a keyboard. See The control commands.
- IBM MQ Administration Interface (MQAI) calls in a program. See IBM MQ Administration Interface (MQAI).
-  IBM MQ for Windows only:
 - MQAI Component Object Model (COM) calls in a program
 - The Windows Default Configuration Application

 For sequences of IBM MQ for z/OS commands that you use regularly, you can write administration programs that create messages containing commands and that put these messages on the system-command input queue. The queue manager processes the messages on this queue in the same way that it processes commands entered from the command line or from the operations and control panels. This technique is described in the Writing programs to administer IBM MQ, and demonstrated in the Mail Manager sample application delivered with IBM MQ for z/OS. For a description of this sample, see Sample programs for IBM MQ for z/OS .

IBM i

For sequences of IBM MQ for IBM i commands that you use regularly you can write CL programs. For more information, see *Managing IBM MQ for IBM i using CL commands*.

ULW

For sequences of IBM MQ commands on UNIX, Linux, and Windows, you can use the MQSC facility to run a series of commands held in a file. For more information, see *Administration using MQSC commands*.

Object types

Many of the administration tasks involve manipulating various types of IBM MQ *objects*.

For information about naming IBM MQ objects, see “Naming IBM MQ objects” on page 175.

For information about the default objects created on a queue manager, see “System default objects” on page 182.

For information about the different types of IBM MQ objects, see the following subtopics:

Related concepts:

“Introduction to message queuing” on page 147

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

“Object attributes” on page 181

The properties of an object are defined by its attributes. Some you can specify, others you can only view.

Related information:

The MQSC commands

Queues

Introduction to IBM MQ queues and queue attributes.

An IBM MQ *queue* is a named object on which applications can put messages, and from which applications can get messages.

Messages are stored on a queue, so that if the putting application is expecting a reply to its message, it is free to do other work while waiting for that reply. Applications access a queue by using the Message Queue Interface (MQI), described in *The Message Queue Interface overview*.

Before a message can be put on a queue, the queue must have already been created. A queue is owned by a queue manager, and that queue manager can own many queues. However, each queue must have a name that is unique within that queue manager.

A queue is maintained through a queue manager. In most cases, each queue is physically managed by its queue manager but this is not apparent to an application program. IBM MQ for z/OS shared queues can be managed by any queue manager in the queue-sharing group.

To create a queue you can use IBM MQ commands (MQSC), PCF commands, or platform-specific interfaces. For example, the IBM MQ for z/OS operations and control panels are platform-specific.

You can create local queues for temporary jobs *dynamically* from your application. For example, you can create *reply-to* queues (which are not needed after an application ends). For more information, see “Dynamic and Model queues” on page 163.

Before using a queue, you must open the queue, specifying what you want to do with it. For example, you can open a queue for:

- Browsing messages only (not retrieving them)
- Retrieving messages (and either sharing the access with other programs, or with exclusive access)
- Putting messages on the queue
- Inquiring about the attributes of the queue
- Setting the attributes of the queue

For a complete list of the options that you can specify when you open a queue, see MQOPEN - Open object.

Attributes of queues

Some of the attributes of a queue are specified when the queue is defined, and cannot be changed afterward (for example, the type of the queue). Other attributes of queues can be grouped into those that can be changed:

- By the queue manager during the processing of the queue (for example, the current depth of a queue)
- Only by commands (for example, the text description of the queue)
- By applications, using the MQSET call (for example, whether put operations are allowed on the queue)

You can find the values of all the attributes using the MQINQ call.

The attributes that are common to more than one type of queue are:

QName

Name of the queue.

QType

Type of the queue.

QDesc

Text description of the queue.

InhibitGet

Whether programs are allowed to get messages from the queue. However, you can never get messages from remote queues.

InhibitPut

Whether programs are allowed to put messages on the queue.

DefPriority

Default priority for messages put on the queue.

DefPersistence

Default persistence for messages put on the queue

Scope

Controls whether an entry for this queue also exists in a name service.

 The **Scope** attribute is not supported on z/OS

For a full description of these attributes, see Attributes for queues.

Related concepts:

▶ z/OS Shared queues

A shared queue is a type of local queue. The messages on that queue can be accessed by one or more queue managers that are in a sysplex.

“Cluster queues” on page 199

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

“Dead-letter queues” on page 190

The dead-letter queue (or undelivered-message queue) is the queue to which messages are sent if they cannot be routed to their correct destination. Each queue manager typically has a dead-letter queue.

Related reference:

“Comparison between shared queues and cluster queues” on page 199

This information is designed to help you compare shared queues and cluster queues, and decide which might be more suitable for your system.

Related information:

The MQSC commands

Developing applications reference

Local queues:

Transmission, initiation, dead-letter, command, default, channel, and event queues are types of local queue.

A queue is known to a program as *local* if it is owned by the queue manager to which the program is connected. You can get messages from, and put messages on, local queues.

The queue definition object holds the definition information of the queue as well as the physical messages put on the queue.

Each queue manager can have some local queues that it uses for special purposes:

Transmission queues

When an application sends a message to a remote queue, the local queue manager stores the message in a special local queue, called a *transmission queue*.

A *message channel agent* is a channel program is associated with the transmission queue and it delivers the message to its next destination. The next destination is the queue manager to which the message channel is connected. It is not necessarily the same queue manager as the final destination of the message. When the message is delivered to its next destination, it is deleted from the transmission queue. The message might have to pass through many queue managers on its journey to its final destination. You must define a transmission queue at each queue manager along the route, each holding messages waiting to be transmitted to the next destination. A normal transmission queue holds messages for the next destination, although the messages might have different eventual destinations. A cluster transmission queue holds messages for multiple destinations. The *correlID* of each message identifies the channel that the message is placed on to transfer it to its next destination.

You can define several transmission queues at a queue manager. You might define several transmission queues for the same destination, with each one being used for a different class of service. For example, you might want to create different transmission queues for small messages and large messages going to the same destination. You can then transfer the messages using different messages channels, so that the large messages do not hold up the smaller messages. All messages to cluster queues or cluster topics are placed on the single cluster transmission queue `SYSTEM.CLUSTER.TRANSMIT.QUEUE`, by default. As an option, you can change the default, and separate the message traffic going to different cluster queue managers onto different cluster

transmission queues. If you set the queue manager attribute DEFCLXQ to CHANNEL, each cluster-sender channel creates a separate cluster transmission queue. As an alternative, you can manually define cluster transmission queues for cluster-sender channels to use.

Transmission queues can trigger a message channel agent to send messages onward; see Starting IBM MQ applications using triggers.

z/OS On IBM MQ for z/OS, if you are using intra-group queuing, the transmission queue is serviced by an *intra-group queuing agent*. A shared transmission queue is used when using intra-group queuing on IBM MQ for z/OS.

Initiation queues

An *initiation queue* is a local queue on which the queue manager puts a trigger message when a trigger event occurs on an application queue.

A trigger event is an event that is intended to cause a program to start processing a queue. For example, an event might be more than 10 messages arriving. For more information about how triggering works, see Starting IBM MQ applications using triggers.

Dead-letter (undelivered message) queue

A *dead-letter (undelivered message) queue* is a local queue on which the queue manager puts messages that it cannot deliver.

When the queue manager puts a message on the dead-letter queue, it adds a header to the message. The header information includes the reason that the queue manager put the message on the dead-letter queue. It also contains the destination of the original message, the date, and the time that the queue manager put the message on the dead-letter queue.

Applications can also use the queue for messages that they cannot deliver. For more information, see Using the dead-letter (undelivered message) queue.

System command queue

The *system command queue* is a queue to which suitably authorized applications can send IBM MQ commands. These queues receive the PCF, MQSC, and CL commands, as supported on your platform, in readiness for the queue manager to action them.

z/OS On IBM MQ for z/OS the queue is called SYSTEM.COMMAND.INPUT ; on other platforms it is called SYSTEM.ADMIN.COMMAND.QUEUE. The commands accepted vary by platform. See Programmable command formats reference for details.

System default queues

The *system default queues* contain the initial definitions of the queues for your system. When you create a queue definition, the queue manager copies the definition from the appropriate system default queue. Creating a queue definition is different from creating a dynamic queue. The definition of the dynamic queue is based upon the model queue you choose as the template for the dynamic queue.

Event queues

Event queues hold event messages. These messages are reported by the queue manager or a channel.

Remote queues:

To a program, a queue is *remote* if it is owned by a different queue manager to the one to which the program is connected.


Where a communication link has been established, a program can send a message to a remote queue. A program can never get a message from a remote queue.

The queue definition object, created when you define a remote queue, only holds the information necessary for the local queue manager to locate the queue to which you want your message to go. This object is known as the *local definition of a remote queue*. All the attributes of the remote queue are held by the queue manager that owns it, because it is a local queue to that queue manager.

When opening a remote queue, to identify the queue you must specify either of the following:

- The name of the local definition that defines the remote queue. From the viewpoint of an application, this is the same as opening a local queue. An application does not need to know if a queue is local or remote.

To create a local definition of a remote queue on all platforms except IBM i, use the DEFINE QREMOTE command.

 On IBM i, use the CRTMQMQ command.

- The name of the remote queue manager and the name of the queue as it is known to that remote queue manager.

Local definitions of remote queues have three attributes in addition to the common attributes described in “Attributes of queues” on page 158. These three attributes are:

RemoteQName

The name that the queue's owning queue manager knows it by.

RemoteQMgrName

The name of the owning queue manager.

XmitQName

The name of the local transmission queue that is used when forwarding messages to other queue managers.

For more information about these attributes, see Attributes for queues.


If you use the MQINQ call against the local definition of a remote queue, the queue manager returns the attributes of the local definition only, that is the remote queue name, the remote queue manager name, and the transmission queue name, not the attributes of the matching local queue in the remote system.

See also Transmission queues.

Alias queues:

An *alias queue* is an IBM MQ object that you can use to access another queue or a topic. This means that more than one program can work with the same queue, accessing it using different names.

The queue resulting from the resolution of an alias name, known as the base queue, can be any of the following types of queues, as supported by the platform:

- A local queue
- The local definition of a remote queue.
-  A shared queue, which is a type of local queue only available on IBM MQ for z/OS.
- A predefined queue
- A dynamic queue

An alias name can also resolve to a topic. If an application currently puts messages onto a queue, it can be made to publish to a topic by making the queue name an alias for the topic. No change to the application code is necessary.

Note: An alias cannot directly resolve to another alias on the same queue manager.

An example of the use of alias queues is for a system administrator to give different access authorities to the base queue name (that is, the queue to which the alias resolves) and to the alias queue name. This means that a program or user can be authorized to use the alias queue, but not the base queue.

Alternatively, authorization can be set to inhibit put operations for the alias name, but allow them for the base queue.

In some applications, the use of alias queues means that system administrators can easily change the definition of an alias queue object without having to get the application changed.

IBM MQ makes authorization checks against the alias name when programs try to use that name. It does not check that the program is authorized to access the name to which the alias resolves. A program can therefore be authorized to access an alias queue name, but not the resolved queue name.

In addition to the general queue attributes described in “Queues” on page 157, alias queues have a **BaseQName** attribute. This is the name of the base queue to which the alias name resolves. For a fuller description of this attribute, see BaseQName (MQCHAR48).

The *InhibitGet* and **InhibitPut** attributes (see “Queues” on page 157) of alias queues belong to the alias name. For example, if the alias-queue name ALIAS1 resolves to the base-queue name BASE, inhibitions on ALIAS1 affect ALIAS1 only and BASE is not inhibited. However, inhibitions on BASE also affect ALIAS1.

The *DefPriority* and **DefPersistence** attributes also belong to the alias name. So, for example, you can assign different default priorities to different aliases of the same base queue. Also, you can change these priorities without having to change the applications that use the aliases.


Dynamic and Model queues:

This information provides an insight into dynamic queues, properties of temporary and permanent dynamic queues, uses of dynamic queues, some considerations when using dynamic queues, and model queues.

When an application program issues an MQOPEN call to open a model queue, the queue manager dynamically creates an instance of a local queue with the same attributes as the model queue. Depending on the value of the *DefinitionType* field of the model queue, the queue manager creates either a temporary or permanent dynamic queue (See Creating dynamic queues).

Properties of temporary dynamic queues

Temporary dynamic queues have the following properties:

-  They cannot be shared queues, accessible from queue managers in a queue-sharing group.
Note that queue-sharing groups are only available on IBM MQ for z/OS.
- They hold nonpersistent messages only.
- They are unrecoverable.
- They are deleted when the queue manager is started.
- They are deleted when the application that issued the MQOPEN call that created the queue closes the queue or terminates.
 - If there are any committed messages on the queue, they are deleted.
 - If there are any uncommitted MQGET, MQPUT, or MQPUT1 calls outstanding against the queue at this time, the queue is marked as being logically deleted, and is only physically deleted (after these calls have been committed) as part of close processing, or when the application terminates.
 - If the queue is in use at this time (by the creating, or another application), the queue is marked as being logically deleted, and is only physically deleted when closed by the last application using the queue.
 - Attempts to access a logically deleted queue (other than to close it) fail with reason code MQRC_Q_DELETED.
 - MQCO_NONE, MQCO_DELETE and MQCO_DELETE_PURGE are all treated as MQCO_NONE when specified on an MQCLOSE call for the corresponding MQOPEN call that created the queue.

Properties of permanent dynamic queues

Permanent dynamic queues have the following properties:

- They hold persistent or nonpersistent messages.
- They are recoverable in the event of system failures.
- They are deleted when an application (not necessarily the one that issued the MQOPEN call that created the queue) successfully closes the queue using the MQCO_DELETE or MQCO_DELETE_PURGE option.
 - A close request with the MQCO_DELETE option fails if there are any messages (committed or uncommitted) still on the queue. A close request with the MQCO_DELETE_PURGE option succeeds even if there are committed messages on the queue (the messages being deleted as part of the close), but fails if there are any uncommitted MQGET, MQPUT, or MQPUT1 calls outstanding against the queue.
 - If the delete request is successful, but the queue happens to be in use (by the creating, or another application), the queue is marked as being logically deleted and is only physically deleted when closed by the last application using the queue.

- They are not deleted if closed by an application that is not authorized to delete the queue, unless the closing application issued the MQOPEN call that created the queue. Authorization checks are performed against the user identifier (or alternate user identifier if MQOO_ALTERNATE_USER_AUTHORITY was specified) that was used to validate the corresponding MQOPEN call.
- They can be deleted in the same way as a normal queue.

Uses of dynamic queues

You can use dynamic queues for:

- Applications that do not require queues to be retained after the application has terminated.
- Applications that require replies to messages to be processed by another application. Such applications can dynamically create a reply-to queue by opening a model queue. For example, a client application can:
 1. Create a dynamic queue.
 2. Supply its name in the **ReplyToQ** field of the message descriptor structure of the request message.
 3. Place the request on a queue being processed by a server.

The server can then place the reply message on the reply-to queue. Finally, the client could process the reply, and close the reply-to queue with the delete option.

Considerations when using dynamic queues

Consider the following points when using dynamic queues:

- In a client-server model, each client must create and use its own dynamic reply-to queue. If a dynamic reply-to queue is shared between more than one client, deleting the reply-to queue might be delayed because there is uncommitted activity outstanding against the queue, or because the queue is in use by another client. Additionally, the queue might be marked as being logically deleted, and inaccessible for subsequent API requests (other than MQCLOSE).
- If your application environment requires that dynamic queues must be shared between applications, ensure that the queue is only closed (with the delete option) when all activity against the queue has been committed. This should be by the last user. This ensures that deletion of the queue is not delayed, and minimizes the period that the queue is inaccessible because it has been marked as being logically deleted.

Model queues

A *model queue* is a template of a queue definition that you use when creating a dynamic queue.

You can create a local queue dynamically from an IBM MQ program, naming the model queue that you want to use as the template for the queue attributes. At that point you can change some attributes of the new queue. However, you cannot change the **DefinitionType**. If, for example, you require a permanent queue, select a model queue with the definition type set to permanent. Some conversational applications can use dynamic queues to hold replies to their queries because they probably do not need to maintain these queues after they have processed the replies.

You specify the name of a model queue in the *object descriptor* (MQOD) of your MQOPEN call. Using the attributes of the model queue, the queue manager dynamically creates a local queue for you.

You can specify a name (in full) for the dynamic queue, or the stem of a name (for example, ABC) and let the queue manager add a unique part to this, or you can let the queue manager assign a complete unique name for you. If the queue manager assigns the name, it puts it in the MQOD structure.

You cannot issue an MQPUT1 call directly to a model queue, but you can issue an MQPUT1 to the dynamic queue that has been created by opening a model queue.

MQSET and MQINQ cannot be issued against a model queue. Opening a model queue with MQOO_INQUIRE or MQOO_SET results in subsequent MQINQ and MQSET calls being made against the dynamically created queue.

The attributes of a model queue are a subset of those of a local queue. For a fuller description, see [Attributes for queues](#).

Ways of defining queues:

You can create a queue by using IBM MQ commands (MQSC), PCF commands, or by using platform-specific interfaces.

You can define queues to IBM MQ by using the MQSC DEFINE command or the PCF Create Queue command. The commands specify the type of queue and its attributes. For example, a local queue object has attributes that specify what happens when applications reference that queue in MQI calls. Examples of attributes are:

- Whether applications can retrieve messages from the queue (GET enabled)
- Whether applications can put messages on the queue (PUT enabled)
- Whether access to the queue is exclusive to one application or shared between applications
- The maximum number of messages that can be stored on the queue at the same time (maximum queue depth)
- The maximum length of messages that can be put on the queue

There are also various platform-specific interfaces that you can use to define queues. For links to more information, see the *Related information*.

Related information:

[Administration using MQSC commands](#)

[Automating administration using PCF commands](#)

[Working with queues in the IBM MQ Console](#)

[Creating and configuring queue managers and objects with MQ Explorer](#)

[Managing IBM MQ for IBM i using CL commands](#)

[Issuing commands to IBM MQ for z/OS](#)

Queues used for specific purposes by IBM MQ:

IBM MQ uses some local queues for specific purposes related to its operation.

You must define these queues before IBM MQ can use them.

Initiation queues

Initiation queues are queues that are used in triggering. A queue manager puts a trigger message on an initiation queue when a trigger event occurs. A trigger event is a logical combination of conditions that is detected by a queue manager. For example, a trigger event might be generated when the number of messages on a queue reaches a predefined depth. This event causes the queue manager to put a trigger message on a specified initiation queue. This trigger message is retrieved by a *trigger monitor*, a special application that monitors an initiation queue. The trigger monitor then starts the application program that was specified in the trigger message.

If a queue manager is to use triggering, at least one initiation queue must be defined for that queue manager. See [Managing objects for triggering](#), [runmqtrm](#), and [Starting IBM MQ applications using triggers](#)

Transmission queues

Transmission queues are queues that temporarily store messages that are destined for a remote queue manager. You must define at least one transmission queue for each remote queue manager

to which the local queue manager is to send messages directly. These queues are also used in remote administration; see Remote administration from a local queue manager. For information about the use of transmission queues in distributed queuing, see IBM MQ distributed queuing techniques.

Each queue manager can have a default transmission queue. If a queue manager that is not part of a cluster puts a message onto a remote queue, the default action is to use the default transmission queue. If there is a transmission queue with the same name as the destination queue manager, the message is placed on that transmission queue. If there is a queue manager alias definition, in which the **RQMNAME** parameter matches the destination queue manager, and the **XMITQ** parameter is specified, the message is placed on the transmission queue named by **XMITQ**. If there is no **XMITQ** parameter, the message is placed on the local queue named in the message.

Cluster transmission queues

Each queue manager within a cluster has a cluster transmission queue called `SYSTEM.CLUSTER.TRANSMIT.QUEUE`, and a model cluster transmission queue, `SYSTEM.CLUSTER.TRANSMIT.MODEL.QUEUE`. Definitions of these queues are created by default when you define a queue manager. If the queue manager attribute, **DEFCLXQ**, is set to `CHANNEL`, a permanent dynamic cluster transmission queue is automatically created for each cluster-sender channel that is created. The queues are called `SYSTEM.CLUSTER.TRANSMIT.ChannelName`. You can also define cluster transmission queues manually.

A queue manager that is part of the cluster sends messages on one of these queues to other queue managers that are in the same cluster.

During name resolution, a cluster transmission queue takes precedence over the default transmission queue, and a specific cluster transmission queue takes precedence over `SYSTEM.CLUSTER.TRANSMIT.QUEUE`.

Dead-letter queues

A dead-letter (undelivered-message) queue is a queue that stores messages that cannot be routed to their correct destinations. A message cannot be routed when, for example, the destination queue is full. The supplied dead-letter queue is called `SYSTEM.DEAD.LETTER.QUEUE`.

For distributed queuing, define a dead-letter queue on each queue manager involved.

Command queues

The command queue, `SYSTEM.ADMIN.COMMAND.QUEUE`, is a local queue to which suitably authorized applications can send MQSC commands for processing. These commands are then retrieved by an IBM MQ component called the command server. 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.

A command queue is created automatically for each queue manager when that queue manager is created.

Reply-to queues

When an application sends a request message, the application that receives the message can send back a reply message to the sending application. This message is put on a queue, called a reply-to queue, which is normally a local queue to the sending application. The name of the reply-to queue is specified by the sending application as part of the message descriptor.

Event queues

Instrumentation events can be used to monitor queue managers independently of MQI applications.

When an instrumentation event occurs, the queue manager puts an event message on an event queue. This message can then be read by a monitoring application, which might inform an administrator or initiate some remedial action if the event indicates a problem.

Note: Trigger events are different from instrumentation events. Trigger events are not caused by the same conditions, and do not generate event messages.

For more information about instrumentation events, see Instrumentation events.

IBM MQ queue managers

An introduction to *queue managers* and the queuing services that they provide to applications.

A program must have a connection to a queue manager before it can use the services of that queue manager. A program can make this connection explicitly (using the MQCONN or MQCONNX call), or the connection might be made implicitly (this depends on the platform and the environment in which the program is running).

Queue managers provide queuing services to applications, and manages the queues that belong to them. A queue manager ensures the following actions:

- 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.
- Messages are put on the correct queue, as requested by the application making the MQPUT call. The application is informed if this cannot be done, and an appropriate reason code is given.

Each queue belongs to a single queue manager and is said to be a *local queue* to that queue manager. The queue manager to which an application is connected is said to be the *local queue manager* for that application. For the application, the queues that belong to its local queue manager are local queues.


A *remote queue* is a queue that belongs to another queue manager. A *remote queue manager* is any queue manager other than the local queue manager. A remote queue manager can exist on a remote machine across the network, or might exist on the same machine as the local queue manager. IBM MQ supports multiple queue managers on the same machine.

A queue manager object can be used in some MQI calls. For example, you can inquire about the attributes of the queue manager object using the MQI call MQINQ.

Attributes of queue managers


Associated with each queue manager is a set of attributes (or properties) that define its characteristics. Some of the attributes of a queue manager are fixed when it is created; you can change others using the IBM MQ commands. You can inquire about the values of all the attributes, except those used for Transport Layer Security (TLS) encryption, using the MQINQ call.

The fixed attributes include:

- The name of the queue manager
- The platform on which the queue manager runs (for example, Windows)
- The level of system control commands that the queue manager supports
- The maximum priority that you can assign to messages processed by the queue manager
- The name of the queue to which programs can send IBM MQ commands
- The maximum length of messages the queue manager can process  (fixed only in IBM MQ for z/OS)
- Whether the queue manager supports syncpointing when programs put and get messages

The *changeable* attributes include:

- A text description of the queue manager

- The identifier of the character set the queue manager uses for character strings when it processes MQI calls
- The time interval that the queue manager uses to restrict the number of trigger messages
-  The time interval that the queue manager uses to determine how often queues are to be scanned for expired messages (IBM MQ for z/OS only)
- The name of the queue manager's dead-letter (undelivered message) queue
- The name of the queue manager's default transmission queue
- The maximum number of open handles for any one connection
- The enabling and disabling of various categories of event reporting
- The maximum number of uncommitted messages within a unit of work

Queue managers and workload management


You can set up a cluster of queue managers that has more than one definition for the same queue (for example, the queue managers in the cluster could be clones of each other). Messages for a particular queue can be handled by any queue manager that hosts an instance of the queue. A workload-management algorithm decides which queue manager handles the message and so spreads the workload between your queue managers; see *The cluster workload management algorithm* for further information.

Process definitions

Process definition objects allow applications to be started without the need for operator intervention by defining the attributes of the application for use by the queue manager.

The process definition object defines an application that starts in response to a trigger event on an IBM MQ queue manager. The process definition attributes include the application ID, the application type, and data specific to the application. For more information, see *Initiation queues* in “Queues used for specific purposes by IBM MQ” on page 165.

To allow an application to be started without the need for operator intervention, as described in *Starting IBM MQ applications using triggers*, the attributes of the application must be known to the queue manager. These attributes are defined in a *process definition object*.

The **ProcessName** attribute is fixed when the object is created. However, you can change other attributes by using the IBM MQ commands.  Alternatively, on z/OS, you can use the IBM MQ for z/OS operations and control panels.

You can inquire about the values of all the attributes by using MQINQ - Inquire object attributes.

Related information:

Attributes for process definitions

Namelists

A *namelist* is an IBM MQ object that contains a list of cluster names, queue names or authentication information object names. In a cluster, it can be used to identify a list of clusters for which the queue manager holds the repositories.

A namelist is an IBM MQ object that contains a list of other IBM MQ objects. Typically, namelists are used by applications such as trigger monitors, where they are used to identify a group of queues. The advantage of using a namelist is that it is maintained independently of applications; it can be updated without stopping any of the applications that use it. Also, if one application fails, the namelist is not affected and other applications can continue using it.

Namelists are also used with queue manager clusters to maintain a list of clusters referred to by more than one IBM MQ object.

You can define and modify namelists by using MQSC commands.  Alternatively, on z/OS, you can use the IBM MQ for z/OS operations and control panels

Programs can use the MQI to find out which queues are included in these namelists. The organization of the namelists is the responsibility of the application designer and system administrator.

Related information:

Attributes for namelists

Authentication information objects

An authentication information object provides the definitions required to perform certificate revocation checking.

The queue manager authentication information object forms part of IBM MQ support for Transport Layer Security (TLS). It provides the definitions needed to check for revoked certificates. Certification Authorities revoke certificates that can no longer be trusted.

You can use the MQSC command **DEFINE AUTHINFO** to define an authentication information object. For more information about the attributes of authentication information objects, see **DEFINE AUTHINFO**.

You can use the following IBM MQ control commands with an authentication information object:

- **setmqaut** (grant or revoke authority)
- **dspmqa** (display object authorization)
- **dmpmqaut** (dump authorizations)
- **rcrmqobj** (re-create object)
- **rcdmqimg** (record media image)
- **dspmqls** (display file names)

For an overview of TLS, and the use of the authentication information objects, see TLS security protocols in IBM MQ.

Related information:

Transport Layer Security (TLS) concepts

Communication information objects

IBM MQ Multicast offers low latency, high fanout, reliable multicast messaging. A communication information (COMMINFO) object is needed to use Multicast transmission.

A COMMINFO object is an IBM MQ object that contains the attributes associated with multicast transmission. For more information about these attributes, see **DEFINE COMMINFO**. For more information about creating a COMMINFO object, see *Getting started with multicast*.

Related concepts:

“IBM MQ Multicast” on page 251

IBM MQ Multicast offers low latency, high fan out, reliable multicast messaging.

Channels

A *channel* is a logical communication link, used by distributed queue managers, between an IBM MQ MQI client and an IBM MQ server, or 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.

A channel has two definitions: one at each end of the connection. For queue managers to communicate with one another, you must define one channel object at the queue manager that is to send messages, and another, complementary one, at the queue manager that is to receive them. The same *channel name* must be used at each end of the connection, and the *channel type* used must be compatible.

There are two categories of channel in IBM MQ, with different channel types within these categories:

- Message channels, which are unidirectional, and transfer messages from one queue manager to another.
- MQI channels, which are bidirectional, and transfer MQI calls from an IBM MQ MQI client to a queue manager, and responses from a queue manager to an IBM MQ client.

Message channels

A message channel is a one-way link. It connects two queue managers by using *message channel agents* (MCAs).

The purpose of a message channel is to transfer messages from one queue manager to another. Message channels are not required by the client server environment.

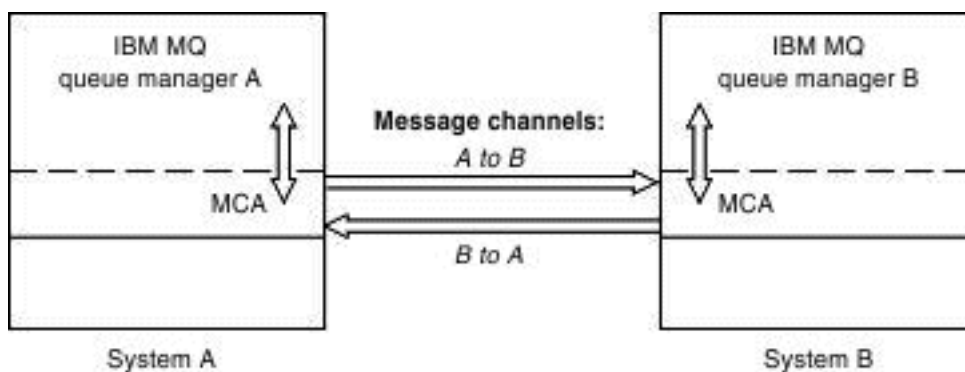


Figure 2. Message channels between two queue managers

For more information, see Channel-exit calls and data structures.

MQI channels

A Message Queue Interface (MQI) channel connects an IBM MQ MQI client to a queue manager on a server machine, and is established when you issue an MQCONN or MQCONNX call from an IBM MQ MQI client application.

It is a two-way link and is used for the transfer of MQI calls and responses only, including MQPUT calls that contain message data and MQGET calls that result in the return of message data. There are different ways of creating and using the channel definitions (see Defining MQI channels).

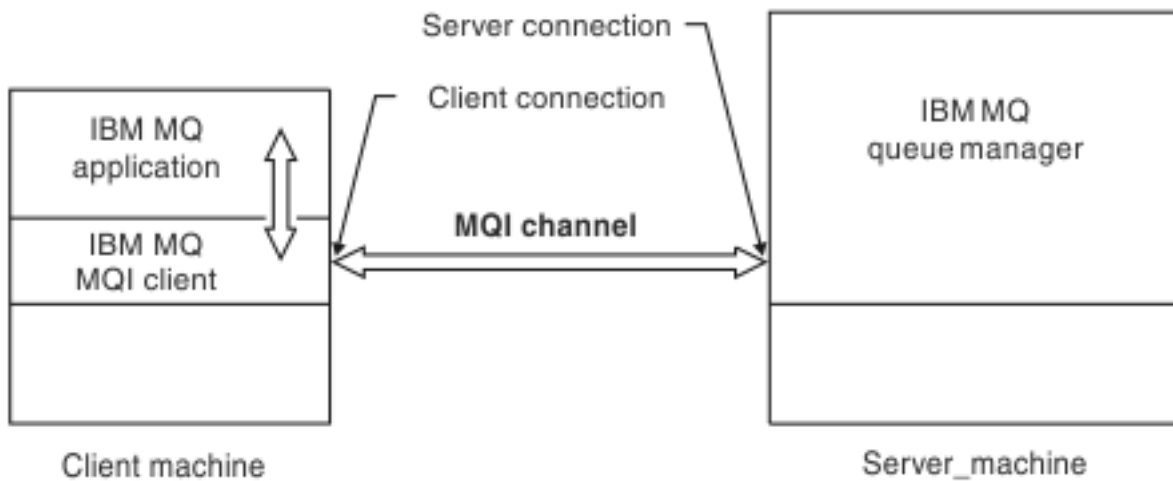


Figure 3. Client-connection and server-connection on an MQI channel

z/OS 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 (see Connecting a client to a queue-sharing group).

There are two channel types for MQI channel definitions. They define the bi-directional MQI channel.

Client-connection channel

This type is for the IBM MQ MQI client.

Server-connection channel

This type is for the server running the queue manager, with which the IBM MQ application, running in an IBM MQ MQI client environment, is to communicate.

Related concepts:

“Distributed queuing and clusters” on page 182

Distributed queuing means sending messages from one queue manager to another. The receiving queue manager can be on the same machine or another; nearby 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 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.

Related reference:

“Communications” on page 172

IBM MQ MQI clients use MQI channels to communicate with the server.

Related information:

Administering remote IBM MQ objects

Channel-exit calls and data structures

Stopping MQI channels

Message Queue Interface overview

What is a channel?:

A channel is a logical communication link between an IBM MQ MQI client and an IBM MQ server, or between two IBM MQ servers.

This information has moved. See “Channels” on page 170.

Message Channels:

A message channel is a one-way link. It connects two queue managers by using *message channel agents* (MCAs).

This information has moved. See “Channels” on page 170.

Message Queue Interface (MQI) channels:

An Message Queue Interface (MQI) channel connects an IBM MQ MQI client to a queue manager on a server machine, and is established when you issue an MQCONN or MQCONNX call from an IBM MQ MQI client application.

This information has moved. See “Channels” on page 170.


Communications

IBM MQ MQI clients use MQI channels to communicate with the server.

A channel definition must be created at both the IBM MQ MQI client and server ends of the connection. How to create channel definitions is explained in Defining MQI channels.

The transmission protocols possible are shown in the following table:

Table 6. Transmission protocols for MQI channels

Client platform	LU 6.2	TCP/IP	NetBIOS	SPX
 IBM i		Yes		
UNIX and Linux systems	Yes ¹	Yes		
Windows	Yes	Yes	Yes	Yes

Note:

1. LU6.2 is not supported on Linux (POWER platform), Linux (x86-64 platform), Linux (zSeries s390x platform), or Solaris (x86-64 platform)

Transmission protocols - combination of IBM MQ MQI client and server platforms shows the possible combinations of IBM MQ MQI client and server platforms, using these transmission protocols.

An IBM MQ application on an IBM MQ MQI client can use all the MQI calls in the same way as when the queue manager is local. **MQCONN** or **MQCONNX** associates the IBM MQ application with the selected queue manager, creating a *connection handle*. Other calls using that connection handle are then processed by the connected queue manager. IBM MQ MQI client communication requires an active connection between the client and server, in contrast to communication between queue managers, which is connection-independent and time-independent.

The transmission protocol is specified by using the channel definition and does not affect the application. For example, a Windows application can connect to one queue manager over TCP/IP and to another queue manager over NetBIOS.

Performance considerations

The transmission protocol you use might affect the performance of the IBM MQ client and server system. For dial-up support over a slow telephone line, it might be advisable to use IBM MQ channel compression.

Client connection channels

Client connection channels are objects that provide a communication path from an IBM MQ MQI client to a queue manager.

Client connection channels are used in distributed queuing to move messages between a queue manager and a client. They shield applications from the underlying communications protocols. The client might exist on the same, or different, platform from the queue manager.

Related information:

Configuring connections between the server and client

Storage classes

▶ z/OS

A storage class maps one or more queues to a page set.

This means that messages for that queue are stored (subject to buffering) on that page set.

Storage classes are supported only on IBM MQ for z/OS.

For further information about storage classes, see the Planning on z/OS.

Listeners

Listeners are processes that accept network requests from other queue managers, or client applications, and start associated channels.

Listener processes can be started using the `runmq1sr` control command.

Listener objects are IBM MQ objects that allow you to manage the starting and stopping of listener processes from within the scope of a queue manager. By defining attributes of a listener object you do the following:

- Configure the listener process.
- Specify whether the listener process automatically starts and stops when the queue manager starts and stops.

Important: ▶ z/OS Listener objects are not supported on IBM MQ for z/OS. For more information about how IBM MQ for z/OS implements listening, by using the channel initiator, see “The channel initiator on z/OS” on page 317.

Related information:

`runmq1sr` (run listener)

Services

Service objects are a way of defining programs to be run when a queue manager starts or stops.

Programs can be one of the following types:

Servers

A *server* is a service object that has the parameter `SERVTYPE` specified as `SERVER`. A server service object is the definition of a program that will be executed when a specified queue manager is started. Only one instance of a server process can be executed concurrently. While running, the status of a server process can be monitored using the MQSC command, `DISPLAY SVSTATUS`. Typically server service objects are definitions of programs such as dead letter handlers or trigger monitors, however the programs that can be run are not limited to those supplied with IBM MQ. Additionally, a server service object can be defined to include a command that will be run when the specified queue manager is shut down to end the program.

Commands

A *command* is a service object that has the parameter `SERVTYPE` specified as `COMMAND`. A command service object is the definition of a program that will be executed when a specified queue manager is started or stopped. Multiple instances of a command process can be executed concurrently. Command service objects differ from server service objects in that once the program is executed the queue manager will not monitor the program. Typically command service objects are definitions of programs that are short lived and will perform a specific task such as starting one, or more, other tasks.

Important:  Service objects are not supported on IBM MQ for z/OS.

Related information:

Working with services

Topic objects

A *topic object* is an IBM MQ object that allows you to assign specific, non-default attributes to topics.

A *topic* is defined by an application publishing or subscribing to a particular *topic string*. A topic string can specify a hierarchy of topics by separating them with a forward slash character (/). This can be visualized by a *topic tree*. For example, if an application publishes to the topic strings `/Sport/American Football` and `/Sport/Soccer`, a topic tree is created that has a parent node `Sport` with two children, `American Football`, and `Soccer`.

Topics inherit their attributes from the first parent administrative node found in their topic tree. If there are no administrative topic nodes in a particular topic tree, then all topics inherit their attributes from the base topic object, `SYSTEM.BASE.TOPIC`.

You can create a topic object at any node in a topic tree by specifying that node's topic string in the `TOPICSTR` attribute of the topic object. You can also define other attributes for the administrative topic node. For more information about these attributes, see *The MQSC commands*, or *Automating administration using PCF commands*. Each topic object, by default, inherits its attributes from its closest parent administrative topic node.

Topic objects can also be used to hide the full topic tree from application developers. If a topic object named `FOOTBALL.US` is created for the topic `/Sport/American Football`, an application can publish or subscribe to the object named `FOOTBALL.US` instead of the string `/Sport/American Football` with the same result.

If you enter a #, +, /, or * character within a topic string on a topic object, the character is treated as a normal character within the string, and is considered to be part of the topic string associated with a topic object.

For more information about topic objects, see “Publish/subscribe messaging” on page 203.

Naming IBM MQ objects

The naming convention adopted for IBM MQ objects depends on the object. The name of the machines and the user IDs that you use with IBM MQ are also subject to some naming restrictions.

Each instance of a queue manager is known by its name. This name must be unique within the network of interconnected queue managers, so that one queue manager can unambiguously identify the target queue manager to which any given message is sent.

For the other types of object, each object has a name associated with it and can be referred to by that name. These names must be unique within one queue manager and object type. For example, you can have a queue and a process with the same name, but you cannot have two queues with the same name.

In IBM MQ, names can have a maximum of 48 characters, with the exception of *channels* which have a maximum of 20 characters. For more information about naming IBM MQ objects, see “Rules for naming IBM MQ objects.”

The name of the machines and the user IDs that you use with IBM MQ are also subject to some naming restrictions:

- Ensure that the machine name does not contain any spaces. IBM MQ does not support machine names that include spaces. If you install IBM MQ on such a machine, you cannot create any queue managers.
- For IBM MQ authorizations, names of user IDs and groups must be no longer than 20 characters (spaces are not allowed).
- **Windows** An IBM MQ for Windows server does not support the connection of a Windows client if the client is running under a user ID that contains the @ character, for example, abc@d.

Related concepts:

“IBM MQ file names” on page 178

Each IBM MQ queue manager, queue, process definition, namelist, channel, client connection channel, listener, service, and authentication information object is represented by a file. Because object names are not necessarily valid file names, the queue manager converts the object name into a valid file name where necessary.

Related reference:

“Rules for naming IBM MQ objects”

IBM MQ object names have maximum lengths and are case-sensitive. Not all characters are supported for every object type, and many objects have rules concerning the uniqueness of names.

Rules for naming IBM MQ objects

IBM MQ object names have maximum lengths and are case-sensitive. Not all characters are supported for every object type, and many objects have rules concerning the uniqueness of names.

There are many different types of IBM MQ object, and objects from each type can all have the same name because they exist in separate object namespaces: For example, a local queue and a sender channel can both have the same name. However, an object cannot have the same name as another object in the same namespace: For example, a local queue cannot have the same name as a model queue, and a sender channel cannot have the same name as a receiver channel.

The following IBM MQ objects exist in separate object namespaces:

- Authentication information


- Channel
- Client channel
- Listener
- Namelist
- Process
- Queue
- Service
- Storage class
- Subscription
- Topic

Character length of object names

In general, IBM MQ object names can be up to 48 characters long. This rule applies to the following objects:

- Authentication information
- Cluster
- Listener
- Namelist
- Process definition
- Queue
- Queue manager
- Service
- Subscription
- Topic











There are restrictions:

1.  On z/OS systems, queue managers must be a maximum of 4 characters, and must be in uppercase characters and numeric characters only.
2. The maximum length of channel object names and client connection channel names is 20 characters. See *Defining the channels* for more information about channels.
3. Topic strings can be a maximum of 10240 bytes. All IBM MQ object names are case-sensitive.
4. Subscription names can be a maximum of 10240 bytes, and can contain spaces.
5. The maximum length of storage class names is 8 characters.
6. The maximum length of CF structure names is 12 characters.

Characters in object names

The valid characters for IBM MQ object names are:

Characters	Restrictions
Uppercase A - Z	<ul style="list-style-type: none"> • None

Characters	Restrictions
Lowercase a - z	<ul style="list-style-type: none"> • In MQSC scripts, names with lowercase characters must be enclosed in single quotation marks. This prevents the lowercase characters being folded into uppercase. • Systems using EBCDIC Katakana cannot use lowercase a- z characters in object names. •  There might be restrictions when using lowercase characters on z/OS systems, for example, queue manager names cannot contain lowercase characters. •  On IBM i systems when using CL commands, names with lowercase characters must be enclosed in single quotation marks. This prevents the lowercase characters being folded into uppercase.
Numerics 0 - 9	<ul style="list-style-type: none"> • None
Period (.)	<ul style="list-style-type: none"> • None
Underscore (_)	<ul style="list-style-type: none"> •  None •  Avoid using names with leading or trailing underscores because they cannot be handled by the IBM MQ for z/OS operations and control panels.
Forward slash (/)	<ul style="list-style-type: none"> •  On Windows systems, the first character of a queue manager name cannot be a forward slash. •  On IBM i systems when using CL commands, names containing a forward slash must be enclosed in single quotation marks. •  None
Percent sign (%)	<ul style="list-style-type: none"> •  None •  If you are using RACF as the external security manager for IBM MQ for z/OS, do not use % in object names because the names are not included in security checks when RACF generic profiles are used. •  On IBM i systems when using CL commands, names containing a percent sign must be enclosed in single quotation marks.

There are also some general rules concerning characters on object names:

1. Leading or embedded blanks are not allowed.
2. National language characters are not allowed.
3. Any name that is less than the full field length can be padded to the right with blanks. All short names that are returned by the queue manager are always padded to the right with blanks.

Queue names

The name of a queue has two parts:

- The name of a queue manager
- The local name of the queue as it is known to that queue manager

Each part of the queue name is 48 characters long.

To refer to a local queue, you can omit the name of the queue manager (by replacing it with blank characters or using a leading null character). However, all queue names returned to a program by IBM MQ contain the name of the queue manager.

z/OS A shared queue, accessible to any queue manager in its queue-sharing group, cannot have the same name as any non-shared local queue in the same queue-sharing group. This restriction avoids the possibility of an application mistakenly opening a shared queue when it intended to open a local queue, or vice versa. Shared queues and queue-sharing groups are only available on IBM MQ for z/OS.

To refer to a remote queue, a program must include the name of the queue manager in the full queue name, or there must be a local definition of the remote queue.

When an application uses a queue name, that name can be either the name of a local queue (or an alias to one) or the name of a local definition of a remote queue, but the application does not need to know which, unless it needs to get a message from the queue (when the queue must be local). When the application opens the queue object, the MQOPEN call performs a name resolution function to determine on which queue to perform subsequent operations. The significance of this is that the application has no built-in dependency on particular queues being defined at particular locations in a network of queue managers. Therefore, if a system administrator relocates queues in the network, and changes their definitions, the applications that use those queues do not need to be changed.

Reserved object names

Object names that start with **SYSTEM.** are reserved for objects defined by the queue manager. You can use the **Alter**, **Define**, and **Replace** commands to change these object definitions to suit your installation. The names that are defined for IBM MQ are listed in full in Queue names.

z/OS On IBM MQ for z/OS, the coupling facility application structure name CSQSYSAPPL is reserved.

IBM MQ file names

Each IBM MQ queue manager, queue, process definition, namelist, channel, client connection channel, listener, service, and authentication information object is represented by a file. Because object names are not necessarily valid file names, the queue manager converts the object name into a valid file name where necessary.

The default path to a queue manager directory is as follows:

- A prefix, which is defined in the IBM MQ configuration information:
 - **Windows** On Windows 32-bit systems, the default prefix is C:\Program Files (x86)\IBM\WebSphere MQ. On Windows 64-bit systems, the default prefix is C:\Program Files\IBM\MQ. For both 32-bit and 64-bit installations, the data directories are installed into C:\ProgramData\IBM\MQ. This is configured in the DefaultPrefix stanza of the mqs.ini configuration file.
 - **UNIX** On UNIX and Linux, the default prefix is /var/mqm. This is configured in the DefaultPrefix stanza of the mqs.ini configuration file.

Where available, the prefix can be changed using the IBM MQ properties page in the IBM MQ Explorer, otherwise edit the mqs.ini configuration file manually.

- The queue manager name is transformed into a valid directory name. For example, the queue manager:

queue.manager

would be represented as:

queue!manager

This process is referred to as *name transformation*.

In IBM MQ, you can give a queue manager a name containing up to 48 characters.

For example, you could name a queue manager:





QUEUE.MANAGER.ACCOUNTING.SERVICES

However, each queue manager is represented by a file and there are limitations on the maximum length of a file name, and on the characters that can be used in the name. As a result, the names of files representing objects are automatically transformed to meet the requirements of the file system.

The rules governing the transformation of a queue manager name are as follows:

1. Transform individual characters:
 - From . to !
 - From / to &
2. If the name is still not valid:
 - a. Truncate it to eight characters
 - b. Append a three-character numeric suffix

For example, assuming the default prefix and a queue manager with the name queue.manager:

-  On Windows with NTFS or FAT32, the queue manager name becomes:
C:\Program Files\IBM\MQ\mqgrs\queue!manager
-  On Windows with FAT, the queue manager name becomes:
C:\Program Files\IBM\MQ\mqgrs\queue!ma
-   On UNIX and Linux, the queue manager name becomes:
/var/mqm/mqgrs/queue!manager

The transformation algorithm also distinguishes between names that differ only in case on file systems that are not case sensitive.

Object name transformation

Object names are not necessarily valid file system names. You might need to transform your object names. The method used is different from that for queue manager names because, although there are only a few queue manager names on each machine, there can be a large number of other objects for each queue manager. Queues, process definitions, namelists, channels, client connection channels, listeners, services, and authentication information objects are represented in the file system.

When a new name is generated by the transformation process, there is no simple relationship with the original object name. You can use the **dspmqls** command to convert between real and transformed object names.

Related information:

dspmqls (display file names)

Object names on IBM i



A queue manager has an associated queue manager library that has a unique name. Queue manager names and object names might need to be transformed to meet the requirements of the IBM i Integrated File System (IFS).

When a queue manager is created, IBM MQ associates a queue manager library with it. This queue manager library is given a unique name, no more than 10 characters long, largely based on the user defined queue manager name. Both the queue manager, and the queue manager library are placed in to a directory that is also based on the queue manager name with the prefix /QIBM/UserData/mqm. An example of a queue manager, queue manager library, and directory follows:

Queue manager name	ORANGE
Queue manager library name	QMORANGE
Directory	/QIBM/UserData/mqm/ORANGE

All queue manager names and queue manager library names are written to stanzas in the file /QIBM/UserData/mqm/mqs.ini.

IBM MQ IFS directories and files

The IBM i Integrated File System (IFS) is used extensively by IBM MQ to store data. For more information about the IFS see the *Integrated File System Introduction*.

Each IBM MQ object, for example, a channel or a queue manager, is represented by a file. Because object names are not necessarily valid file names, the queue manager converts the object name into a valid file name where necessary.

The path to a queue manager directory is formed from the following:

- A prefix, which is defined in the queue manager configuration file, qm.ini. The default prefix is /QIBM/UserData/mqm.
- A literal, qmgrs.
- A coded queue manager name, which is the queue manager name transformed into a valid directory name. For example, the queue manager queue/manager is represented by queue&manager.

This process is referred to as name transformation.

IFS queue manager name transformation

In IBM MQ, you can give a queue manager a name containing up to 48 characters.

For example, you can name a queue manager QUEUE/MANAGER/ACCOUNTING/SERVICES. In the same way that a library is created for each queue manager, each queue manager is also represented by a file. Because of variant codepoints in EBCDIC, there are limitations to the characters that can be used in the name. As a result, the names of IFS files representing objects are automatically transformed to meet the requirements of the file system.

Using the example of a queue manager with the name queue/manager, transforming the character / to &, and assuming the default prefix, the queue manager name in IBM MQ for IBM i becomes /QIBM/UserData/mqm/qmgrs/queue&manager.

Object name transformation

Object names are not necessarily valid file system names, so the object names might need to be transformed. The method used is different from that for queue manager names because, although there only a few queue manager names for each machine, there can be a large number of other objects for each queue manager. Only process definitions, queues, and namelists are represented in the file system; channels are not affected by these considerations.

When a new name is generated by the transformation process, there is no simple relationship with the original object name. You can use the `DSPMQMOBJN` command to view the transformed names for IBM MQ objects.

Object attributes

The properties of an object are defined by its attributes. Some you can specify, others you can only view.

For example, the maximum message length that a queue can accommodate is defined by its **MaxMsgLength** attribute; you can specify this attribute when you create a queue. The **DefinitionType** attribute specifies how the queue was created; you can only display this attribute.

In IBM MQ, there are two ways of referring to an attribute:

- Using its PCF name, for example, **MaxMsgLength**.
- Using its MQSC command name, for example, `MAXMSGL`.

Queue-sharing groups

z/OS

Queue managers that can access the same set of shared queues form a group called a *queue-sharing group* (QSG), and they communicate with each other using a coupling facility (CF) that stores the shared queues.

A shared queue is a type of local queue with messages that can be accessed by one or more queue managers that are in a queue-sharing group. This is not the same as a queue being shared by more than one application, using the same queue manager.

Queue-sharing groups have a name of up to four characters. The name must be unique in your network, and must be different from any queue manager names.

Queue-sharing groups are not strictly objects, but are mentioned here for convenience.

Important: Shared queues and queue-sharing groups are supported only on IBM MQ for z/OS.

Related concepts:

“Shared queues and queue-sharing groups” on page 322

You can use shared queues and queue-sharing groups, to implement high availability of IBM MQ resources. Shared queues and queue-sharing groups are functions unique to IBM MQ for z/OS on the z/OS platform.

System default objects

System default objects are a set of object definitions that are created automatically whenever a queue manager is created.

You can copy and modify any of these object definitions for use in applications at your installation.

Default object names have the stem SYSTEM; for example, the default local queue is SYSTEM.DEFAULT.LOCAL.QUEUE, and the default receiver channel is SYSTEM.DEF.RECEIVER. You cannot rename these objects; default objects of these names are required.

When you define an object, any attributes that you do not specify explicitly are copied from the appropriate default object. For example, if you define a local queue, those attributes that you do not specify are taken from the default queue SYSTEM.DEFAULT.LOCAL.QUEUE.

Related information:

System and default objects

Distributed queuing and clusters

Distributed queuing means sending messages from one queue manager to another. The receiving queue manager can be on the same machine or another; nearby 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 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.

Distributed queuing

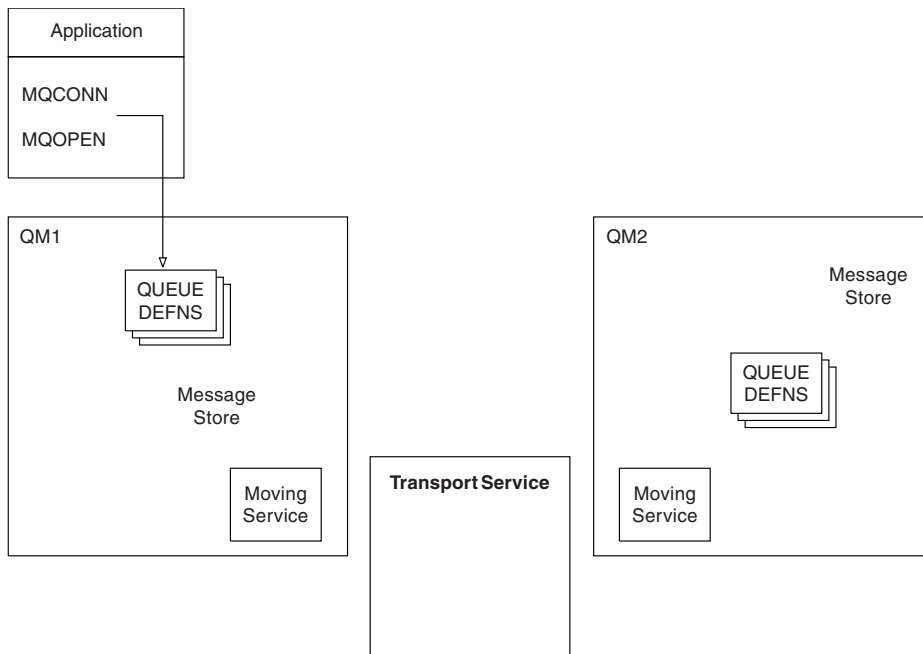


Figure 4. Overview of the components of distributed queuing

In the previous figure:

- An application uses the MQCONN call to connect to a queue manager. The application then uses the MQOPEN call to open a queue so that it can put messages on the queue.
- Each queue manager has a definition for each of its queues. It can have definitions of *local queues* (that is, hosted by this queue manager), and definitions of *remote queues* (that is, hosted by other queue managers).
- If the messages are destined for a remote queue, the local queue manager holds them on a *transmission queue*, which persists them in a message store, until they can be forwarded to the remote queue manager.
- Each queue manager contains communications software, known as the *moving service*, that the queue manager uses to communicate with other queue managers.
- The *transport service* is independent of the queue manager and can be any one of the following (depending on the platform):
 - Systems Network Architecture Advanced Program-to-Program Communication (SNA APPC)
 - Transmission Control Protocol/Internet Protocol (TCP/IP)
 - Network Basic Input/Output System (NetBIOS)
 - Sequenced Packet Exchange (SPX)

Components needed to send a message

If a message is to be sent to a remote queue manager, the local queue manager needs definitions for a *transmission queue* and a *channel*. A channel is a one-way communication link between two queue managers. It can carry messages destined for any number of queues at the remote queue manager.

Each end of a channel has a separate definition, defining it, for example, as the sending end or the receiving end. A simple channel consists of a *sender* channel definition at the local queue manager and a *receiver* channel definition at the remote queue manager. These two definitions must have the same name, and together they constitute one channel.

The software that handles the sending and receiving of messages is called the *Message Channel Agent* (MCA). There is a *message channel agent* (MCA) at each end of a channel.

Each queue manager should have a *dead-letter queue* (also known as the *undelivered message queue*). Messages are put on this queue if they cannot be delivered to their destination.

The following figure shows the relationship between queue managers, transmission queues, channels, and MCAs:

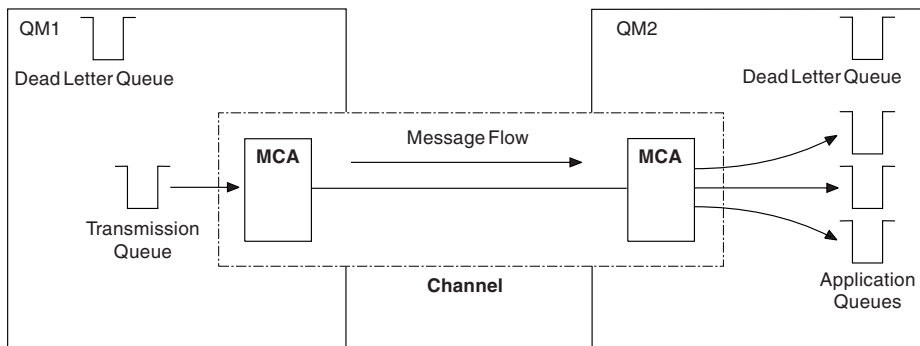


Figure 5. Sending messages

Components needed to return a message

If your application requires messages to be returned from the remote queue manager, you need to define another channel, to run in the opposite direction between the queue managers, as shown in the following figure:

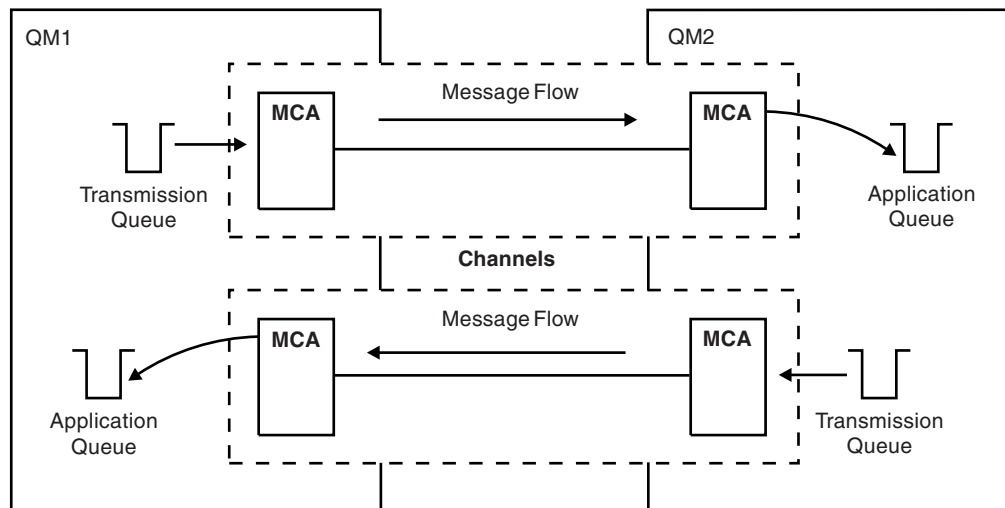


Figure 6. Sending messages in both directions

Clusters

Rather than manually defining all the connections in a distributed queuing environment, you can group a set of queue managers in a cluster. When you do this, the queue managers can make the queues that they host available to other queue managers in the cluster without the need for explicit channel definitions, remote-queue definitions, or transmission queues for each destination. Every queue manager in a cluster has a single transmission queue that transmits messages to any other queue manager in the cluster. For

each queue manager, you only need to define one cluster-receiver channel and one cluster-sender channel; any additional channels are automatically managed by the cluster.

An IBM MQ client can connect to a queue manager that is part of a cluster, just as it can connect to any other queue manager. As with manually-configured distributed queuing, you use the MQPUT call to put a message to a queue at any queue manager. You use the MQGET call to retrieve messages from a local queue.

Queue managers on platforms that support clusters do not have to be part of a cluster. You can continue to manually configure distributed queuing as well as, or instead of, using clusters.

Benefits of using clusters

Clustering provides two key benefits:

- Clusters simplify the administration of IBM MQ networks, which usually require many object definitions for channels, transmit queues, and remote queues to be configured. This situation is especially true in large, potentially changing, networks where many queue managers need to be interconnected. This architecture is particularly hard to configure and actively maintain.
- 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 located on multiple queue managers. This distribution of the workload can be used to achieve greater resilience to system failures, and to improve the scaling performance of particularly active message flows in a system. In such an environment, each of the instances of the distributed queues have consuming applications processing the messages. For more information, see *Using clusters for workload management*.

How messages are routed in a cluster

You can think of a cluster as a network of queue managers maintained by a conscientious systems administrator. Whenever you define a cluster queue, the systems administrator automatically creates corresponding remote-queue definitions as needed on the other queue managers.

You do not need to make transmission queue definitions because IBM MQ provides a transmission queue on each queue manager in the cluster. This single transmission queue can be used to carry messages to any other queue manager in the cluster. You are not limited to using a single transmission queue. A queue manager can use multiple transmission queues to separate the messages going to each queue manager in a cluster. Typically, a queue manager uses a single cluster transmission queue. You can change the queue manager attribute DEFCLXQ, so that a queue manager uses a different cluster transmission queue for each queue manager in a cluster. You can also define cluster transmission queues manually.

All the queue managers that join a cluster agree to work in this way. They send out information about themselves and about the queues they host, and they receive information about the other members of the cluster.

To ensure that no information is lost when a queue manager becomes unavailable, you specify two queue managers in the cluster to act as *full repositories*. These queue managers store a full set of information about all the queue managers and queues in the cluster. All other queue managers in the cluster only store information about those queue managers and queues with which they exchange messages. These queue managers are known as *partial repositories*. For more information, see “Cluster repository” on page 197.

In order to become part of a cluster, a queue manager must have two channels; a cluster-sender channel and a cluster-receiver channel:

- A cluster-sender channel is a communication channel like a sender channel. You must manually create one cluster-sender channel on a queue manager to connect it to a full repository that is already a member of the cluster.
- A cluster-receiver channel is a communication channel like a receiver channel. You must manually create one cluster-receiver channel. The channel acts as the mechanism for the queue manager to receive cluster communications.

All other channels that are needed for communication between this queue manager and other members of the cluster are then created automatically.

The following figure shows the components of a cluster called CLUSTER:

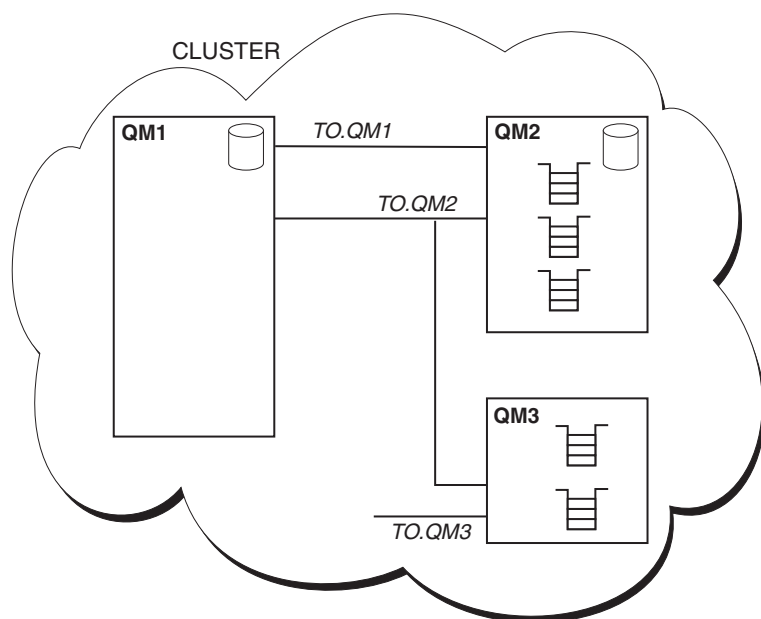


Figure 7. A cluster of queue managers

- CLUSTER contains three queue managers, QM1, QM2, and QM3.
- QM1 and QM2 host full repositories of information about the queue managers and queues in the cluster.
- QM2 and QM3 host some cluster queues, that is, queues that are accessible to any other queue manager in the cluster.
- Each queue manager has a cluster-receiver channel called TO.qmgr on which it can receive messages.
- Each queue manager also has a cluster-sender channel on which it can send information to one of the repository queue managers.
- QM1 and QM3 send to the repository at QM2 and QM2 sends to the repository at QM1.

Distributed queuing components

The components of distributed queuing are message channels, message channel agents, transmission queues, channel initiators and listeners, and channel-exit programs. The definition of each end of a message channel can be one of several types.

Message channels are the channels that carry messages from one queue manager to another. Do not confuse message channels with MQI channels. There are two types of MQI channel, server-connection and client-connection. For more information, see MQI channels.

The definition of each end of a message channel can be one of the following types:

- Sender
- Receiver
- Server
- Requester
- Cluster sender
- Cluster receiver

A message channel is defined using one of these types defined at one end, and a compatible type at the other end. Possible combinations are:

- Sender-receiver
- Requester-server
- Requester-sender (callback)
- Server-receiver
- Cluster sender-cluster receiver

Detailed instructions for creating a sender-receiver channel are included in [Defining the channels](#). For examples of the parameters needed to set up sender-receiver channels, see [Example configuration information](#) applicable to your platform. For the parameters needed to define a channel of any type, see [DEFINE CHANNEL](#).

Sender-receiver channels

A sender in one system starts the channel so that it can send messages to the other system. The sender requests the receiver at the other end of the channel to start. The sender sends messages from its transmission queue to the receiver. The receiver puts the messages on the destination queue. Figure 8 on page 188 illustrates this.

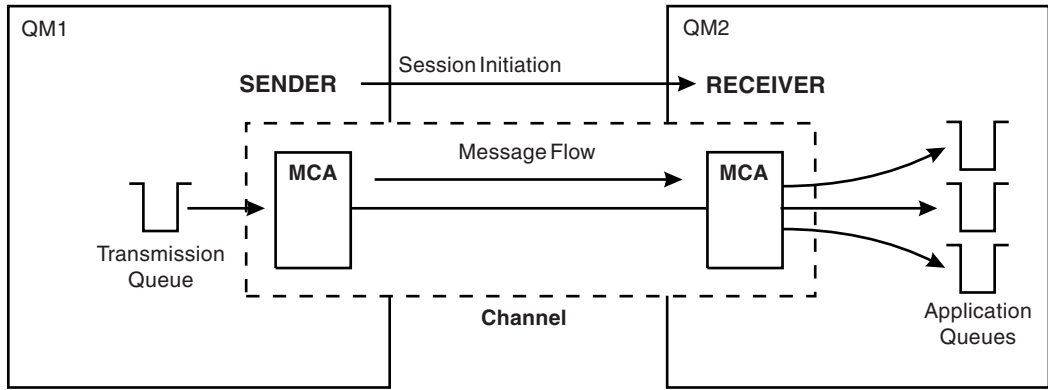


Figure 8. A sender-receiver channel

Requester-server channels

A requester in one system starts the channel so that it can receive messages from the other system. The requester requests the server at the other end of the channel to start. The server sends messages to the requester from the transmission queue defined in its channel definition.

A server channel can also initiate the communication and send messages to a requester. This applies only to *fully qualified* servers, that is server channels that have the connection name of the partner specified in the channel definition. A fully qualified server can either be started by a requester, or can initiate a communication with a requester.

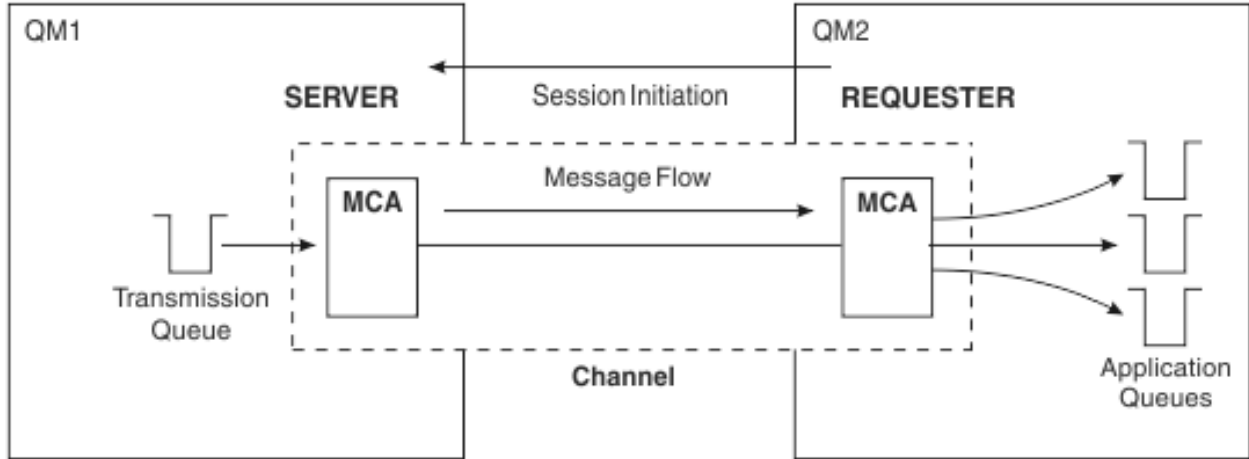


Figure 9. A requester-server channel

Requester-sender channels

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

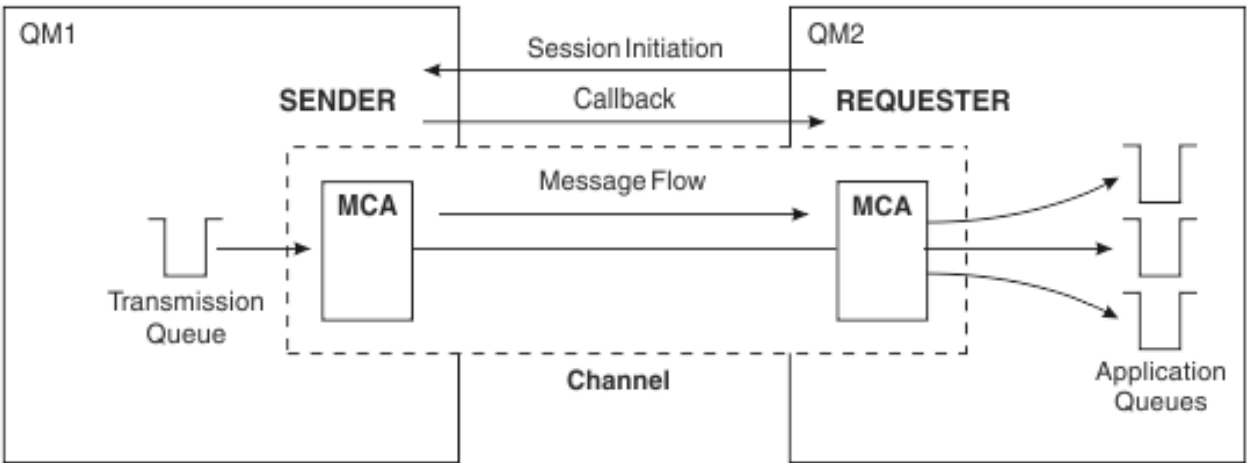


Figure 10. A requester-sender channel

Server-receiver channels

This is like sender-receiver but applies only to *fully qualified* servers, that is server channels that have the connection name of the partner specified in the channel definition. Channel startup must be initiated at the server end of the link. The illustration of this is like the illustration in Figure 8 on page 188.

Cluster-sender channels

In a cluster, each queue manager has a cluster-sender channel on which it can send cluster information to one of the full repository queue managers. Queue managers can also send messages to other queue managers on cluster-sender channels.

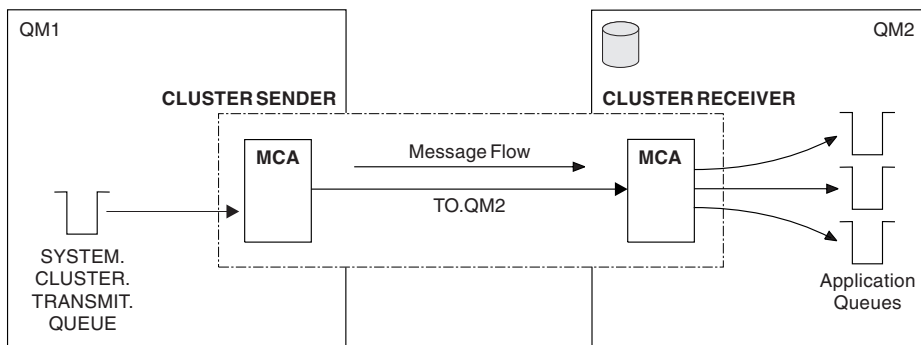


Figure 11. A cluster-sender channel

Cluster-receiver channels

In a cluster, each queue manager has a cluster-receiver channel on which it can receive messages and information about the cluster. The illustration of this is like the illustration in Figure 11.

Dead-letter queues

The dead-letter queue (or undelivered-message queue) is the queue to which messages are sent if they cannot be routed to their correct destination. Each queue manager typically has a dead-letter queue.

A *dead-letter queue* (DLQ), sometimes referred to as an *undelivered-message queue*, is a holding queue for messages that cannot be delivered to their destination queues, for example because the queue does not exist, or because it is full. Dead-letter queues are also used at the sending end of a channel, for data-conversion errors.. Every queue manager in a network typically has a local queue to be used as a dead-letter queue so that messages that cannot be delivered to their correct destination can be stored for later retrieval.

Messages can be put on the DLQ by queue managers, message channel agents (MCAs), and applications. All messages on the DLQ 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 DLQ.

You should typically define a dead-letter queue for each queue manager. If you do not, and the MCA is unable to put a message, it is left on the transmission queue and the channel is stopped. Also, if fast, non-persistent messages (see Fast, nonpersistent messages) cannot be delivered, and no dead-letter queue exists on the target system, these messages are discarded.

However, using dead-letter queues can affect the sequence in which messages are delivered, and so you might choose not to use them.

Remote queue definitions

Remote queue definitions are definitions for queues that are owned by another queue manager.

Whereas applications can retrieve messages only from local queues, they can put messages on local queues or remote queues. Therefore, as well as a definition for each of its local queues, a queue manager can have *remote queue definitions*. The advantage of remote queue definitions is that they enable an application to put a message to a remote queue without having to specify the name of the remote queue or the remote queue manager, or the name of the transmission queue. Remote queue definitions give you location independence.

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

How to get to the remote queue manager

You might not always have one channel between each source and target queue manager. There are a number of other ways of linking between the two, including multi-hopping, sharing channels, using different channels and clustering.

Multi-hopping

If there is no direct communication link between the source queue manager and the target queue manager, it is possible to pass through one or more *intermediate queue managers* on the way to the target queue manager. This is known as a *multi-hop*.

You need to define channels between all the queue managers, and transmission queues on the intermediate queue managers. This is shown in Figure 12 on page 191.

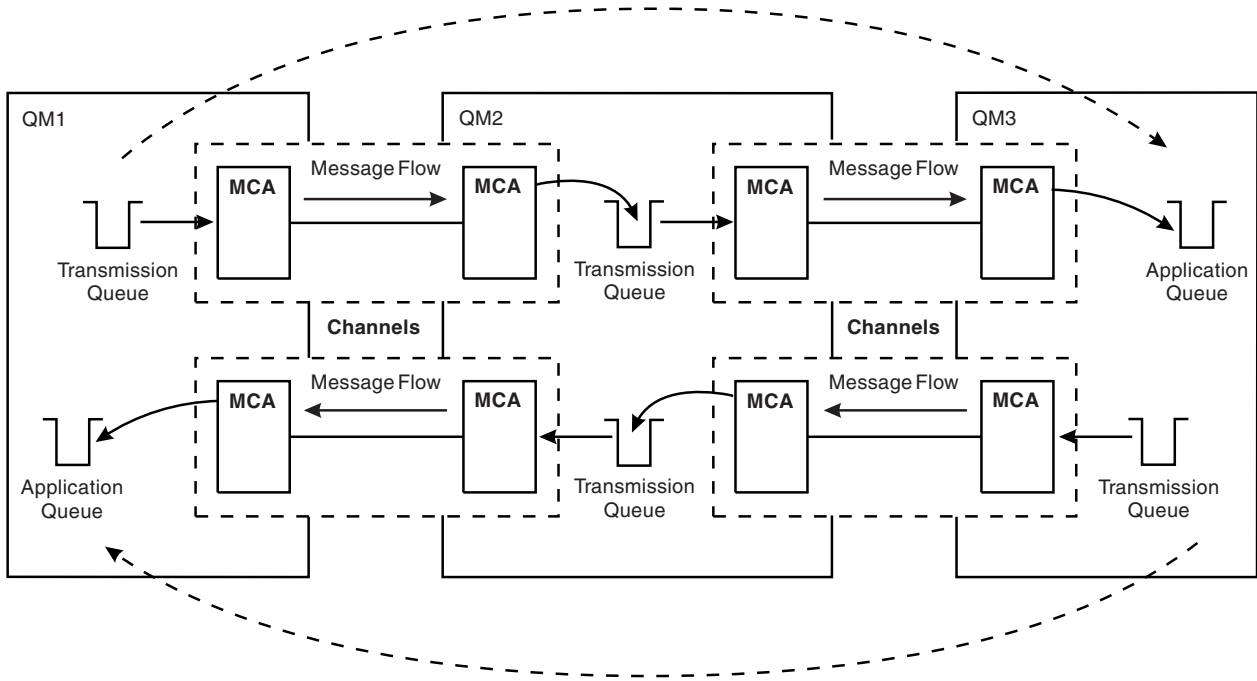


Figure 12. Passing through intermediate queue managers

Sharing channels

As an application designer, you have the choice of forcing your applications to specify the remote queue manager name along with the queue name, or creating a *remote queue definition* for each remote queue. This definition holds the remote queue manager name, the queue name, and the name of the transmission queue. Either way, all messages from all applications addressing queues at the same remote location have their messages sent through the same transmission queue. This is shown in Figure 13.

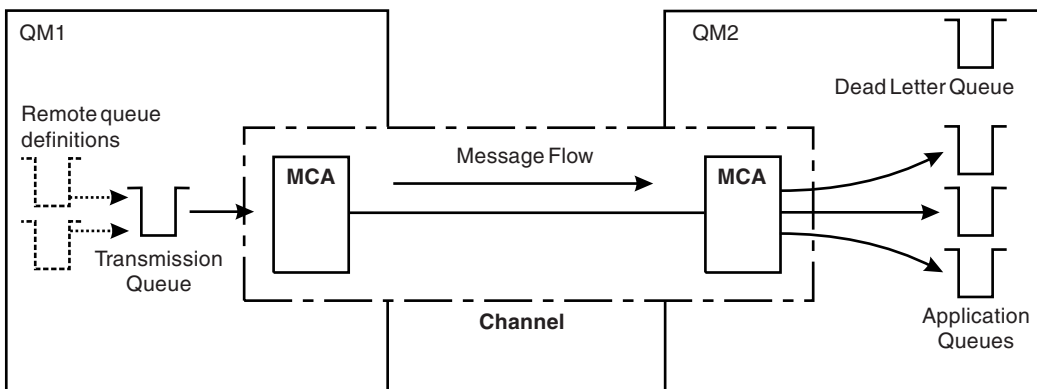


Figure 13. Sharing a transmission queue

Figure 13 illustrates that messages from multiple applications to multiple remote queues can use the same channel.

Using different channels

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

To set up a second channel you need to define another channel and another transmission queue, and create a remote queue definition specifying the location and the transmission queue name. Your applications can then use either channel but the messages are still delivered to the same target queues. This is shown in Figure 14.

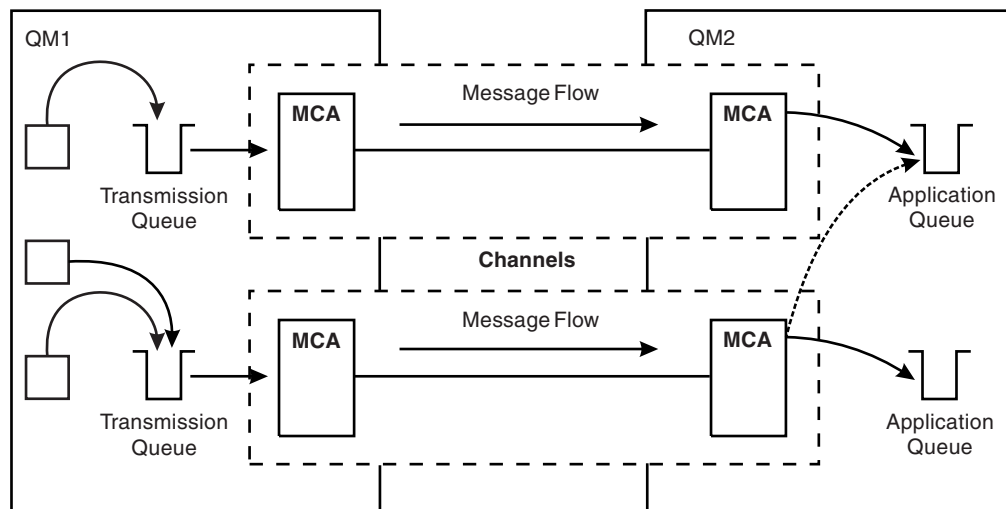


Figure 14. Using multiple channels

When you use remote queue definitions to specify a transmission queue, your applications must **not** specify the location (that is, the destination queue manager) themselves. If they do, the queue manager does not use the remote queue definitions. Remote queue definitions give you location independence. Applications can put messages to a *logical* queue without knowing where the queue is located and you can alter the *physical* queue without having to change your applications.

Using clustering

Every queue manager within a cluster defines a cluster-receiver channel. When another queue manager wants to send a message to that queue manager, it defines the corresponding cluster-sender channel automatically. For example, if there is more than one instance of a queue in a cluster, the cluster-sender channel could be defined to any of the queue managers that host the queue. IBM MQ uses a workload management algorithm that uses a round-robin routine to select an available queue manager to route a message to. For more information see Clusters.

Addressing information

When an application puts messages that are destined for a remote queue manager, the local queue manager adds a transmission header to them before placing them on the transmission queue. This header contains the name of the destination queue and queue manager, that is, the *addressing information*.

In a single queue manager environment, the address of a destination queue is established when an application opens a queue for putting messages to. Because the destination queue is on the same queue manager, there is no need for any addressing information.

In a distributed queuing environment, the queue manager needs to know not only the destination queue name, but also the location of that queue (that is, the queue manager name), and the route to that remote location (that is, the transmission queue). This addressing information is contained in the transmission header. The receiving channel removes the transmission header and uses the information in it to locate the destination queue.

You can avoid the need for your applications to specify the name of the destination queue manager if you use a remote queue definition. This definition specifies the name of the remote queue, the name of the remote queue manager to which messages are destined, and the name of the transmission queue used to transport the messages.

What are aliases?

Aliases are used to provide a quality of service for messages. The queue manager alias enables a system administrator to alter the name of a target queue manager without causing you to have to change your applications. It also enables the system administrator to alter the route to a destination queue manager, or to set up a route that involves passing through a number of other queue managers (multi-hopping). The reply-to queue alias provides a quality of service for replies.

Queue manager aliases and reply-to queue aliases are created using a remote-queue definition that has a blank RNAME. These definitions do not define real queues; they are used by the queue manager to resolve physical queue names, queue manager names, and transmission queues.

Alias definitions are characterized by having a blank RNAME.

Queue name resolution

Queue name resolution occurs at every queue manager each time a queue is opened. Its purpose is to identify the target queue, the target queue manager (which might be local), and the route to that queue manager (which might be null). The resolved name has three parts: the queue manager name, the queue name, and, if the queue manager is remote, the transmission queue.

When a remote queue definition exists, no alias definitions are referenced. The queue name supplied by the application is resolved to the name of the destination queue, the remote queue manager, and the transmission queue specified in the remote queue definition. For more detailed information about queue name resolution, see Queue name resolution.

If there is no remote queue definition and a queue manager name is specified, or resolved by the name service, the queue manager looks to see if there is a queue manager alias definition that matches the supplied queue manager name. If there is, the information in it is used to resolve the queue manager name to the name of the destination queue manager. The queue manager alias definition can also be used to determine the transmission queue to the destination queue manager.

If the resolved queue name is not a local queue, both the queue manager name and the queue name are included in the transmission header of each message put by the application to the transmission queue.

The transmission queue used typically has the same name as the resolved queue manager, unless changed by a remote queue definition or a queue manager alias definition. If you have not defined such a transmission queue but you have defined a default transmission queue, then this is used.

► **z/OS** Names of queue managers running on z/OS are limited to four characters.

Queue manager alias definitions

Queue manager alias definitions apply when an application that opens a queue to put a message, specifies the queue name **and** the queue manager name.

Queue manager alias definitions have three uses:

- When sending messages, remapping the queue manager name
- When sending messages, altering or specifying the transmission queue
- When receiving messages, determining whether the local queue manager is the intended destination for those messages

Outbound messages - remapping the queue manager name

Queue manager alias definitions can be used to remap the queue manager name specified in an MQOPEN call. For example, an MQOPEN call specifies a queue name of THISQ and a queue manager name of YOURQM. At the local queue manager, there is a queue manager alias definition like the following example:

```
DEFINE QREMOTE (YOURQM) RQMNAME (REALQM)
```

This shows that the real queue manager to be used, when an application puts messages to queue manager YOURQM, is REALQM. If the local queue manager is REALQM, it puts the messages to the queue THISQ, which is a local queue. If the local queue manager is not called REALQM, it routes the message to a transmission queue called REALQM. The queue manager changes the transmission header to say REALQM instead of YOURQM.

Outbound messages - altering or specifying the transmission queue

Figure 15 shows a scenario where messages arrive at queue manager QM1 with transmission headers showing queue names at queue manager QM3. In this scenario, QM3 is reachable by multi-hopping through QM2.

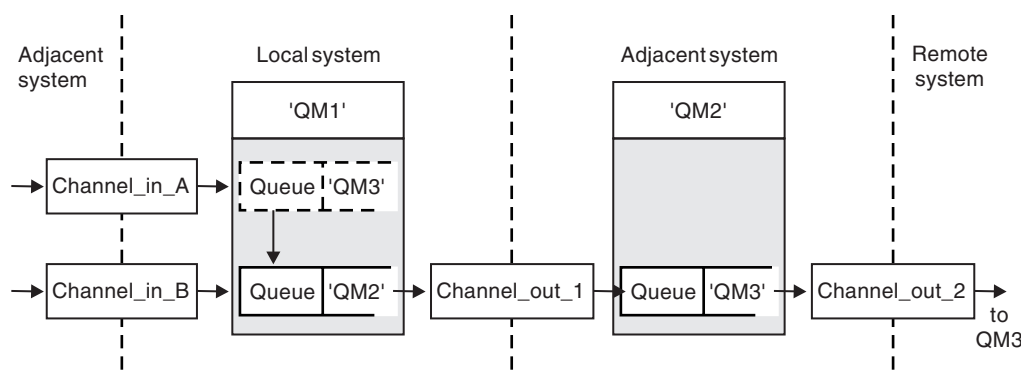


Figure 15. Queue manager alias

All messages for QM3 are captured at QM1 with a queue manager alias. The queue manager alias is named QM3 and contains the definition QM3 through transmission queue QM2. The definition looks like the following example:

```
DEFINE QREMOTE (QM3) RNAME(' ') RQMNAME(QM3) XMITQ(QM2)
```

The queue manager puts the messages on transmission queue QM2 but does not alter the transmission queue header because the name of the destination queue manager, QM3, does not alter.

All messages arriving at QM1 and showing a transmission header containing a queue name at QM2 are also put on the QM2 transmission queue. In this way, messages with different destinations are collected onto a common transmission queue to an appropriate adjacent system, for onward transmission to their destinations.

Inbound messages - determining the destination

A receiving MCA opens the queue referenced in the transmission header. If a queue manager alias definition exists with the same name as the queue manager referenced, then the queue manager name received in the transmission header is replaced with the RQMNAME from that definition.

This process has two uses:

- Directing messages to another queue manager
- Altering the queue manager name to be the same as the local queue manager

Reply-to queue alias definitions

A reply-to queue alias definition specifies alternative names for the reply information in the message descriptor. The advantage of this is that you can alter the name of a queue or queue manager without having to alter your applications.

Queue name resolution

When an application replies to a message, it uses the data in the *message descriptor* of the message it received to find out the name of the queue to reply to. The sending application indicates where replies are sent to and attaches this information to its messages. This concept must be coordinated as part of your application design.

Queue name resolution takes place at the sending end of your application, before the message is put to a queue. Queue name resolution therefore occurs before interaction with the remote application that the message is being sent to. This is the only situation in which name resolution takes place at a time when a queue is not being opened.

Queue name resolution using a queue manager alias

Normally an application specifies a reply-to queue and leaves the reply-to queue manager name blank. The queue manager completes its own name at put time. This method works well except when you want an alternative channel to be used for replies, for example, a channel that uses transmission queue QM1_relief instead of the default return channel which uses transmission queue QM1. In this situation, the queue manager names specified in transmission-queue headers do not match "real" queue manager names, but are respecified using queue manager alias definitions. In order to return replies along alternative routes, it is necessary to map reply-to queue data as well, using reply-to queue alias definitions.

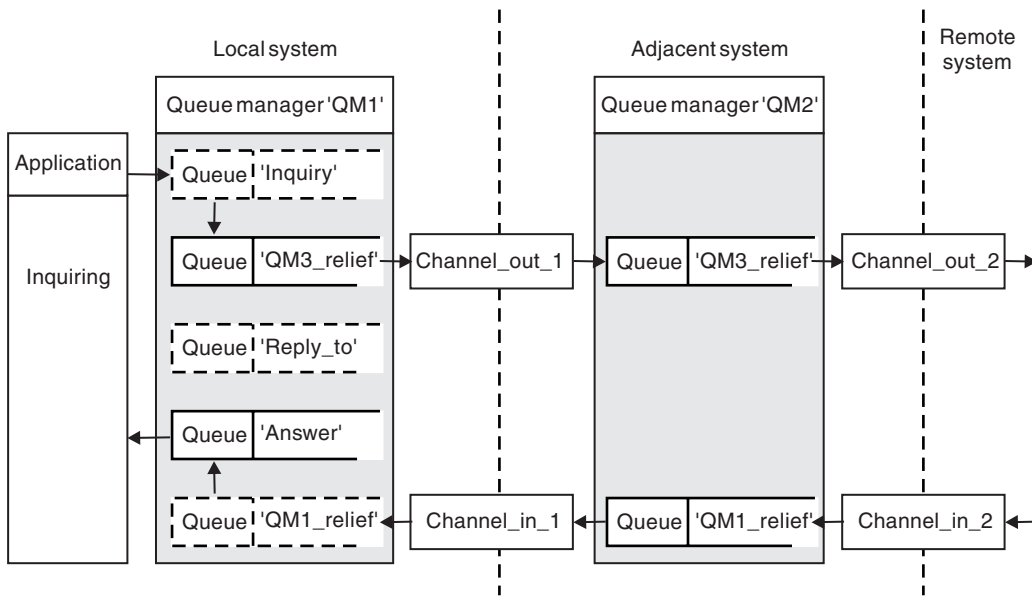


Figure 16. Reply-to queue alias used for changing reply location

In the example in Figure 16:

1. The application puts a message using the MQPUT call and specifying the following information in the message descriptor:

```
ReplyToQ='Reply_to'
ReplyToQMgr=' '
```

ReplyToQMgr must be blank in order for the reply-to queue alias to be used.

2. You create a reply-to queue alias definition called Reply_to, which contains the name Answer, and the queue manager name QM1_relief.

```
DEFINE QREMOTE ('Reply_to') RNAME ('Answer')
RQMNAME ('QM1_relief')
```

3. The messages are sent with a message descriptor showing ReplyToQ='Answer' and ReplyToQMgr='QM1_relief'.
4. The application specification must include the information that replies are to be found in queue Answer rather than Reply_to.

To prepare for the replies you have to create the parallel return channel, defining:

- At QM2, the transmission queue named QM1_relief

```
DEFINE QLOCAL ('QM1_relief') USAGE(XMITQ)
```

- At QM1, the queue manager alias QM1_relief

```
DEFINE QREMOTE ('QM1_relief') RNAME() RQMNAME(QM1)
```

This queue manager alias terminates the chain of parallel return channels and captures the messages for QM1.

If you think you might want to do this at sometime in the future, ensure applications use the alias name from the start. For now this is a normal queue alias to the reply-to queue, but later it can be changed to a queue manager alias.

Reply-to queue name

Care is needed with naming reply-to queues. The reason that an application puts a reply-to queue name in the message is that it can specify the queue to which its replies are sent. When you create a reply-to queue alias definition with this name, you cannot have the actual reply-to queue (that is, a local queue definition) with the same name. Therefore, the reply-to queue alias definition must contain a new queue name as well as the queue manager name, and the application specification must include the information that its replies are found in this other queue.

The applications now have to retrieve the messages from a different queue from the one they named as the reply-to queue when they put the original message.

Cluster components

Clusters are composed of queue managers, cluster repositories, cluster channels, and cluster queues.

See the following subtopics for information about each of the cluster components:

Related information:



Configuring a queue manager cluster

Comparison of clustering and distributed queuing

Setting up a new cluster

Cluster repository



A repository is a collection of information about the queue managers that are members of a cluster.

The repository information includes queue manager names, their locations, their channels, which queues they host, and other information. The information is stored in the form of messages on a queue called `SYSTEM.CLUSTER.REPOSITORY.QUEUE`. The queue is one of the default objects.  On Multiplatforms, it is defined when you create an IBM MQ queue manager.  On IBM MQ for z/OS, it is defined as part of queue manager customization.

Typically, two queue managers in a cluster hold a full repository. The remaining queue managers all hold a partial repository.

Full repository and partial repository

A queue manager that hosts a complete set of information about every queue manager in the cluster has a full repository. Other queue managers in the cluster have partial repositories containing a subset of the information in the full repositories.


A partial repository contains information about only those queue managers with which the queue manager needs to exchange messages. The queue managers request updates to the information they need, so that if it changes, the full repository queue manager sends them the new information. For much of the time, a partial repository contains all the information a queue manager needs to perform within the cluster. When a queue manager needs some additional information, it makes inquiries of the full repository and updates its partial repository. The queue managers use a queue called `SYSTEM.CLUSTER.COMMAND.QUEUE` to request and receive updates to the repositories. This queue is one of the default objects.  On Multiplatforms, it is defined when you create an IBM MQ queue manager.  On IBM MQ for z/OS, it is defined as part of queue manager customization.

Cluster queue manager

A cluster queue manager is a queue manager that is a member of a cluster.

A queue manager can be a member of more than one cluster. Each cluster queue manager must have a name that is unique throughout all the clusters of which it is a member.

A cluster queue manager can host queues, which it advertises to the other queue managers in the cluster. However, it does not have to do this. It can instead feed messages into queues hosted elsewhere in the cluster, and receive only responses that are directed explicitly to it.

 In IBM MQ for z/OS, a cluster queue manager can be a member of a queue-sharing group. In this case, it shares its queue definitions with other queue managers in the same queue-sharing group.

Cluster queue managers are autonomous. They have full control over queues and channels that they define. Their definitions cannot be modified by other queue managers (other than queue managers in the same queue-sharing group). Repository queue managers do not control the definitions in other queue managers in the cluster. They hold a complete set of all the definitions, for use when required. A cluster is a federation of queue managers.

After you create or alter a definition on a cluster queue manager, the information is sent to the full repository queue manager. Other repositories in the cluster are updated later.

Full Repository queue manager

A full repository queue manager is a cluster queue manager that holds a full representation of the cluster's resources. To ensure availability, set up two or more full repository queue managers in each cluster. Full repository queue managers receive information sent by the other queue managers in the cluster and update their repositories. They send messages to each other to be sure that they are both kept up to date with new information about the cluster.

Queue managers and repositories

Every cluster has at least one (preferably two) queue managers holding full repositories of information about the queue managers, queues, and channels in a cluster. These repositories also contain requests from the other queue managers in the cluster for updates to the information.

The other queue managers each hold a partial repository, containing information about the subset of queues and queue managers with which they need to communicate. The queue managers build up their partial repositories by making inquiries when they first need to access another queue or queue manager. They request that they are notified of any new information concerning that queue or queue manager.

Each queue manager stores its repository information in messages on a queue called `SYSTEM.CLUSTER.REPOSITORY.QUEUE`. The queue managers exchange repository information in messages on a queue called `SYSTEM.CLUSTER.COMMAND.QUEUE`.

Each queue manager that joins a cluster defines a cluster-sender, `CLUSDR`, channel to one of the repositories. It immediately learns which other queue managers in the cluster hold full repositories. From then on, the queue manager can request information from any of the repositories. When the queue manager sends information to the chosen repository, it also sends information to one other repository (if there is one).

A full repository is updated when the queue manager hosting it receives new information from one of the queue managers that is linked to it. The new information is also sent to another repository, to reduce the risk of it being delayed if a repository queue manager is out of service. Because all the information is sent twice, the repositories have to discard duplicates. Each item of information carries a sequence number, which the repositories use to identify duplicates. All repositories are kept in step with each other


by exchanging messages.

Cluster queues

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. A cluster queue can be advertised in more than one cluster by using a cluster namelist.

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 can be a queue that is shared by members of a queue-sharing group in IBM MQ for z/OS.

Related information:

Defining cluster queues

Comparison between shared queues and cluster queues

This information is designed to help you compare shared queues and cluster queues, and decide which might be more suitable for your system.

Channel Initiator costs

In cluster queues, messages are sent by channels, so allow for channel initiator costs in addition to application costs. There are costs in the network because channels get and put messages. These costs are not present with shared queues, which therefore use less processing power than cluster queues when moving messages between queue managers in a queue-sharing group.

Availability of messages

When putting to a queue, cluster queues send the message to one of the queue managers with active channels connected to your queue manager. On the remote queue manager, if applications used to process the messages are not working, the messages are not processed and wait until the applications start. Similarly, if a queue manager is shut down, any messages on the queue manager are not made available until the queue manager restarts. These instances show lower message availability than when using shared queues.

When using shared queues, any application in the queue-sharing group can get messages that are sent. If you shut down one queue manager in the queue-sharing group, messages are available to the other queue managers, providing higher message availability than when using cluster queues.

Capacity

A coupling facility is more expensive than a disk; therefore the cost of storing 1,000,000 messages in a local queue is lower than having a coupling facility with enough capacity to store the same number of messages.

Sending to other queue managers

Shared-queue messages are only available within a queue-sharing group. If you want to use a queue manager outside of the queue-sharing group, you must use channels. You can use clustering to workload balance between multiple remote distributed queue managers.

Workload balancing

You can use clustering to give weight to which channels and queue managers get a proportion of the messages sent. For example, you can send 60% of messages to one queue manager, and 40% of messages to another queue manager. This instance does not depend on the ability of the remote queue manager to process work. The system with the first queue manager might be overloaded, and the system with the second queue manager might be idle, but most of the messages still go to the first queue manager.

With shared queues, two CICS systems can get messages. If one system is overloaded, the other system takes over most the workload.

Cluster channels

On every full repository, you manually define a cluster-receiver channel, and a set of cluster-sender channels to connect to every other full repository in the cluster. When you add a partial repository, you manually define a cluster-receiver channel, and a single cluster-sender channel that connects to one of the full repositories. Further cluster-sender channels are defined automatically by the cluster when needed. Auto-defined cluster-sender channels take their attributes from the corresponding cluster-receiver channel definition on the receiving queue manager.

Cluster-receiver channel: CLUSRCVR

A CLUSRCVR channel definition defines the end of a channel on which a cluster queue manager can receive messages from other queue managers in the cluster.

You must define at least one CLUSRCVR channel for each cluster queue manager. By defining the CLUSRCVR channel, the queue manager shows the other cluster queue managers that it is available to receive messages.

A CLUSRCVR channel definition also enables other queue managers to auto-define corresponding cluster-sender channel definitions. See the “Auto-defined cluster-sender channels” on page 201 section of this article.

Cluster-sender channel: CLUSSDR

You manually define a CLUSSDR channel from every full repository queue manager to every other full repository queue manager in the cluster. All the updates exchanged by the full repositories flow exclusively on these channels. By manually defining these channels, you control the network of full repositories explicitly.

When you add a partial repository queue manager to a cluster, you manually define a single CLUSSDR channel to connect to one of the full repositories. It makes little difference which full repository you choose, because after the initial contact has been made, further cluster queue manager objects for your queue manager, including CLUSSDR channels, are defined automatically as necessary. This enables your queue manager to send cluster information to any full repository, and send messages to any queue manager in the cluster.

As is explained in the section of this article, auto-defined sender channels are based on the configuration of the cluster-receiver channel. Therefore any channel properties you set on cluster channels should be set identically on matching CLUSSDR and cluster-receiver channels, or only set on cluster-receiver channels.

You should only manually define CLUSSDR channels for the reasons previously described. That is, to initially connect a partial repository to a full repository, or to connect two full repositories together. Manually configuring a CLUSSDR channel that connects to a partial repository, or to a queue manager not in the cluster, causes error messages such as AMQ9427 and AMQ9428 to be issued. Although this might sometimes be unavoidable as a temporary situation, for example when modifying the location of a full repository, the manual definition should be deleted as soon as possible.

Auto-defined cluster-sender channels

Typically when you add a partial repository queue manager to a cluster you manually define only two cluster channels on the queue manager:

- A cluster-sender (`CLUSSDR`) channel to a full repository queue manager for the cluster.
- A cluster receiver (`CLUSRCVR`) channel.

The `CLUSSDR` channel you define lets the queue manager make initial contact with the cluster. After the initial contact has been made, further `CLUSSDR` channels are defined automatically by the cluster when needed.

An auto-defined `CLUSSDR` channel takes its attributes from the corresponding `CLUSRCVR` channel definition on the receiving queue manager. Even if there is a manually defined `CLUSSDR` channel, the attributes from the auto-defined `CLUSSDR` channel are used. Suppose, for example that you define a `CLUSRCVR` channel without specifying a port number in the `CONNNAME` parameter, and manually define a `CLUSSDR` channel that does specify a port number. When the auto-defined `CLUSSDR` channel replaces the manually defined one, the port number (taken from the `CLUSRCVR` channel) becomes blank. The default port number is used and the channel fails.

Where there are configuration differences between a manually defined `CLUSSDR` channel and the corresponding `CLUSRCVR` channel definition, some differences take effect immediately (for example, the workload balancing parameters), and some take effect only on channel restart (for example, TLS configuration).

To avoid confusion, as far as possible observe the following guidelines:

- Only manually define `CLUSSDR` channels to point to full repositories.
- Where you do have manually defined `CLUSSDR` channels, configure them to identically match the corresponding `CLUSRCVR` channel definition on the receiving queue manager.

See also [Working with auto-defined channels](#).

Related information:

[Setting up a new cluster](#)

[Adding a queue manager to a cluster](#)

[Working with auto-defined channels](#)

[Working with cluster transmission queues and cluster-sender channels](#)

Cluster topics

Cluster topics are administrative topics with the `cluster` attribute defined. Information about cluster topics is pushed to all members of a cluster, and combined with local topics to create portions of a topic space that spans multiple queue managers. This enables messages published on a topic on one queue manager to be delivered to subscriptions of other queue managers in the cluster.

When you define a cluster topic on a queue manager, the cluster topic definition is sent to the full repository queue managers. The full repositories then propagate the cluster topic definition to all queue managers within the cluster, making the same cluster topic available to publishers and subscribers at any queue manager in the cluster. The queue manager on which you create a cluster topic is known as a cluster topic host. The cluster topic can be used by any queue manager in the cluster, but any modifications to a cluster topic must be made on the queue manager where that topic is defined (the host) at which point the modification is propagated to all members of the cluster through the full repositories.

For information about configuring cluster topics to use *direct routing* or *topic host routing*, and about clustered topic inheritance and wildcard subscriptions, see [Defining cluster topics](#).

For information about the commands to use to display cluster topics, see the related information.

Related information:

Working with administrative topics



Working with subscriptions

DISPLAY TOPIC

DISPLAY TPSTATUS

DISPLAY SUB

Default cluster objects

 On Multiplatforms, the default cluster objects are included in the set of default objects automatically created when you define a queue manager.  On z/OS, the default cluster object definitions can be found in the customization samples.

Note: You can alter the default channel definitions in the same way as any other channel definition, by running MQSC or PCF commands. Do not alter the default queue definitions, except for `SYSTEM.CLUSTER.HISTORY.QUEUE`.

SYSTEM.CLUSTER.COMMAND.QUEUE

Each queue manager in a cluster has a local queue called `SYSTEM.CLUSTER.COMMAND.QUEUE` which is used to transfer messages to the full repository. The message contains any new or changed information about the queue manager, or any requests for information about other queue managers. `SYSTEM.CLUSTER.COMMAND.QUEUE` is normally empty.

SYSTEM.CLUSTER.HISTORY.QUEUE

Each queue manager in a cluster has a local queue called `SYSTEM.CLUSTER.HISTORY.QUEUE`. `SYSTEM.CLUSTER.HISTORY.QUEUE` is used to store the history of cluster state information for service purposes.

In the default object settings, `SYSTEM.CLUSTER.HISTORY.QUEUE` is set to `PUT (ENABLED)`. To suppress history collection change the setting to `PUT (DISABLED)`.

SYSTEM.CLUSTER.REPOSITORY.QUEUE

Each queue manager in a cluster has a local queue called `SYSTEM.CLUSTER.REPOSITORY.QUEUE`. This queue is used to store all the full repository information. This queue is not normally empty.

SYSTEM.CLUSTER.TRANSMIT.QUEUE

Each queue manager has a definition for a local queue called `SYSTEM.CLUSTER.TRANSMIT.QUEUE`. `SYSTEM.CLUSTER.TRANSMIT.QUEUE` is the default transmission queue for all messages to all queues and queue managers that are within clusters. You can change the default transmission queue for each cluster-sender channel to `SYSTEM.CLUSTER.TRANSMIT.ChannelName`, by changing the queue manager attribute `DEFCLXQ`. You cannot delete `SYSTEM.CLUSTER.TRANSMIT.QUEUE`. It is also used to define authorization checks whether the default transmission queue that is used is `SYSTEM.CLUSTER.TRANSMIT.QUEUE` or `SYSTEM.CLUSTER.TRANSMIT.ChannelName`.

SYSTEM.DEF.CLUSRCVR

Each cluster has a default `CLUSRCVR` channel definition called `SYSTEM.DEF.CLUSRCVR`. `SYSTEM.DEF.CLUSRCVR` is used to supply default values for any attributes that you do not specify when you create a cluster-receiver channel on a queue manager in the cluster.

SYSTEM.DEF.CLUSSDR

Each cluster has a default `CLUSSDR` channel definition called `SYSTEM.DEF.CLUSSDR`. `SYSTEM.DEF.CLUSSDR` is used to supply default values for any attributes that you do not specify when you create a cluster-sender channel on a queue manager in the cluster.

Related information:

Working with default cluster objects

Publish/subscribe messaging

Publish/subscribe messaging allows you to 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.

Before a point-to-point IBM MQ application can send a message to another application, it needs to know something about that application. For example, it needs to know the name of the queue to which to send the information, and might also specify a queue manager name.

IBM MQ publish/subscribe removes the need for your application to know anything about the target application. All the sending application has to do is this:

- *Put* an IBM MQ message that contains the information that the application wants.
- Assign the message to a topic that denotes the subject of the information.
- Let IBM MQ handle the distribution of that information.

Similarly, the target application does not have to know anything about the source of the information it receives.

The following figure shows the simplest publish/subscribe system. There is one publisher, one queue manager, and one subscriber. A subscription is made by the subscriber on a queue manager, a publication is sent from the publisher to the queue manager, and the publication is then forwarded by the queue manager to the subscriber.

A typical publish/subscribe system has more than one publisher and more than one subscriber on many

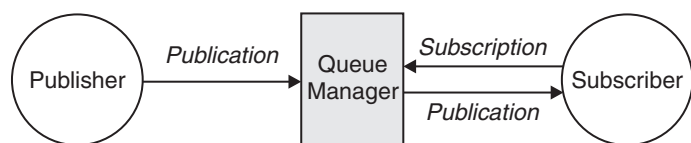


Figure 17. Simple publish/subscribe configuration

different topics, and often has more than one queue manager. An application can be both a publisher and a subscriber.

Another significant difference between publish/subscribe messaging and point-to-point is that a message sent to a point-to-point queue is only processed by a single consuming application. A message published to a publish/subscribe topic, where more than one subscriber has registered an interest, is processed by every interested subscriber.

Publish/subscribe components

Publish/subscribe is the mechanism by which subscribers can receive information, in the form of messages, from publishers. The interactions between publishers and subscribers are controlled by queue managers, using standard IBM MQ facilities.

A typical publish/subscribe system has more than one publisher and more than one subscriber on many different topics, and often has more than one queue manager. An application can be both a publisher and a subscriber.

The provider of information is called a *publisher*. Publishers supply information about a subject, without needing to know anything about the applications that are interested in that information. Publishers generate this information in the form of messages, called *publications* that they want to publish and define the topic of these messages.

The consumer of the information is called a *subscriber*. Subscribers create *subscriptions* that describe the topic that the subscriber is interested in. Thus, the subscription determines which publications are forwarded to the subscriber. Subscribers can make multiple subscriptions and can receive information from many different publishers.

Published information is sent in an IBM MQ message, and the subject of the information is identified by its *topic*. The publisher specifies the topic when it publishes the information, and the subscriber specifies the topics about which it wants to receive publications. The subscriber is sent information about only those topics it subscribes to.

It is the existence of topics that allows the providers and consumers of information to be decoupled in publish/subscribe messaging by removing the need to include a specific destination in each message as is required in point-to-point messaging.

Interactions between publishers and subscribers are all controlled by a queue manager. The queue manager receives messages from publishers, and subscriptions from subscribers (to a range of topics). The queue manager's job is to route the published messages to the subscribers that have registered an interest in the topic of the messages.

Standard IBM MQ facilities are used to distribute messages, so your applications can use all the features that are available to existing IBM MQ applications. This means that you can use persistent messages to get once-only assured delivery, and that your messages can be part of a transactional unit-of-work to ensure that messages are delivered to the subscriber only if they are committed by the publisher.

Publishers and publications

In IBM MQ publish/subscribe a publisher is an application that makes information about a specified topic available to a queue manager in the form of a standard IBM MQ message called a publication. A publisher can publish information about more than one topic.

Publishers use the MQPUT verb to put a message to a previously opened topic, this message is a publication. The local queue manager then routes the publication to any subscribers who have subscriptions to the topic of the publication. A published message can be consumed by more than one subscriber.

In addition to distributing publications to all local subscribers that have appropriate subscriptions, a queue manager can also distribute the publication to any other queue managers connected to it, either directly or through a network of queue managers that have subscribers to the topic.

In an IBM MQ publish/subscribe network, a publishing application can also be a subscriber.

Publications under syncpoint

Publishers can issue MQPUT or MQPUT1 calls in syncpoint to include all messages delivered to subscribers in a unit of work. If the MQPMO_RETAIN option, or topic delivery options NPMSGDLV or PMSGDLV with values ALL or ALLDUR are specified, the queue manager uses internal MQPUT or MQPUT1 calls in syncpoint, within the scope of the publisher MQPUT or MQPUT1 call.

State and event information:

Publications can be categorized as either state publications, such as the current price of a stock, or event publications, such as a trade in that stock.

State publications

State publications contain information about the current state of something, such as the price of stock or the current score in a soccer match. When something happens (for example, the stock price changes or the soccer score changes), the previous state information is no longer required because it is superseded by the new information.

A subscriber will want to receive the current version of the state information when it starts, and be sent new information whenever the state changes.

If a publication contains state information, it is often published as a retained publication. A new subscriber typically wants the current state information immediately; the subscriber does not want to wait for an event that causes the information to be republished. Subscribers will automatically receive a topic's retained publication when it subscribes unless the subscriber uses the MQSO_PUBLICATIONS_ON_REQUEST or MQSO_NEW_PUBLICATIONS_ONLY options.

Event publications

Event publications contain information about individual events that occur, such as a trade in some stock or the scoring of a particular goal. Each event is independent of other events.

A subscriber will want to receive information about events as they happen.

Retained publications:

By default, after a publication is sent to all interested subscribers it is discarded. However, a publisher can specify that a copy of a publication is retained so that it can be sent to future subscribers who register an interest in the topic.

Deleting publications after they have been sent to all interested subscribers is suitable for event information, but is not always suitable for state information. By retaining a message, new subscribers do not have to wait for information to be published again before they receive initial state information. For example, a subscriber with a subscription to a stock price would receive the current price straight away, without waiting for the stock price to change (and hence be republished).

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. However, the deletion of the existing publication might not occur synchronously with the arrival of the new retained publication. Therefore, wherever possible, have no more than one publisher sending retained publications on any topic.

Subscribers can specify that they do not want to receive retained publications by using the MQSO_NEW_PUBLICATIONS_ONLY subscription option. Existing subscribers can ask for duplicate copies of retained publications to be sent to them.

There are times when you might not want to retain publications, even for state information:

- If all subscriptions to a topic are made before any publications are made on that topic, and you do not expect, or do not allow, new subscriptions, there is no need to retain publications because they are delivered to the complete set of subscribers the first time they are published.
- If publications occur frequently, such as every second, a new subscriber (or a subscriber recovering from a failure) receives the current state almost immediately after their initial subscription, so there is no need to retain these publications.
- If the publications are large, you might end up needing a considerable amount of storage space to store the retained publication for each topic. In a multiple queue manager environment, retained publications are stored by all queue managers in the network that have a matching subscription.

When deciding whether to use retained publications, consider how subscribing applications recover from a failure. If the publisher does not use retained publications, the subscriber application might need to store its current state locally.

To ensure that a publication is retained, use the MQPMO_RETAIN put-message option. If this option is used and the publication cannot be retained, the message is not published and the call fails with MQRC_PUT_NOT_RETAINED.

If a message is a retained publication, this is indicated by the MQIsRetained message property. The persistence of a message is as it was when it was originally published.

Related information:

Design considerations for retained publications in publish/subscribe clusters

Publications under syncpoint:

In IBM MQ publish/subscribe, syncpoint can be used by publishers or internally by the queue manager.

Publishers use syncpoint when they issue MQPUT/MQPUT1 calls with the MQPMO_SYNCPOINT option. All messages delivered to subscribers count towards the maximum number of uncommitted messages in a unit of work. The MAXUMSGS queue manager attribute specifies this limit. If the limit is reached then the publisher receives the 2024 (07E8) (RC2024): MQRC_SYNCPOINT_LIMIT_REACHED reason code.

When a publisher issues MQPUT/MQPUT1 calls using MQPMO_NO_SYNCPOINT with the MQPMO_RETAIN option, or topic delivery options NPMSGDLV/PMSGDLV with values ALL or ALLDUR, the queue manager uses internal syncpoints to guarantee that messages are delivered as requested. The publisher can receive the 2024 (07E8) (RC2024): MQRC_SYNCPOINT_LIMIT_REACHED reason code if the limit is reached within the scope of the publisher MQPUT/MQPUT1 call.

Subscribers and subscriptions

In IBM MQ publish/subscribe, a subscriber is an application that requests information about a specific topic from a queue manager in a publish/subscribe network. A subscriber can receive messages, about the same or different topics, from more than one publisher.

Subscriptions can be created manually using an MQSC command or by applications. These subscriptions are issued to the local queue manager and contain information about the publications the subscriber wants to receive:

- The topic the subscriber is interested in; this can resolve to multiple topics if wildcards are used.
- An optional selection string to be applied to published messages.
- A handle to a queue (known as the *subscriber queue*), on which selected publications should be placed, and the optional CorrelId.

The local queue manager stores subscription information and when it receives a publication, scans the information to determine whether there is a subscription that matches the publication's topic and selection string. For each matching subscription, the queue manager directs the publication to the subscriber's subscriber queue. The information that a queue manager stores about subscriptions can be viewed by using the DIS SUB and DIS SBSTATUS commands.

A subscription is deleted only when one of the following events occurs:

- The subscriber unsubscribes using the MQCLOSE call (if the subscription was made non-durably).
- The subscription expires.
- The subscription is deleted by the system administrator using the DELETE SUB command.
- The subscriber application ends (if the subscription was made non-durably).
- The queue manager is stopped or restarted (if the subscription was made non-durably).

Related information:

Cloned and shared subscriptions

Examples of how to define the sharedSubscription property

Managed queues and publish/subscribe:

When you create a subscription you can choose to use managed queuing. If you use managed queuing a subscription queue is automatically created when you create a subscription. Managed queues are tidied up automatically in accordance with the durability of the subscription. Using managed queues means that you do not have to worry about creating queues to receive publications and any unconsumed publications are removed from subscriber queues automatically if a non-durable subscription connection is closed.

If an application has no need to use a particular queue as its subscriber queue, the destination for the publications it receives, it can make use of *managed subscriptions* using the MQSO_MANAGED subscription option. If you create a managed subscription, the queue manager returns an object handle to the subscriber for a subscriber queue that the queue manager creates where publications will be received. The queue's object handle will be returned allowing you to browse, get or inquire on the queue (it is not possible to put to or set attributes of a managed queue unless you have been explicitly given access to temporary dynamic queues).

The durability of the subscription determines whether the managed queue remains when the subscribing application's connection to the queue manager is broken.

Managed subscriptions are particularly useful when used with non-durable subscriptions because when the application's connection is ended, unconsumed messages would otherwise remain on the subscriber queue taking up space in your queue manager indefinitely. If you are using a managed subscription, the managed queue will be a temporary dynamic queue and as such will be deleted along with any unconsumed messages when the connection is broken for any of the following reasons:

- MQCLOSE with MQCO_REMOVE_SUB is used and the managed Hobj is closed.
- a connection is lost to an application using a non-durable subscription (MQSO_NON_DURABLE).
- a subscription is removed because it has expired and the managed Hobj is closed.

Managed subscriptions can also be used with durable subscriptions but it is possible that you would want to leave unconsumed messages on the subscriber queue so that they can be retrieved when the connection is reopened. For this reason, managed queues for durable subscriptions take the form of a permanent dynamic queue and will remain when the subscribing application's connection to the queue manager is broken.

You can set an expiry on your subscription if you want to use a permanent dynamic managed queue so that although the queue will still exist after the connection is broken, it will not continue to exist indefinitely.

If you delete the managed queue you will receive an error message.

The managed queues that are created are named with numbers at the end (timestamps) so that each is unique.

Subscription durability:

Subscriptions can be configured to be durable or non-durable. Subscription durability determines what happens to subscriptions when subscribing applications disconnect from a queue manager.

Durable subscriptions

Durable subscriptions continue to exist when a subscribing application's connection to the queue manager is closed. If a subscription is durable, when the subscribing application disconnects, the subscription remains in place and can be used by the subscribing application when it reconnects requesting the subscription again using the **SubName** that was returned when the subscription was created.

When subscribing durably, a subscription name (**SubName**) is required. Subscription names must be unique within a queue manager so that it can be used to identify a subscription. This means of identification is necessary when specifying a subscription you want to resume, if you have either deliberately closed the connection to the subscription (using the MQCO_KEEP_SUB option) or have been disconnected from the queue manager. You can resume an existing subscription by using the MQSUB call with the MQSO_RESUME option. Subscription names are also displayed if you use the DISPLAY SBSTATUS command with SUBTYPE ALL or ADMIN.

When an application no longer requires a durable subscription it can be removed using the MQCLOSE function call with the MQCO_REMOVE_SUB option or it can be deleted manually use the MQSC command DELETE SUB.

You can use the **DURSUB** topic attribute to specify whether or not durable subscriptions can be made to a topic.

On return from an MQSUB call using the MQSO_RESUME option, subscription expiry is set to the original expiry of the subscription and not the remaining expiry time.

A queue manager continues to send publications to satisfy a durable subscription even if that subscriber application is not connected. This leads to a build up of messages on the subscriber queue. The easiest way to avoid this problem is to use a non-durable subscription wherever appropriate. However, where it is necessary to use durable subscriptions, a build up of messages can be avoided if the subscriber subscribes using the Retained publications option. A subscriber can then control when it receives publications by using the MQSUBRQ call.

Non-durable subscriptions

Non-durable subscriptions exist only as long as the subscribing application's connection to the queue manager remains open. The subscription is removed when the subscribing application disconnects from the queue manager either deliberately or by loss of connection. When the connection is closed, the information about the subscription is removed from the queue manager, and is no longer shown if you display subscriptions using the `DISPLAY SBSTATUS` command. No more messages are put to the subscriber queue.

What happens to any unconsumed publications on the subscriber queue for non-durable subscriptions is determined as follows.

- If a subscribing application is using a managed destination, any publications that have not been consumed are automatically removed.
- If the subscribing application provides a handle to its own subscriber queue when it subscribes, unconsumed messages are not removed automatically. It is the responsibility of the application to clear the queue if that is appropriate. If the queue is shared by more than one subscriber, or other point-to-point applications, it might not be appropriate to clear the queue completely.

Although not required for non-durable subscriptions, a subscription name if provided, is used by the queue manager. Subscription names must be unique within the queue manager so that it can be used to identify a subscription.

Related information:

Cloned and shared subscriptions

Using JMS 2.0 shared subscriptions

Examples of how to define the `sharedSubscription` property

Selection strings:

A *selection string* is an expression that is applied to a publication to determine whether it matches a subscription. Selection strings can include wildcard characters.

When you subscribe, in addition to specifying a topic, you can specify a selection string to select publications according to their message properties.

The selection string is evaluated against the message as put by the publisher before it is modified for delivery to each subscriber. Take care when using fields in the selection string that might be modified as part of the publish operation. For example, the MQMD fields `UserIdentifier`, `MsgId`, and `CorrelId`.

Selection strings should not reference any of the message property fields added by the queue manager as part of the publish operation (see `Publish/subscribe message properties`), except for the message property `MQTopicString`, which contains the topic string for the publication.

Related information:

Selection string rules and restrictions

Topics

A topic is the subject of the information that is published in a publish/subscribe message.

Messages in point-to-point systems are sent to a specific destination address. Messages in subject-based publish/subscribe systems are sent to subscribers based on the subject that describes the contents of the message. In content-based systems, messages are sent to subscribers based on the contents of the message itself.

The IBM MQ publish/subscribe system is a subject-based publish/subscribe system. A publisher creates a message, and publishes it with a topic string that best fits the subject of the publication. To receive publications, a subscriber creates a subscription with a pattern matching topic string to select publication topics. The queue manager delivers publications to subscribers that have subscriptions that match the publication topic, and are authorized to receive the publications. The article, "Topic strings" on page 211, describes the syntax of topic strings that identify the subject of a publication. Subscribers also create topic strings to select which topics to receive. The topic strings that subscribers create can contain either of two alternative wildcard schemes to pattern match against the topic strings in publications. Pattern matching is described in "Wildcard schemes" on page 212.

In subject-based publish/subscribe, publishers, or administrators, are responsible for classifying subjects into topics. Typically subjects are organized hierarchically, into topic trees, using the '/' character to create subtopics in the topic string. See "Topic trees" on page 218 for examples of topic trees. Topics are nodes in the topic tree. Topics can be leaf-nodes with no further subtopics, or intermediate nodes with subtopics.

In parallel with organizing subjects into a hierarchical topic tree, you can associate topics with administrative topic objects. You assign attributes to a topic, such as whether the topic is distributed in a cluster, by associating it with an administrative topic object. The association is made by naming the topic using the TOPICSTR attribute of the administrative topic object. If you do not explicitly associate an administrative topic object to a topic, the topic inherits the attributes of its closest ancestor in the topic tree that you *have* associated with an administrative topic object. If you have not defined any parent topics at all, it inherits from SYSTEM.BASE.TOPIC. Administrative topic objects are described in "Administrative topic objects" on page 219.

Note: Even if you inherit all the attributes of a topic from SYSTEM.BASE.TOPIC, define a root topic for your topics that directly inherits from SYSTEM.BASE.TOPIC. For example, in the topic space of US states, USA/Alabama USA/Alaska, and so on, USA is the root topic. The main purpose of the root topic is to create discrete, non-overlapping topic spaces to avoid publications matching the wrong subscriptions. It also means you can change the attributes of your root topic to affect your whole topic space. For example, you might set the name for the **CLUSTER** attribute.

When you refer to a topic as a publisher or subscriber, you have a choice of supplying a topic string, or referring to a topic object. Or you can do both, in which case the topic string you supply defines a subtopic of the topic object. The queue manager identifies the topic by appending the topic string to the topic string prefix named in the topic object, inserting an additional '/' in between the two topic strings, for example, *topic string/object string*. "Combining topic strings" on page 216 describes this further. The resulting topic string is used to identify the topic and associate it with an administrative topic object. The administrative topic object is not necessarily the same topic object as the topic object corresponding to the master topic.

In content based publish/subscribe, you define what messages you want to receive by providing selection strings that search the contents of every message. IBM MQ provides an intermediate form of content based publish/subscribe using message selectors that scan message properties rather than the full content of the message, see Selectors. The archetypal use of message selectors is to subscribe to a topic

and then qualify the selection with a numeric property. The selector enables you to specify you are interested in values only in a certain range; something you cannot do using either character or topic-based wildcards. If you do need to filter based on the full content of the message, you need to use IBM Integration Bus.

Topic strings:

Label information you publish as a topic using a topic string. Subscribe to groups of topics using either character or topic based wildcard topic strings.

Topics

A *topic string* is a character string that identifies the topic of a publish/subscribe message. You can use any characters you like when you construct a topic string.

Topic string



Three characters have special meaning in Version 7 publish/subscribe. They are allowed anywhere in a topic string, but use them with caution. The use of the special characters is explained in “Topic-based wildcard scheme” on page 212.

A forward slash (/)

The topic level separator. Use the '/' character to structure the topic into a topic tree.

Avoid empty topic levels, '//', if you can. These correspond to nodes in the topic hierarchy with no topic string. A leading or trailing '/' in a topic string corresponds to a leading or trailing empty node and should be avoided too.

The hash sign (#)

Used in combination with '/' to construct a multilevel wildcard in subscriptions. Take care using '#' adjacent to '/' in topic strings used to name published topics. “Examples of topic strings” shows a sensible use of '#'.

The strings '.../#/...', '#/...' and '.../#' have a special meaning in subscription topic strings. The strings match all topics at one or more levels in the topic hierarchy. Thus if you created a topic with one of those sequences, you could not subscribe to it, without also subscribing to all topics at multiple levels in the topic hierarchy.

The plus sign (+)

Used in combination with '/' to construct a single-level wildcard in subscriptions. Take care using '+' adjacent to '/' in topic strings used to name published topics.

The strings '.../+/...', '+/...' and '.../+' have a special meaning in subscription topic strings. The strings match all topics at one level in the topic hierarchy. Thus if you created a topic with one of those sequences, you could not subscribe to it, without also subscribing to all topics at one level in the topic hierarchy.

Examples of topic strings

IBM/Business Area#/Results
IBM/Diversity/%African American

Related information:

TOPIC

Wildcard schemes:

There are two wildcard schemes used to subscribe to multiple topics. The choice of scheme is a subscription option.

MQSO_WILDCARD_TOPIC

Select topics to subscribe to using the topic-based wildcard scheme.

This is the default if no wildcard schema is explicitly selected.

MQSO_WILDCARD_CHAR

Select topics to subscribe to using the character-based wildcard scheme.

Set either scheme by specifying the **wschema** parameter on the DEFINE SUB command. For more information, see DEFINE SUB.

Note: Subscriptions that were created before IBM WebSphere MQ Version 7.0 always use the character-based wildcard scheme.

Examples

IBM+/Results
#/Results
IBM/Software/Results
IBM/*ware/Results

Topic-based wildcard scheme:

Topic-based wildcards allow subscribers to subscribe to more than one topic at a time.

Topic-based wildcards are a powerful feature of the topic system in IBM MQ publish/subscribe. The multilevel wildcard and single level wildcard can be used for subscriptions, but they cannot be used within a topic by the publisher of a message.

The topic-based wildcard scheme allows you to select publications grouped by topic level. You can choose, for each level in the topic hierarchy, whether the string in the subscription for that topic level must exactly match the string in the publication or not. For example, the subscription IBM+/Results selects all the topics,

IBM/Software/Results
IBM/Services/Results
IBM/Hardware/Results

There are two types of wildcard.

Multilevel wildcard

- The multilevel wildcard is used in subscriptions. When used in a publication it is treated as a literal.
- The multilevel wildcard character '#' is used to match any number of levels within a topic. For example, using the example topic tree, if you subscribe to 'USA/Alaska/#', you receive messages on topics 'USA/Alaska' and 'USA/Alaska/Juneau'.
- The multilevel wildcard can represent zero or more levels. Therefore, 'USA/#' can also match the singular 'USA', where '#' represents zero levels. The topic level separator is meaningless in this context, because there are no levels to separate.
- The multilevel wildcard is only effective when specified on its own or next to the topic level separator character. Therefore, '#' and 'USA/#' are valid topics where the '#' character is

treated as a wildcard. However, although 'USA#' is also a valid topic string, the '#' character is not regarded as a wildcard and does not have any special meaning. See “When topic-based wildcards are not wild” on page 215 for more information.

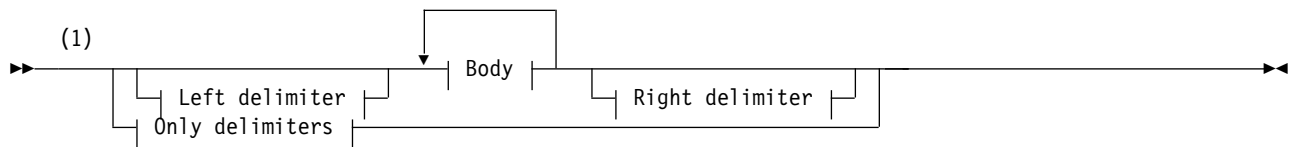
Single level wildcard

- The single wildcard is used in subscriptions. When used in a publication it is treated as a literal.
- The single-level wildcard character '+' matches one, and only one, topic level. For example, 'USA/+' matches 'USA/Alabama', but not 'USA/Alabama/Auburn'. Because the single-level wildcard matches only a single level, 'USA/+' does not match 'USA'.
- The single-level wildcard can be used at any level in the topic tree, and in conjunction with the multilevel wildcard. The single-level wildcard must be specified next to the topic level separator, except when it is specified on its own. Therefore, '+' and 'USA/+' are valid topics where the '+' character is treated as a wildcard. However, although 'USA+' is also a valid topic string, the '+' character is not regarded as a wildcard and does not have any special meaning. See “When topic-based wildcards are not wild” on page 215 for more information.

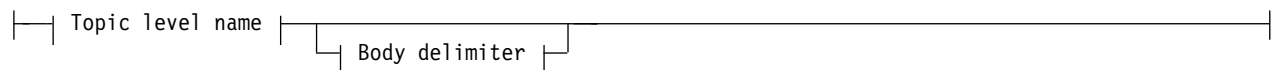
The syntax for the topic-based wildcard scheme has no escape characters. Whether '#' and '+' are treated as wildcards or not depends on their context. See “When topic-based wildcards are not wild” on page 215 for more information.

Note: The beginning and end of a topic string is treated in a special way. Using '\$' to denote the end of the string, then '\$#/...' is a multilevel wildcard, and '\$#/..' is an empty node at the root, followed by a multilevel wildcard.

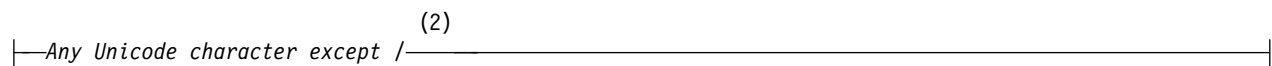
Topic-based wildcard string



Body:



Topic level name:



Only delimiters:



Notes:

- 1 A null or zero length topic string is invalid
- 2 You are advised not to use any of *, ?, % in level name strings for compatibility between char based and topic-based wildcard schemes.
- 3 These cases are equivalent to the *left delimiter* pattern.
- 4 / with no wildcards matches a single empty topic.
- 5 These cases are equivalent to the *right delimiter* pattern.
- 6 Match every topic.
- 7 Match every topic where there is only one level.

Left delimiter:



Body delimiter:



Right delimiter:



Notes:

- 1 The topic string starts with an empty topic
- 2 Matches zero or more levels. Multiple multi-level match strings have the same effect as one multi-level match string.
- 3 Matches exactly one level.
- 4 // is an empty topic - a topic object with no topic string.
- 5 The topic string ends with an empty topic

When topic-based wildcards are not wild

The wildcard characters '+' and '#' have no special meaning when they are mixed with other characters (including themselves) in a topic level.

This means that topics that contain '+' or '#' together with other characters in a topic level can be published.

For example, consider the following two topics:

1. level0/level1/+/level4/#
2. level0/level1/#+/level4/level#

In the first example, the characters '+' and '#' are treated as wildcards and are therefore not valid in a topic string that is to be published to but are valid in a subscription.

In the second example, the characters '+' and '#' are not treated as wildcards and therefore the topic string can be both published and subscribed to.

Examples

```
IBM+/Results
#/Results
IBM/Software/Results
```

Character-based wildcard scheme:

The character-based wildcard scheme allows you to select topics based on traditional character matching.

You can select all topics at multiple levels in a topic hierarchy using the string '*'. Using '*' in the character-based wildcard scheme is equivalent to using the topic-based wildcard string '#'

'x*/y' is equivalent to 'x#/y' in the topic-based scheme, and selects all topics in the topic hierarchy between levels 'x' and 'y', where 'x' and 'y' are topic names that are not in the set of levels returned by the wildcard.

'+/' in the topic-based scheme has no exact equivalent in the character-based scheme. 'IBM*/Results' would also select 'IBM/Patents/Software/Results'. Only if the set of topic names at each level of the hierarchy are unique, can you always construct queries with the two schemes that yield identical matches.

Used in a general way, '*' and '?' in the character-based scheme have no equivalents in the topic-based scheme. The topic-based scheme does not perform partial matching using wildcards. The character based wildcard subscription 'IBM*ware/Results' has no topic-based equivalent.

Note: Matches using character wildcard subscriptions are slower than matches using topic-based subscriptions.

Character-based wildcard string



V6 literal:

—Any Unicode character except *, ? and %—

Notes:

- 1 Means "Escape the following character", so that it is treated as a literal. '%' must be followed by either '*', '?' or '%'. See "Examples of topic strings" on page 211.
- 2 Means "Match zero or more characters" in a subscription.
- 3 Means "Match exactly one character" in a subscription.

Examples

IBM/*/Results
IBM/*ware/Results

Combining topic strings:

When creating subscriptions, or opening topics so you can publish messages to them, the topic string can be formed by combining two separate sub-topic strings, or "subtopics". One subtopic is provided by the application or administrative command as a topic string, and the other is the topic string associated with a topic object. You can use either subtopic as the topic string on its own, or combine them to form a new topic name.

For example, when you define a subscription using the MQSC command **DEFINE SUB**, the command can take either **TOPICSTR** (topic string) or **TOPICOBJ** (topic object) as an attribute, or both together. If only **TOPICOBJ** is provided, the topic string associated with that topic object is used as the topic string. If only **TOPICSTR** is provided, that is used as the topic string. If both are provided, they are concatenated to form a single topic string in the form **TOPICOBJ / TOPICSTR**, where the **TOPICOBJ** configured topic string is always first and the two parts of the string are always separated by a "/" character.

Similarly, in an MQI program the full topic name is created by MQOPEN. It is composed of two fields used in publish/subscribe MQI calls, in the order listed:

1. The **TOPICSTR** attribute of the topic object, named in the **ObjectName** field.
2. The **ObjectString** parameter defining the subtopic provided by the application.

The resulting topic string is returned in the **ResObjectString** parameter.

These fields are considered to be present if the first character of each field is not a blank or null character, and the field length is greater than zero. If only one of the fields is present, it is used unchanged as the

topic name. If neither field has a value, the call fails with reason code MQRC_UNKNOWN_OBJECT_NAME, or MQRC_TOPIC_STRING_ERROR if the full topic name is not valid.

If both fields are present, a “/” character is inserted between the two elements of the resultant combined topic name.

Table 7 shows examples of topic string concatenation:

Table 7. Topic string concatenation examples

TOPICSTR of the topic object	Topic string provided by application or DEFINE SUB command	Full topic name	Comment
Football/Scores	' '	Football/Scores	The TOPICSTR of the topic object is used alone.
' '	Football/Scores	Football/Scores	The ObjectString/ TOPICSTR is used alone.
Football	Scores	Football/Scores	A “/” character is added at the concatenation point.
Football	/Scores	Football//Scores	An 'empty node' is produced between the two strings. This is different to “Football/Scores”.
/Football	Scores	/Football/Scores	The topic starts with an 'empty node'. This is different to “Football/Scores”.

The “/” character is considered as a special character, providing structure to the full topic name in “Topic trees” on page 218. The “/” character must not be used for any other reason, because the structure of the topic tree is affected. The topic “/Football” is not the same as the topic “Football”.

Note: If you use a topic object when creating a subscription, the value of the topic object topic string is fixed in the subscription at define time. Any subsequent change to the topic object does not affect the topic string that the subscription is defined to.

Wildcard characters in topic strings

The following wildcard characters are special characters:

- plus sign (+)
- number sign (#)
- asterisk (*)
- question mark (?)

Wildcard characters only have special meaning when used by a subscription. These characters are not considered as invalid when used elsewhere, however you must ensure you understand how they are used and you might prefer not to use these characters in your topic strings when publishing or defining topic objects.

If you publish on a topic string with # or + mixed in with other characters (including themselves) within a topic level, the topic string can be subscribed to with either wildcard scheme.

If you publish on a topic string with # or + as the only character between two / characters, the topic string cannot be subscribed to explicitly by an application using the wildcard scheme MQSO_WILDCARD_TOPIC. This situation results in the application getting more publications than expected.

You should not use a wildcard character in the topic string of a defined topic object. If you do this, the character is treated as a literal character when the object is used by a publisher, and as a wildcard character when used by a subscription. This can lead to confusion.

Example code snippet

This code snippet, extracted from the example program Example 2: Publisher to a variable topic, combines a topic object with a variable topic string:

```
MQOD td = {MQOD_DEFAULT}; /* Object Descriptor */
td.ObjectType = MQOT_TOPIC; /* Object is a topic */
td.Version = MQOD_VERSION_4; /* Descriptor needs to be V4 */
strncpy(td.ObjectName, topicName, MQ_TOPIC_NAME_LENGTH);
td.ObjectString.VSPtr = topicString;
td.ObjectString.VSLength = (MQLONG)strlen(topicString);
td.ResObjectString.VSPtr = resTopicStr;
td.ResObjectString.VSBufSize = sizeof(resTopicStr)-1;
MQOPEN(Hconn, &td, MQOO_OUTPUT | MQOO_FAIL_IF QUIESCING, &Hobj, &CompCode, &Reason);
```

Topic trees:

Each topic that you define is an element, or node, in the topic tree. The topic tree can either be empty to start with or contain topics that have been defined previously using MQSC or PCF commands. You can define a new topic either by using the create topic commands or by specifying the topic for the first time in a publication or subscription.

Although you can use any character string to define a topic's topic string, it is advisable to choose a topic string that fits into a hierarchical tree structure. Thoughtful design of topic strings and topic trees can help you with the following operations:

- Subscribing to multiple topics.
- Establishing security policies.

Although you can construct a topic tree as a flat, linear structure, it is better to build a topic tree in a hierarchical structure with one or more root topics. For more information about security planning and topics, see Publish/subscribe security.

Figure 18 shows an example of a topic tree with one root topic.

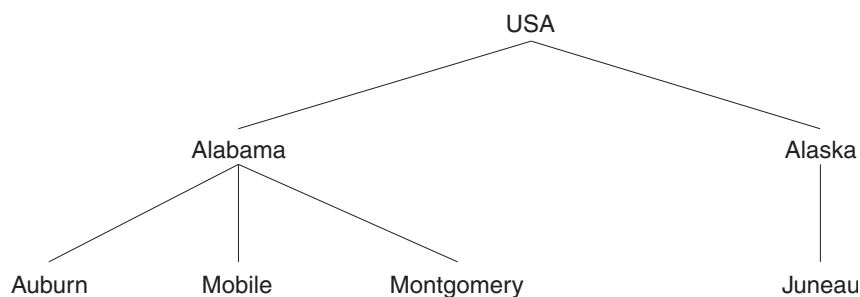


Figure 18. Example of a topic tree

Each character string in the figure represents a node in the topic tree. A complete topic string is created by aggregating nodes from one or more levels in the topic tree. Levels are separated by the "/" character. The format of a fully specified topic string is: "root/level2/level3".

The valid topics in the topic tree shown in Figure 18 on page 218 are:

"USA"
"USA/Alabama"
"USA/Alaska"
"USA/Alabama/Auburn"
"USA/Alabama/Mobile"
"USA/Alabama/Montgomery"
"USA/Alaska/Juneau"

When you design topic strings and topic trees, remember that the queue manager does not interpret, or attempt to derive meaning from, the topic string itself. It simply uses the topic string to send selected messages to subscribers of that topic.

The following principles apply to the construction and content of a topic tree:

- There is no limit to the number of levels in a topic tree.
- There is no limit to the length of the name of a level in a topic tree.
- There can be any number of "root" nodes; that is, there can be any number of topic trees.

Related information:

Reducing the number of unwanted topics in the topic tree

Administrative topic objects:

Using an administrative topic object, you can assign specific, non-default attributes to topics.

Figure 19 shows how a high-level topic of Sport divided into separate topics covering different sports can be visualized as a topic tree:

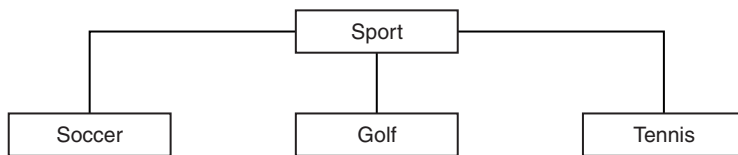


Figure 19. Visualization of a topic tree

Figure 20 shows how the topic tree can be divided further, to separate different types of information about each sport:

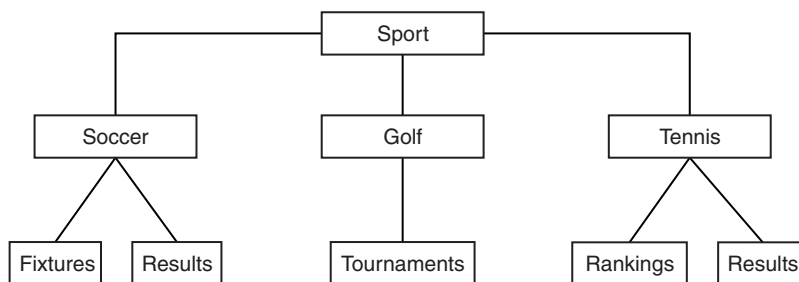


Figure 20. Extended topic tree

To create the topic tree illustrated, no administrative topic objects need be defined. Each of the nodes in this tree are defined by a topic string created in a publish or subscribe operation. Each topic in the tree inherits its attributes from its parent. Attributes are inherited from the parent topic object, because by

default all attributes are set to ASPARENT. In this example, every topic has the same attributes as the Sport topic. The Sport topic has no administrative topic object, and inherits its attributes from SYSTEM.BASE.TOPIC.

Note, that it is not good practice to give authorities for non-mqm users at the root node of the topic tree, which is SYSTEM.BASE.TOPIC, because the authorities are inherited but cannot be restricted. Therefore, by giving authorities at this level, you are giving authorities to the whole tree. You should give the authority at a lower topic level in the hierarchy.

Administrative topic objects can be used to define specific attributes for particular nodes in the topic tree. In the following example, the administrative topic object is defined to set the durable subscriptions property DURSUB of the soccer topic to the value NO:

```
DEFINE TOPIC(FOOTBALL.EUROPEAN)
TOPICSTR('Sport/Soccer')
DURSUB(NO)
DESCR('Administrative topic object to disallow durable subscriptions')
```

The topic tree can now be visualized as:

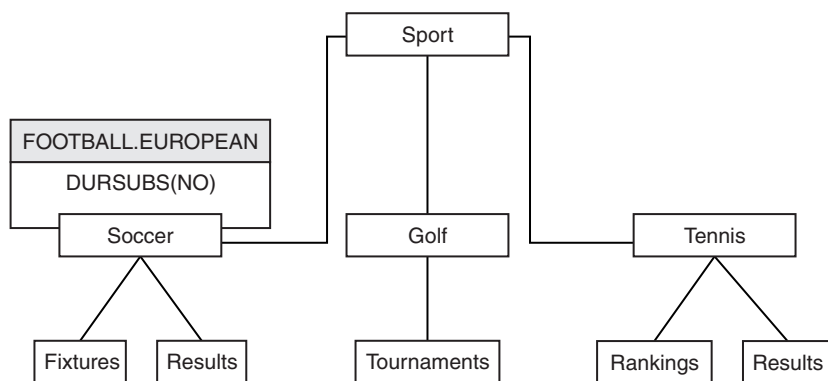


Figure 21. Visualization of an administrative topic object associated with the Sport/Soccer topic

Any applications subscribing to topics beneath Soccer in the tree can still use the topic strings they used before the administrative topic object was added. However, an application can now be written to subscribe using the object name FOOTBALL.EUROPEAN, instead of the string /Sport/Soccer. For example, to subscribe to /Sport/Soccer/Results, an application can specify MQSD.ObjectName as FOOTBALL.EUROPEAN and MQSD.ObjectString as Results.

With this feature, you can hide part of the topic tree from application developers. Define an administrative topic object at a particular node in the topic tree, then application developers can define their own topics as children of the node. Developers must know about the parent topic, but not about any other nodes in the parent tree.

Inheriting attributes

If a topic tree has many administrative topic objects, each administrative topic object, by default, inherits its attributes from its closest parent administrative topic. The previous example has been extended in Figure 22 on page 221:

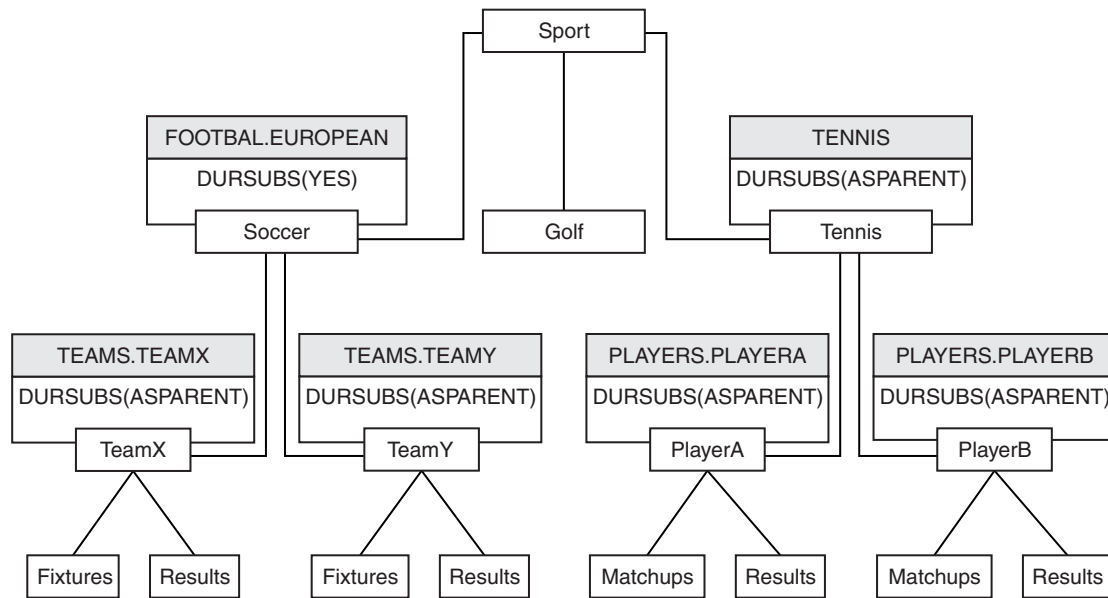


Figure 22. Topic tree with several administrative topic objects

For example use inheritance to give all the child topics of /Sport/Soccer the property that subscriptions are non-durable. Change the DURSUB attribute of FOOTBALL.EUROPEAN to NO.

This attribute can be set using the following command:

```
ALTER TOPIC(FOOTBALL.EUROPEAN) DURSUB(NO)
```

All the administrative topic objects of child topics of Sport/Soccer have the property DURSUB set to the default value ASPARENT. After changing the DURSUB property value of FOOTBALL.EUROPEAN to NO, the child topics of Sport/Soccer inherit the DURSUB property value NO. All child topics of Sport/Tennis inherit the value of DURSUB from SYSTEM.BASE.TOPIC object. SYSTEM.BASE.TOPIC has the value of YES.

Trying to make a durable subscription to the topic Sport/Soccer/TeamX/Results would now fail; however, trying to make a durable subscription to Sport/Tennis/PlayerB/Results would succeed.

Controlling wildcard usage with the WILDCARD property

Use the MQSC **Topic** WILDCARD property or the equivalent PCF Topic WildcardOperation property to control the delivery of publications to subscriber applications that use wildcard topic string names. The WILDCARD property can have one of two possible values:

WILDCARD

The behavior of wildcard subscriptions with respect to this topic.

PASSTHRU

Subscriptions made to a wildcarded topic less specific than the topic string at this topic object receive publications made to this topic and to topic strings more specific than this topic.

BLOCK

Subscriptions made to a wildcarded topic less specific than the topic string at this topic object do not receive publications made to this topic or to topic strings more specific than this topic.

The value of this attribute is used when subscriptions are defined. If you alter this attribute, the set of topics covered by existing subscriptions is not affected by the modification. This scenario

applies also if the topology is changed when topic objects are created or deleted; the set of topics matching subscriptions created following the modification of the WILDCARD attribute is created using the modified topology. If you want to force the matching set of topics to be re-evaluated for existing subscriptions, you must restart the queue manager.

In the example, “Example: Create the Sport publish/subscribe cluster” on page 225, you can follow the steps to create the topic tree structure shown in Figure 23.

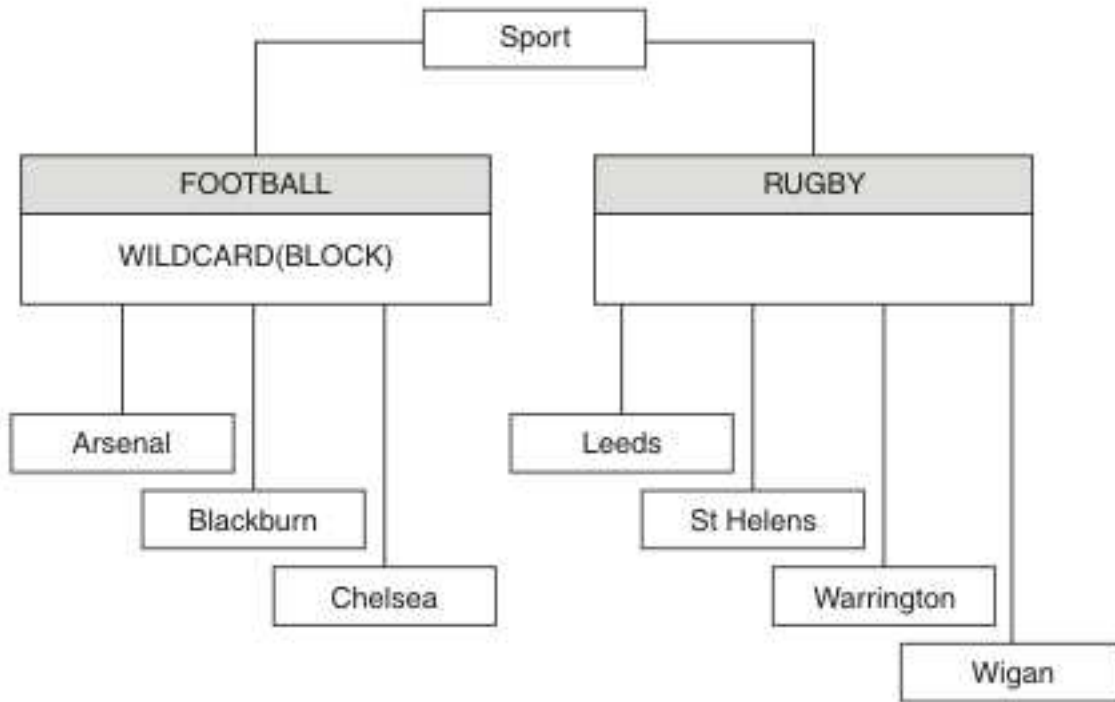


Figure 23. A topic tree that uses the WILDCARD property, BLOCK

A subscriber using the wildcard topic string # receives all publications to the Sport topic and the Sport/Rugby subtree. The subscriber receives no publications to the Sport/Football subtree, because the WILDCARD property value of the Sport/Football topic is BLOCK.

PASSTHRU is the default setting. You can set the WILDCARD property value PASSTHRU to nodes in the Sport tree. If the nodes do not have the WILDCARD property value BLOCK, setting PASSTHRU does not alter the behavior observed by subscribers to nodes in the Sports tree.

In the example, create subscriptions to see how the wildcard setting affects the publications that are delivered; see Figure 27 on page 227. Run the publish command in Figure 30 on page 228 to create some publications.

pub QMA

Figure 24. Publish to QMA

The results are shown in Table 8. Notice how setting the WILDCARD property value BLOCK, prevents subscriptions with wildcards from receiving publications to topics within the scope of the wildcard.

Table 8. Publications received on QMA

Subscription	Topic string	Publications received	Notes
SPORTS	Sports/#	Sports Sports/Rugby Sports/Rugby/Leeds	All publications to Football subtree blocked by WILDCARD(BLOCK) on Sports/Football
SARSENAL	Sports/#/Arsenal	-	WILDCARD(BLOCK) on Sports/Football prevents wildcard subscription on Arsenal
SLEEDS	Sports/#/Leeds	Sports/Rugby/Leeds	Default WILDCARD on Sports/Rugby does not prevent wildcard subscription on Leeds.

Note:

Suppose a subscription has a wildcard that matches a topic object with the WILDCARD property value BLOCK. If the subscription also has a topic string to the right of the matching wildcard, the subscription never receives a publication. The set of publications that are not blocked are publications to topics that are parents of the blocked wildcard. Publications to topics that are children of the topic with the BLOCK property value are blocked by the wildcard. Therefore subscription topic strings that include a topic to the right of the wildcard never receive any publications to match.

Setting the WILDCARD property value to BLOCK does not mean you cannot subscribe using a topic string that includes wildcards. Such a subscription is normal. The subscription has an explicit topic that matches the topic with a topic object having a WILDCARD property value BLOCK. It uses wildcards for topics that are parents or children of the topic with the WILDCARD property value BLOCK. In the example in Figure 23 on page 222, a subscription such as Sports/Football/# can receive publications.

Wildcards and cluster topics

Cluster topic definitions are propagated to every queue manager in a cluster. A subscription to a cluster topic at one queue manager in a cluster results in the queue manager creating proxy subscriptions. A proxy subscription is created at every other queue manager in the cluster. Subscriptions using topic strings containing wildcards, combined with cluster topics, can give hard to predict behavior. The behavior is explained in the following example.

In the cluster set up for the example, “Example: Create the Sport publish/subscribe cluster” on page 225, QMB has the same set of subscriptions as QMA, yet QMB received no publications after the publisher published to QMA, see Figure 24. Although the Sports/Football and Sports/Rugby topics are cluster topics, the subscriptions defined in fullsubs.tst do not reference a cluster topic. No proxy subscriptions are propagated from QMB to QMA. Without proxy subscriptions, no publications to QMA are forwarded to QMB.

Some of the subscriptions, such as Sports/#/Leeds, might seem to reference a cluster topic, in this case Sports/Rugby. The Sports/#/Leeds subscription actually resolves to the topic object SYSTEM.BASE.TOPIC.

The rule for resolving the topic object referenced by a subscription such as, Sports/#/Leeds is as follows. Truncate the topic string to the first wildcard. Scan left through the topic string looking for the first topic that has an associated administrative topic object. The topic object might specify a cluster name, or define

a local topic object. In the example, Sports/#/Leeds, the topic string after truncation is Sports, which has no topic object, and so Sports/#/Leeds inherits from SYSTEM.BASE.TOPIC, which is a local topic object.

To see how subscribing to clustered topics can change the way wildcard propagation works, run the batch script, upsubs.bat. The script clears the subscription queues, and adds the cluster topic subscriptions in fullsubs.tst. Run puba.bat again to create a batch of publications; see Figure 24 on page 223.

Table 9 shows the result of adding two new subscriptions to the same queue manager that the publications were published on. The result is as expected, the new subscriptions receive one publication each, and the numbers of publications received by the other subscriptions are unchanged. The unexpected results occur on the other cluster queue manager; see Table 10.

Table 9. Publications received on QMA

Subscription	Topic string	Publications received	Notes
SPORTS	Sports/#	Sports Sports/Rugby Sports/Rugby/Leeds	All publications to Football subtree blocked by WILDCARD(BLOCK) on Sports/Football
SARSENAL	Sports/#/Arsenal	-	WILDCARD(BLOCK) on Sports/Football prevents wildcard subscription on Arsenal
SLEEDS	Sports/#/Leeds	Sports/Rugby/Leeds	Default WILDCARD on Sports/Rugby does not prevent wildcard subscription on Leeds.
FARSENAL	Sports/Football/Arsenal	Sports/Football/Arsenal	Arsenal receives a publication because the subscription does not have a wildcard.
FLEEDS	Sports/Rugby/Leeds	Sports/Rugby/Leeds	Leeds would receive a publication in any event.

Table 10 shows the results of adding the two new subscriptions on QMB and publishing on QMA. Recall that QMB received no publications without these two new subscriptions. As expected, the two new subscriptions receive publications, because Sports/FootBall and Sports/Rugby are both cluster topics. QMB forwarded proxy subscriptions for Sports/Football/Arsenal and Sports/Rugby/Leeds to QMA, which then sent the publications to QMB.

The unexpected result is that the two subscriptions Sports/# and Sports/#/Leeds that previously received no publications, now receive publications. The reason is that the Sports/Football/Arsenal and Sports/Rugby/Leeds publications forwarded to QMB for the other subscriptions are now available for any subscriber attached to QMB. Consequently the subscriptions to the local topics Sports/# and Sports/#/Leeds receive the Sports/Rugby/Leeds publication. Sports/#/Arsenal continues not to receive a publication, because Sports/Football has its WILDCARD property value set to BLOCK.

Table 10. Publications received on QMB

Subscription	Topic string	Publications received	Notes
SPORTS	Sports/#	Sports/Rugby/Leeds	All publications to Football subtree blocked by WILDCARD(BLOCK) on Sports/Football
SARSENAL	Sports/#/Arsenal	-	WILDCARD(BLOCK) on Sports/Football prevents wildcard subscription on Arsenal
SLEEDS	Sports/#/Leeds	Sports/Rugby/Leeds	Default WILDCARD on Sports/Rugby does not prevent wildcard subscription on Leeds.

Table 10. Publications received on QMB (continued)

Subscription	Topic string	Publications received	Notes
FARSENAL	Sports/Football/Arsenal	Sports/Football/Arsenal	Arsenal receives a publication because the subscription does not have a wildcard.
FLEEDS	Sports/Rugby/Leeds	Sports/Rugby/Leeds	Leeds would receive a publication in any event.

In most applications, it is undesirable for one subscription to influence the behavior of another subscription. One important use of the WILDCARD property with the value BLOCK is to make the subscriptions to the same topic string containing wildcards behave uniformly. Whether the subscription is on the same queue manager as the publisher, or a different one, the results of the subscription are the same.

Wildcards and streams

For a new application written to the publish/subscribe API, the effect is that a subscription to * receives no publications. To receive all the Sports publications you must subscribe to Sports/*, or Sports/#, and similarly for Business publications.

The behavior of an existing queued publish/subscribe application does not change when the publish/subscribe broker is migrated to IBM MQ Version 7 and later versions. The **StreamName** property in the **Publish**, **Register Publisher**, or **Subscriber** commands is mapped to the name of the topic the stream has been migrated to.

Wildcards and subscription points

For a new application written to the publish/subscribe API, the effect of the migration is that a subscription to * receives no publications. To receive all the Sports publications you must subscribe to Sports/*, or Sports/#, and similarly for Business publications.

The behavior of an existing queued publish/subscribe application does not change when the publish/subscribe broker is migrated to IBM MQ Version 7 and later versions. The **SubPoint** property in the **Publish**, **Register Publisher**, or **Subscriber** commands is mapped to the name of the topic the subscription has been migrated to.

Example: Create the Sport publish/subscribe cluster

The steps that follow create a cluster, CL1, with four queue managers: two full repositories, CL1A and CL1B, and two partial repositories, QMA and QMB. The full repositories are used to hold only cluster definitions. QMA is designated the cluster topic host. Durable subscriptions are defined on both QMA and QMB.

Note: The example is coded for Windows. You must recode Create qmgrs.bat and create pub.bat to configure and test the example on other platforms.

1. Create the script files.
 - a. Create topics.tst
 - b. Create wildsubs.tst
 - c. Create fullsubs.tst
 - d. Create qmgrs.bat
 - e. create pub.bat
2. Run Create qmgrs.bat to create the configuration.

qmgrs

Create the topics in Figure 23 on page 222. The script in figure 5 creates the cluster topics Sports/Football and Sports/Rugby.

Note: The REPLACE option does not replace the TOPICSTR properties of a topic. TOPICSTR is a property that is usefully varied in the example to test different topic trees. To change topics, delete the topic first.

```
DELETE TOPIC ('Sports')
DELETE TOPIC ('Football')
DELETE TOPIC ('Arsenal')
DELETE TOPIC ('Blackburn')
DELETE TOPIC ('Chelsea')
DELETE TOPIC ('Rugby')
DELETE TOPIC ('Leeds')
DELETE TOPIC ('Wigan')
DELETE TOPIC ('Warrington')
DELETE TOPIC ('St. Helens')

DEFINE TOPIC ('Sports') TOPICSTR('Sports')
DEFINE TOPIC ('Football') TOPICSTR('Sports/Football') CLUSTER(CL1) WILDCARD(BLOCK)
DEFINE TOPIC ('Arsenal') TOPICSTR('Sports/Football/Arsenal')
DEFINE TOPIC ('Blackburn') TOPICSTR('Sports/Football/Blackburn')
DEFINE TOPIC ('Chelsea') TOPICSTR('Sports/Football/Chelsea')
DEFINE TOPIC ('Rugby') TOPICSTR('Sports/Rugby') CLUSTER(CL1)
DEFINE TOPIC ('Leeds') TOPICSTR('Sports/Rugby/Leeds')
DEFINE TOPIC ('Wigan') TOPICSTR('Sports/Rugby/Wigan')
DEFINE TOPIC ('Warrington') TOPICSTR('Sports/Rugby/Warrington')
DEFINE TOPIC ('St. Helens') TOPICSTR('Sports/Rugby/St. Helens')
```

Figure 25. Delete and create topics: topics.tst

Note: Delete the topics, as REPLACE does not replace topic strings.

Create subscriptions with wildcards. The wildcards corresponding the topics with topic objects in Figure 23 on page 222. Create a queue for each subscription. The queues are cleared and the subscriptions deleted when the script is run or rerun.

Note: The REPLACE option does not replace TOPICOBJ or TOPICSTR properties of a subscription. TOPICOBJ or TOPICSTR are the properties that are usefully varied in the example to test different subscriptions. To change them, delete the subscription first.

```
DEFINE QLOCAL(QSPORTS) REPLACE
DEFINE QLOCAL(QARSENAL) REPLACE
DEFINE QLOCAL(QSLEEDS) REPLACE
CLEAR QLOCAL(QSPORTS)
CLEAR QLOCAL(QARSENAL)
CLEAR QLOCAL(QSLEEDS)

DELETE SUB (SPORTS)
DELETE SUB (SARSENAL)
DELETE SUB (SLEEDS)
DEFINE SUB (SPORTS) TOPICSTR('Sports/#') DEST(QSPORTS)
DEFINE SUB (SARSENAL) TOPICSTR('Sports+/Arsenal') DEST(QARSENAL)
DEFINE SUB (SLEEDS) TOPICSTR('Sports+/Leeds') DEST(QSLEEDS)
```

Figure 26. Create wildcard subscriptions: wildsubs.tst

Create subscriptions that reference the cluster topic objects.

Note:

The delimiter, /, is automatically inserted between the topic string referenced by TOPICOBJ, and the topic string defined by TOPICSTR.

The definition, `DEFINE SUB(FARSENAL) TOPICSTR('Sports/Football/Arsenal') DEST(QFARSENAL)` creates the same subscription. `TOPICOBJ` is used as a quick way to reference topic string you have already defined. The subscription, when created, no longer refers to the topic object.

```
DEFINE QLOCAL(QFARSENAL) REPLACE
DEFINE QLOCAL(QRLEEDS) REPLACE
CLEAR QLOCAL(QFARSENAL)
CLEAR QLOCAL(QRLEEDS)

DELETE SUB (FARSENAL)
DELETE SUB (RLEEDS)
DEFINE SUB (FARSENAL) TOPICOBJ('Football') TOPICSTR('Arsenal') DEST(QFARSENAL)
DEFINE SUB (RLEEDS) TOPICOBJ('Rugby') TOPICSTR('Leeds') DEST(QRLEEDS)
```

Figure 27. Delete and create subscriptions: fullsubs.tst

Create a cluster with two repositories. Create two partial repositories for publishing and subscribing. Rerun the script to delete everything and start again. The script also creates the topic hierarchy, and the initial wildcard subscriptions.

Note:

On other platforms, write a similar script, or type all the commands. Using a script makes it quick to delete everything and start again with an identical configuration.

```
@echo off
set port.CL1B=1421
set port.CL1A=1420
for %%A in (CL1A CL1B QMA QMB) do call :createQM %%A
call :configureQM CL1A CL1B %port.CL1B% full
call :configureQM CL1B CL1A %port.CL1A% full
for %%A in (QMA QMB) do call :configureQM %%A CL1A %port.CL1A% partial
for %%A in (topics.tst wilddsubs.tst) do runmqsc QMA < %%A
for %%A in (wilddsubs.tst) do runmqsc QMB < %%A
goto:eof

:createQM
echo Configure Queue manager %1
endmqm -p %1
for %%B in (dlt crt str) do %%Bmqm %1
goto:eof

:configureQM
if %1==CL1A set p=1420
if %1==CL1B set p=1421
if %1==QMA set p=1422
if %1==QMB set p=1423
echo configure %1 on port %p% connected to repository %2 on port %3 as %4 repository
echo DEFINE LISTENER(LST%1) TRPTYPE(TCP) PORT(%p%) CONTROL(QMGR) REPLACE | runmqsc %1
echo START LISTENER(LST%1) | runmqsc %1
if full==%4 echo ALTER QMGR REPOS(CL1) DEADQ(SYSTEM.DEAD.LETTER.QUEUE) | runmqsc %1
echo DEFINE CHANNEL(TO.%2) CHLTYPE(CLUSSDR) TRPTYPE(TCP) CONNAME('LOCALHOST(%3)') CLUSTER(CL1) REPLACE | runmqsc %1
echo DEFINE CHANNEL(TO.%1) CHLTYPE(CLUSRCVR) TRPTYPE(TCP) CONNAME('LOCALHOST(%p)') CLUSTER(CL1) REPLACE | runmqsc %1
goto:eof
```

Figure 28. Create queue managers: qmgrs.bat

Update the configuration by adding the subscriptions to the cluster topics.

```
@echo off
for %%A in (QMA QMB) do runmqsc %%A < wildsubs.tst
for %%A in (QMA QMB) do runmqsc %%A < upsubs.tst
```

Figure 29. Update subscriptions: *upsubs.bat*

Run *pub.bat*, with a queue manager as a parameter, to publish messages containing the publication topic string. *Pub.bat* uses the sample program **amqspub**.

```
@echo off
@rem Provide queue manager name as a parameter
set S=Sports
set S=6 Sports/Football Sports/Football/Arsenal
set S=6 Sports/Rugby Sports/Rugby/Leeds
for %%B in (6) do echo %%B | amqspub %%B %1
```

Figure 30. Publish: *pub.bat*

Streams and topics:

Queued publish/subscribe has the concept of a publication stream that does not exist in the integrated publish/subscribe model. In queued publish/subscribe, streams provide a way of separating the flow of information for different topics. From IBM WebSphere MQ Version 7.0 onwards, a stream is implemented as a top-level topic that can be mapped to a different topic identifier administratively.

The default stream `SYSTEM.BROKER.DEFAULT.STREAM` is set up automatically for all brokers and queue managers on a network, and no additional configuration is required to use the default stream. Think of the default stream as an unnamed default topic space. Topics published to the default stream are immediately available to all connected queue managers from Version 7.0 onwards, with queued publish/subscribe enabled. Named streams are like separate, named, topic spaces. The named stream must be defined on each broker where it is used.

If the publishers and subscribers are on different queue managers, then after the brokers are connected in the same broker hierarchy, no further configuration is required for the publications, and subscriptions to flow between them. The same interoperability works in reverse, too.

Named streams

A solution designer, working with the queued publish/subscribe programming model, might decide to place all sports publications into a named stream called `Sport`. In Version 6.0 a stream is often replicated automatically onto other brokers that use the model queue, `SYSTEM.BROKER.MODEL.STREAM`. However, for the stream to be available to a queue manager that runs on Version 7.0 onwards with queued publish/subscribe enabled, the stream must be added manually.

Queued publish/subscribe applications that subscribe to `Soccer/Results` on stream `Sport` work without change. Integrated publish/subscribe applications that subscribe to the topic `Sport` using `MQSUB`, and supplying the topic string `Soccer/Results` receive the same publications too.

The task of adding a stream is described in the topic `Adding a stream`. You might need to add streams manually for two reasons.

1. You continue to develop your queued publish/subscribe applications that are running on later version queue managers, rather than migrate the applications to the integrated publish/subscribe MQI interface.
2. The default mapping of streams to topics leads to a "collision" in topic space, and publications on a stream have the same topic string as publications from elsewhere.

Authorities

By default, at the root of the topic tree there are multiple topic objects: `SYSTEM.BASE.TOPIC`, `SYSTEM.BROKER.DEFAULT.STREAM`, and `SYSTEM.BROKER.DEFAULT.SUBPOINT`. Authorities (for example, for publishing or subscribing) are determined by the authorities on the `SYSTEM.BASE.TOPIC`; any authorities on `SYSTEM.BROKER.DEFAULT.STREAM` or `SYSTEM.BROKER.DEFAULT.SUBPOINT` are ignored. If either of `SYSTEM.BROKER.DEFAULT.STREAM` or `SYSTEM.BROKER.DEFAULT.SUBPOINT` are deleted and re-created with a non-empty topic string, authorities defined on those objects are used in the same way as a normal topic object.

Mapping between streams and topics

A queued publish/subscribe stream is mimicked in Version 7.0 onwards by creating a queue, and giving it the same name as the stream. Sometimes the queue is called the stream queue, because that is how it appears to queued publish/subscribe applications. The queue is identified to the publish/subscribe engine by adding it to the special namelist called `SYSTEM.QPUBSUB.QUEUE.NAMELIST`. You can add as many streams as you need, by adding additional special queues to the namelist. Finally you need to add topics, with the same names as the streams, and the same topic strings as the stream name, so you can publish and subscribe to the topics.

However, in exceptional circumstances, you can give the topics corresponding to the streams any topic strings you choose when you define the topics. The purpose of the topic string is to give the topic a unique name in the topic space. Typically the stream name serves that purpose perfectly. Sometimes, a stream name and an existing topic name collide. To resolve the problem, you can choose another topic string for the topic associated with the stream. Choose any topic string, ensuring it is unique.

The topic string defined in the topic definition is prefixed in the normal way to the topic string provided by publishers and subscribers using the `MQOPEN` or `MQSUB` MQI calls. Applications referring to topics using topic objects are not affected by the choice of prefix topic string - which is why you can choose any topic string that keeps the publications unique in the topic space.

The remapping of different streams onto different topics relies on the prefixes used for the topic strings being unique, to separate one set of topics completely from another. You must define a universal topic naming convention that is rigidly adhered to for the mapping to work.

In Version 7.0, if topic strings collided you might use streams to separate the topic spaces.

From Version 7.0 onwards, you use the prefixing mechanism to remap a topic string to another place in topic space.

Note: When you delete a stream, delete all the subscriptions on the stream first. This action is most important if any of the subscriptions originate from other brokers in the broker hierarchy.

Example

In Figure 31 on page 230, topic 'Sport' has the topic string 'xyz' resulting in publications that originate from stream 'Sport' being prefixed with the string 'xyz' in the version 7 queue manager topic space. Publishing or subscribing in version 7 to the topic 'Sport' prefixes 'xyz' to the topic string. If the publication flows to a Version 6 subscriber, the prefix 'xyz' is removed from the publication and it is placed in the 'Sport' stream. Conversely, when a publication flows from Version 6 to version 7, from the 'Sport' stream to the 'Sport' topic, the prefix 'xyz' is added to the topic string.

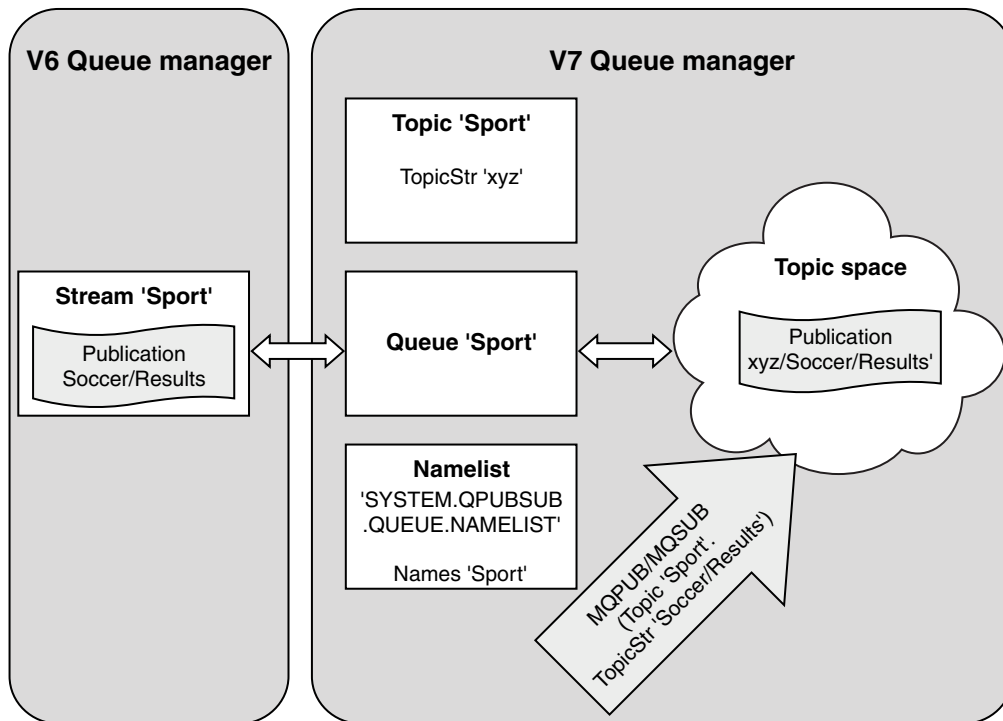


Figure 31. Version 6 streams that coexist with version 7 topics

Subscription points and topics:

Named subscription points are emulated by topics and topic objects.

To add subscription points manually, see Adding a subscription point.

Subscription points in IBM MQ

IBM MQ maps subscription points to different topic spaces within the IBM MQ topic tree. Topics in command messages without a subscription point are mapped unchanged to the root of the IBM MQ topic tree and inherit properties from SYSTEM.BASE.TOPIC.

Command messages with a subscription point are processing using the list of topic objects in SYSTEM.QPUBSUB.SUBPOINT.NAMELIST. The subscription point name in the command message is matched against the topic string for each of the topic objects in the list. If a match is found, then the subscription point name is prepended, as a topic node, to the topic string. The topic inherits its properties from the associated topic object found in SYSTEM.QPUBSUB.SUBPOINT.NAMELIST.

The effect of using subscription points is to create a separate topic space for each subscription point. The topic space is rooted in a topic that has the same name as the subscription point. Topics in each topic space inherit their properties from the topic object with the same name as the subscription point.

Any properties not set in the matching topic object are inherited, in the normal fashion, from SYSTEM.BASE.TOPIC.

Existing queued publish/subscribe applications, using MQRFH2 message headers, continue to work by setting the **SubPoint** property in the Publish or Register subscriber command messages. The subscription point is combined with the topic string in the command message and the resulting topic processed like any other.

IBM WebSphere MQ Version 7.0, or later, applications are unaffected by subscription points. If an application uses a topic that inherits information from one of the matching topic objects, that application interoperates with a queued application using the matching subscription point.

Example

An existing WebSphere Message Broker (now known as IBM Integration Bus) publish/subscribe application that was migrated to IBM MQ created two topic objects, GBP and USD, with the corresponding topic strings 'GBP' and 'USD'.

Existing publishers to the topic NYSE/IBM/SPOT, migrated to run on IBM MQ, that use the subscription point USD create publications on the topic USD/NYSE/IBM/SPOT. Similarly existing subscribers to NYSE/IBM/SPOT, using the subscription point USD create subscriptions to USD/NYSE/IBM/SPOT.

Subscribe to the dollar spot price in a IBM WebSphere MQ Version 7.0, or later, publish/subscribe program by calling MQSUB. Create a subscription using the USD topic object and the topic string 'NYSE/IBM/SPOT', as illustrated in the 'C' code fragment.

```
strncpy(sd.ObjectName, "USD", MQ_TOPIC_NAME_LENGTH);
sd.ObjectString.VSPtr = "NYSE/IBM/SPOT";
sd.ObjectString.VSLength = MQVS_NULL_TERMINATED;
MQSUB(Hconn, &sd, &Hobj, &Hsub, &CompCode, &Reason);
```

1. Set the CLUSTER attribute of the USD and GBP topic objects on the cluster topic host.
2. Delete all the copies of the USD and GBP topic objects on other queue managers in the cluster.
3. Make sure that USD and GBP are defined in SYSTEM.QPUBSUB.SUBPOINT.NAMELIST on every queue manager in the cluster.

Example of a single queue manager publish/subscribe configuration

Figure 32 on page 232 illustrates a basic single queue manager publish/subscribe configuration. The example shows the configuration for a news service, where information is available from publishers about several topics:

- Publisher 1 is publishing information about sports results using a topic of Sport
- Publisher 2 is publishing information about stock prices using a topic of Stock
- Publisher 3 is publishing information about film reviews using a topic of Films, and about television listings using a topic of TV

Three subscribers have registered an interest in different topics, so the queue manager sends them the information that they are interested in:

- Subscriber 1 receives the sports results and stock prices
- Subscriber 2 receives the film reviews
- Subscriber 3 receives the sports results

None of the subscribers have registered an interest in the television listings, so these are not distributed.

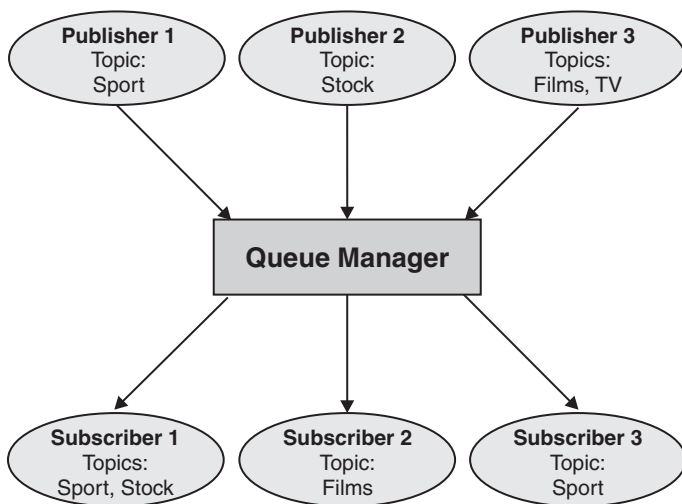


Figure 32. Single queue manager publish/subscribe example. This shows the relationship between publishers, subscribers, and queue managers.

Distributed publish/subscribe networks

Each queue manager matches messages published to a topic with the locally created subscriptions that have subscribed to that topic. You can configure a network of queue managers so that messages published by an application connected to one queue manager are delivered to matching subscriptions created on other queue managers in the network. This requires additional configuration over simple channels between queue managers.

A distributed publish/subscribe configuration is a set of queue managers connected together. The queue managers can all be on the same physical system, or they can be distributed over several physical systems. When you connect queue managers together, subscribers can subscribe to one queue manager and receive messages that were initially published to another queue manager. To illustrate this, the following figure adds a second queue manager to the configuration described in “Example of a single queue manager publish/subscribe configuration” on page 231.

- Queue manager 2 is used by Publisher 4 to publish weather forecast information, using a topic of Weather, and information about traffic conditions on major roads, using a topic of Traffic.
- Subscriber 4 also uses this queue manager, and subscribes to information about traffic conditions using topic Traffic.
- Subscriber 3 also subscribes to information about weather conditions, even though it uses a different queue manager from the publisher. This is possible because the queue managers are linked to each other.

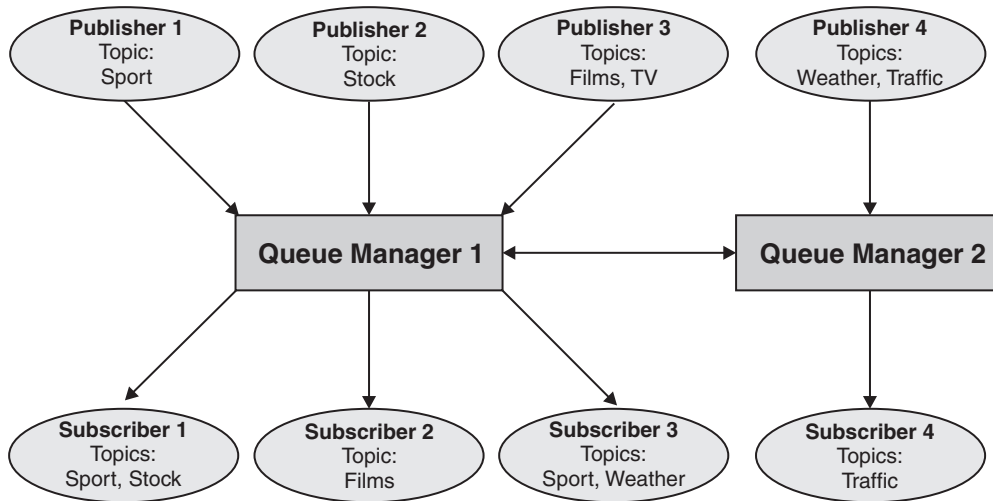


Figure 33. Publish/subscribe example with two queue managers

You can manually connect queue managers in a parent and child hierarchy, or you can create a publish/subscribe cluster and let IBM MQ define much of the connection detail for you. You can also use both topologies in combination, for example by joining several clusters together in a hierarchy.

Overview of publish/subscribe clusters

A publish/subscribe cluster is a standard cluster with one or more topic objects added to the cluster. When you define an administrative topic object on any queue manager in a cluster, and make that topic object clustered by specifying a cluster name, then publishers and subscribers to the topic can connect to any of the queue managers in the cluster, and messages published are routed to the subscribers over cluster channels between queue managers.

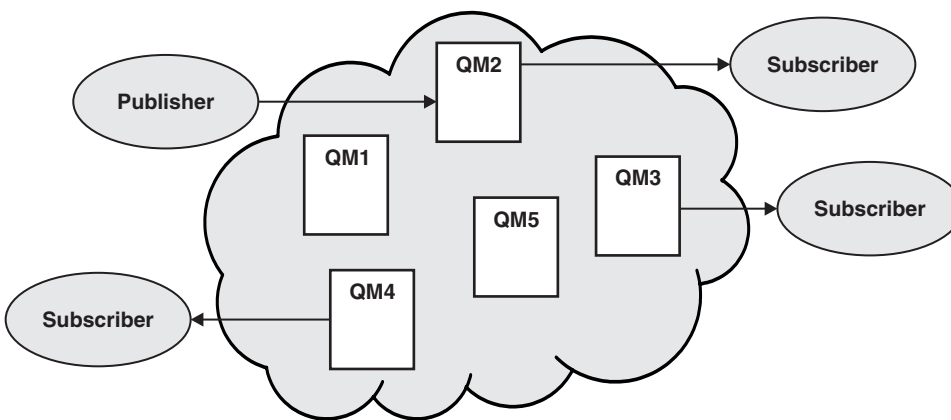


Figure 34. Publish/subscribe cluster

There are two ways to configure how publish/subscribe messages are routed in a cluster:

- direct routing
- topic host routing

When you configure a direct routed clustered topic, messages published on one queue manager are sent directly from that queue manager to every subscription on any other queue manager in the cluster. This

can provide the most direct path for publications but does result in all queue managers in a cluster becoming aware of all other queue managers, each potentially having cluster channels established between them.

When you use topic host routing, messages published on one queue manager are sent from there to a queue manager that hosts a definition of the administered topic object. That *topic host queue manager* routes the message on to every subscription on any other queue manager in the cluster. If the publishers or subscribers are not located on the topic host queue managers, this results in a longer route for publications. However, the benefit is that only the topic host queue managers become aware of all other queue managers in the cluster, and potentially have cluster channels established with them.

For more information, see “Publish/subscribe clusters” on page 235.

Overview of publish/subscribe hierarchies

A publish/subscribe hierarchy is a set of queue managers connected by channels into a hierarchical structure. Each queue manager identifies its *parent* queue manager, as described in Connecting a queue manager to a publish/subscribe hierarchy.

Publishers and subscribers to a topic can connect to any queue manager in the hierarchy, and messages flow between them using the hierarchical queue manager connectivity.

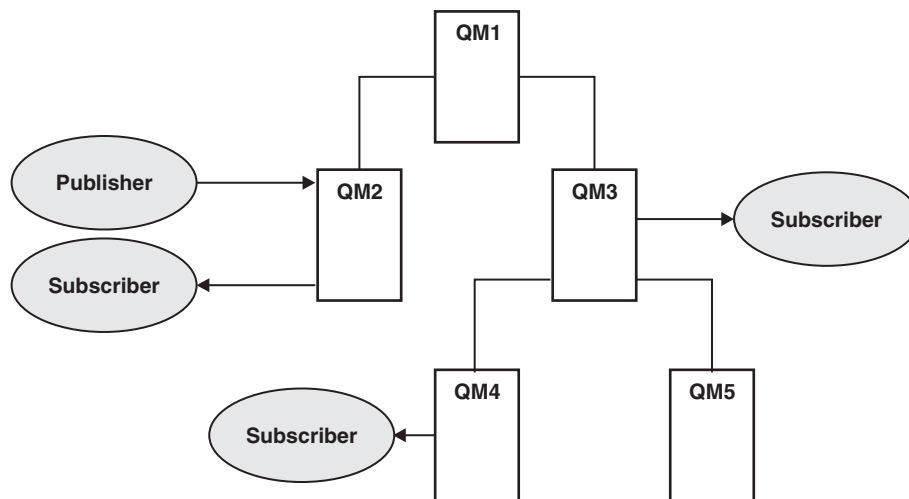


Figure 35. Publish/subscribe hierarchy

In the previous figure, publications delivered to the subscribers on QM3 and QM4 have been routed from QM2 to QM1 and then onto QM3 and finally QM4.

Hierarchies give you direct control over the relationships between every queue manager in the hierarchy. This allows fine-grained control over the routing of messages from publishers to subscribers, and is especially useful when routing between queue manager networks with restricted connectivity. You should give careful consideration to the availability and capability of every queue manager through which a message is routed on its way from publisher to subscribers.

For more information, see “Publish/subscribe hierarchies” on page 238.

Publication distribution between queue managers

In addition to the routing choices, there are two approaches to distributing publications across a network of queue managers:

- Only send publications from one queue manager to the queue managers that currently host a subscription for that publication.
- Send each publication to all queue managers, and let them match it against their subscriptions.

The former results in publication messages only being sent where necessary, but does require a level of subscription knowledge to be shared between queue managers. The latter does not require subscription knowledge being shared, but can result in unnecessary publication messages being sent between queue managers.

By default, IBM MQ uses the former method, in which publications are only sent to queue managers that have subscriptions for them. The subscription knowledge is propagated between queue managers in the form of *proxy subscriptions*. It depends on the distribution and lifetime of subscriptions, and the frequency of publications, as to which is the most efficient to use in a distributed publish/subscribe topology. See Subscription performance in publish/subscribe networks.

Related concepts:

“Topic trees” on page 218

Each topic that you define is an element, or node, in the topic tree. The topic tree can either be empty to start with or contain topics that have been defined previously using MQSC or PCF commands. You can define a new topic either by using the create topic commands or by specifying the topic for the first time in a publication or subscription.

Related information:

Designing publish/subscribe clusters

Publish/subscribe hierarchy scenarios

Publish/subscribe clusters

A publish/subscribe cluster is a standard cluster of interconnected queue managers, on which publications are automatically moved from publishing applications to subscriptions that exist on any of the queue managers in the cluster. There are two options for routing publications across a publish/subscribe cluster: *direct routing* and *topic host routing*. The routing you choose depends upon the size and expected activity patterns for your cluster.

A cluster that is used for publish/subscribe messaging is no different from a standard IBM MQ cluster. As such, the queue managers within the publish/subscribe cluster can exist on physically separate computers and each pair of queue managers is automatically connected together by cluster channels when necessary. For more information, see Clusters.

To configure a standard cluster of queue managers for publish/subscribe messaging, you define one or more administered topic objects on a queue manager in the cluster. To make the topic a cluster topic, you configure the **CLUSTER** property with the name of the cluster. When you do this, any topic used by a publisher or subscriber at that point or below in the topic tree is shared across all queue managers in the cluster, and messages published to a clustered branch of the topic tree are automatically routed to subscriptions on other queue managers in the cluster.

Only one copy of each message is sent between the publisher queue manager and each of the other queue managers, irrespective of the number of subscribers for the message on the target queue manager. On arrival at a queue manager with one or more subscriptions, the message is duplicated across all subscriptions.

Any queue manager joining the cluster automatically becomes aware of the clustered topics, and publishers and subscribers on that queue manager automatically participate in the cluster.

Non-clustered publish/subscribe activity can also take place in a publish/subscribe cluster, by working with topic strings that do not fall under a clustered topic object.

There are two options for routing publications across a publish/subscribe cluster: *direct routing* and *topic host routing*. To choose the message routing to use within the cluster, you set the **CLROUTE** property on the administered topic object to one of the following values:

- **DIRECT**
- **TOPICHOST**

By default, topic routing is **DIRECT**. This was the only option prior to IBM MQ Version 8.0. When you configure a direct routed clustered topic on a queue manager, all queue managers in the cluster become aware of all other queue managers in the cluster. When performing publish and subscribe operations, each queue manager can connect direct to any other queue manager in the cluster.

From IBM MQ Version 8.0, you can instead configure topic routing as **TOPICHOST**. When you use topic host routing, all queue managers in the cluster become aware of the cluster queue managers that host the routed topic definition (that is, the queue managers on which you have defined the topic object). When performing publish and subscribe operations, queue managers in the cluster connect only to these topic host queue managers, and not directly to each other. The topic host queue managers are responsible for routing publications from queue managers on which publications are published to queue managers with matching subscriptions.

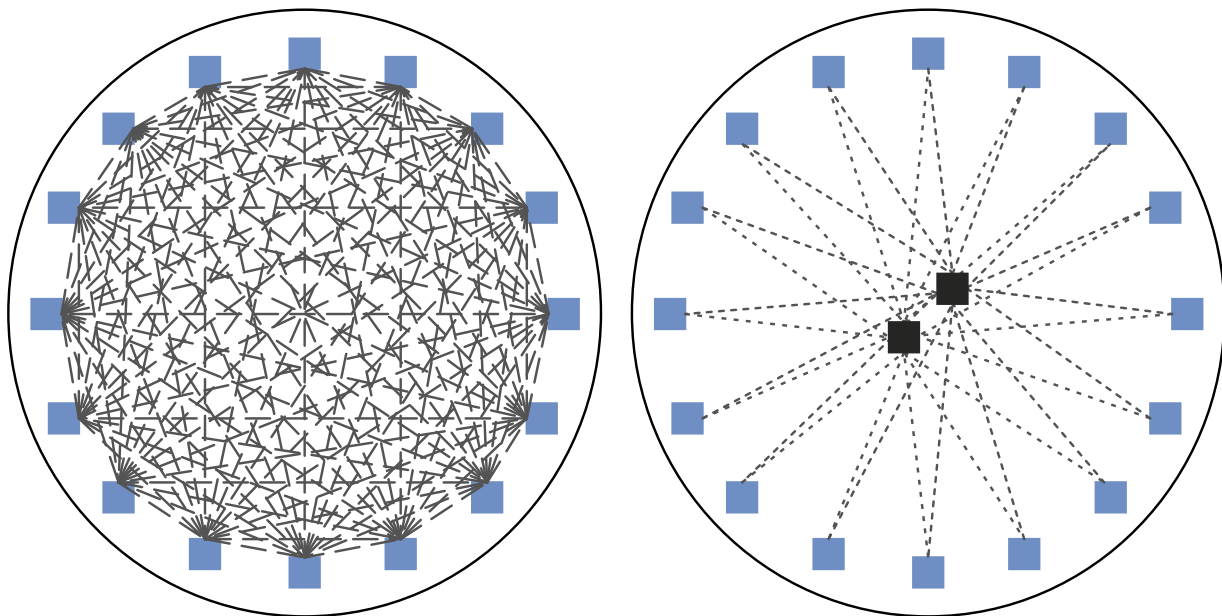


Figure 36. Direct routing and topic host routing

An overview of direct routing

When an administered topic object is configured for direct routing, the topic object only needs to be defined on one of the queue managers in the cluster for all queue managers to learn of it. The choice of queue manager on which the topic is defined does not affect the behavior of publish/subscribe messaging for the topic.

Each message flows directly from the publisher queue manager to each subscription on the other queue managers in the cluster, not passing through any intermediate queue managers.

By default, messages are sent only to other queue managers in the cluster that host one or more subscriptions.

- This relies on each queue manager directly informing all other queue managers in the cluster of all topics that currently have one or more subscriptions to it. This results in all queue managers in the cluster being aware of all topics being subscribed to, and any queue manager that hosts a subscription establishing a channel to every other queue manager. This is independent of whether each queue manager has a publisher.
- The knowledge of each individual subscribed topic on all queue managers can be removed by changing to a model of sending all publications to all queue managers in the cluster, irrespective of whether they have subscriptions. This reduces the subscription knowledge traffic, but is likely to increase the publication traffic and the number of channels each queue manager establishes. See Subscription performance in publish/subscribe networks.

Publish/subscribe message flows using direct routed clustered topics can span multiple publish/subscribe clusters by adding one queue manager from each cluster into a publish/subscribe hierarchy. See Combining the topic spaces of multiple clusters.

For a more detailed exploration of direct routing, see Direct routing in publish/subscribe clusters.

An overview of topic host routing

When an administered topic object is configured for topic host routing, publications from a queue manager in the cluster are routed through a queue manager where the topic object is configured (a "topic host"), and from there on to the queue managers where subscriptions exist.

- This relies on each queue manager informing all topic hosts of every topic that currently has one or more subscriptions to it. Any queue manager hosting a subscription establishes a channel to every topic host for the topic that the subscription relates to.
- Non-topic hosting queue managers are not made aware of other non-topic hosting queue managers in the cluster for the purposes of publish/subscribe, and channels are not established between them for that purpose.
- If the publishing application is connected to a queue manager hosting the topic, the published messages are routed directly to the queue managers where matching subscriptions have been created, without requiring an additional 'hop'. Similarly, if the matching subscriptions are created on the only queue manager hosting the topic, messages published to that topic are routed directly to that queue manager, without requiring an additional hop.
- Subscriptions on the same queue manager as the publisher are satisfied without first routing publications to the hosts of the topic object.

As for clustered queues, multiple queue managers can configure the same administrative topic object. This provides higher availability of message routing, and horizontal scaling through workload balancing. For topic host routed topic objects, when multiple queue managers configure the same named topic for the same branch of the topic tree, each topic host is made aware of the subscribed topics by every queue manager hosting a subscription.

- When a message is published, it is sent to one of the topic host queue managers to forward on to the subscription hosting queue managers. The choice of the topic host queue manager follows the same default workload balancing rules as for clustered point-to-point queues.
- If one or more topic host queue managers cannot be contacted by a publishing queue manager, messages are routed to the remaining available topic hosting queue managers.

Every publication to a topic in a routed branch of the topic tree is forwarded to one of the topic hosts, even if there are no subscriptions to that topic anywhere in the cluster. By default, messages are sent from here only to other queue managers in the cluster that host one or more subscriptions.

- This relies on each topic host queue manager being informed of all subscribed topic strings on each queue manager in the cluster.
- The knowledge of each individual subscribed topic can be removed by changing to a model of sending all publications routed to a topic host on to all queue managers in the cluster, irrespective of whether

they have subscriptions. This reduces the subscription knowledge traffic but is likely to increase the publication traffic and potentially the number of channels established with each topic hosting queue manager. See Subscription performance in publish/subscribe networks.

Publish/subscribe message flows using topic host routed clustered topics **cannot** span multiple publish/subscribe clusters through the use of a publish/subscribe hierarchy.

For a more detailed exploration of topic host routing, see Topic host routing in publish/subscribe clusters.

Publish/subscribe hierarchies

You build a publish/subscribe hierarchy by linking the queue managers together using channels, then defining a child-parent relationship between pairs of queue managers. A message flows from a publisher to the subscriptions through the direct relations in a hierarchy. Note that this might mean multiple “hops” to get there.

Only one copy of the message is sent between any one pair of queue managers, irrespective of the number of subscribers for the message on the target queue manager. On arrival at a queue manager with one or more subscriptions, the message is duplicated across all subscriptions.

By default, messages are only sent to other queue managers in the hierarchy that are on the route to a subscription on another queue manager:

- This relies on each queue manager informing each direct relation of all topics that currently have one or more subscriptions to it, either on this queue manager or on one of its other relations. This results in all queue managers in the hierarchy being aware of all topics being subscribed to.
- This behavior can be changed to always send publications to all queue managers in the hierarchy, irrespective of any subscriptions existing. This removes the need for propagating the subscription information across the hierarchy, but can increase the publication traffic.

When you create a cluster, you need to take care not to create a loop causing messages to cycle forever within the network. No such loops can be created in a hierarchy.

Every queue manager must have a unique queue manager name.

Publish/subscribe message flows can span multiple publish/subscribe clusters. To do this, add one queue manager from each cluster into a publish/subscribe hierarchy.

For a more detailed exploration, see Routing in publish/subscribe hierarchies.

Proxy subscriptions in a publish/subscribe network

A proxy subscription is a subscription made by one queue manager for topics published on another queue manager. A proxy subscription flows between queue managers for each individual topic string that is subscribed to by a subscription. You do not create proxy subscriptions explicitly, the queue manager does so on your behalf.

You can connect queue managers together into a publish/subscribe cluster, or into a publish/subscribe hierarchy. Proxy subscriptions flow between the connected queue managers. Proxy subscriptions cause publications to a topic created by a publisher connected to one queue manager to be received by subscribers to that topic connected to other queue managers. See “Distributed publish/subscribe networks” on page 232.

In publish/subscribe topologies with many thousands of subscriptions to individual topic strings, or where the existence of those subscriptions might be rapidly changing, the overhead of proxy subscription propagation must be considered. In addition to the automatic aggregation described in the rest of this topic, you can make manual configuration changes that further restrict the flow of proxy subscriptions

and publications between connected queue managers, and that reduce the latency of waiting for a proxy subscription to be propagated to all the connected queue managers. See Subscription performance in publish/subscribe networks.

Proxy subscriptions do not contain any selectors used by local subscriptions, and subscription topic strings that contain wildcards might be simplified. This can result in publications matching proxy subscriptions where the actual subscriptions do not, resulting in additional publication flow between queue managers. The queue manager hosting the subscriptions filters out such discrepancies so that additional publications are not returned to the subscriptions.

Proxy subscription aggregation

Proxy subscriptions are aggregated using a duplicate elimination system. For a particular resolved topic string, a proxy subscription is sent on the first local subscription or received proxy subscription. Subsequent subscriptions to the same topic string make use of this existing proxy subscription.

The proxy subscription is canceled after the last local subscription or received proxy subscription is canceled.

Publication aggregation

When there is more than one subscription to the same topic string on a queue manager, only a single copy of each publication matching that topic string is sent from other queue managers in the publish/subscribe topology. On arrival of the message, the local queue manager delivers a copy of the message to each matching subscription.

It is possible for more than one proxy subscription to match the topic string of a single publication when the proxy subscriptions contain wildcards. If a message is published on a queue manager that matches two or more proxy subscriptions created by a single connected queue manager, only one copy of the publication is forwarded to the remote queue manager to satisfy the multiple proxy subscriptions.

Related information:

Loop detection in a distributed publish/subscribe network

Wildcards in proxy subscriptions:

Subscriptions can use wildcards in topic strings to match against multiple topic strings in publications.

There are two wildcard schemas that a subscription can use: *topic-based* and *character-based*. See “Wildcard schemes” on page 212.

In IBM WebSphere MQ Version 7.0 and later versions, all proxy subscriptions for wildcard subscriptions are converted to use topic-based wildcards. If a character-based wildcard is found, it is replaced with a # character, back to the nearest /. For example, /aaa/bbb/c*d is converted to /aaa/bbb/#. The conversion results in remote queue managers sending slightly more publications than were explicitly subscribed to. The additional publications are filtered out by the local queue manager, when it delivers the publications to its local subscribers.

Controlling wildcard usage with the WILDCARD property

Use the MQSC **Topic WILDCARD** property or the equivalent PCF Topic W i l d c a r d O p e r a t i o n property to control the delivery of publications to subscriber applications that use wildcard topic string names. The WILDCARD property can have one of two possible values:

WILDCARD

The behavior of wildcard subscriptions with respect to this topic.

PASSTHRU

Subscriptions made to a wildcarded topic less specific than the topic string at this topic object receive publications made to this topic and to topic strings more specific than this topic.

BLOCK

Subscriptions made to a wildcarded topic less specific than the topic string at this topic object do not receive publications made to this topic or to topic strings more specific than this topic.

The value of this attribute is used when subscriptions are defined. If you alter this attribute, the set of topics covered by existing subscriptions is not affected by the modification. This scenario applies also if the topology is changed when topic objects are created or deleted; the set of topics matching subscriptions created following the modification of the WILDCARD attribute is created using the modified topology. If you want to force the matching set of topics to be re-evaluated for existing subscriptions, you must restart the queue manager.

In the example, “Example: Create the Sport publish/subscribe cluster” on page 225, you can follow the steps to create the topic tree structure shown in Figure 23 on page 222.

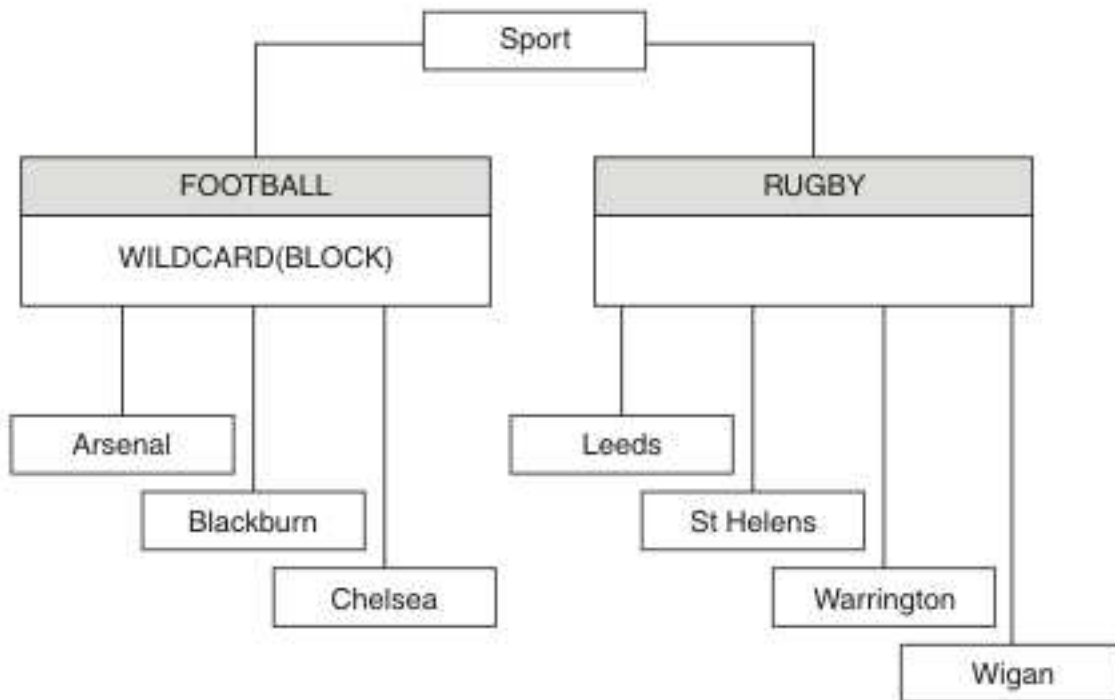


Figure 37. A topic tree that uses the WILDCARD property, BLOCK

A subscriber using the wildcard topic string # receives all publications to the Sport topic and the Sport/Rugby subtree. The subscriber receives no publications to the Sport/Football subtree, because the WILDCARD property value of the Sport/Football topic is BLOCK.

PASSTHRU is the default setting. You can set the WILDCARD property value PASSTHRU to nodes in the Sport tree. If the nodes do not have the WILDCARD property value BLOCK, setting PASSTHRU does not alter the behavior observed by subscribers to nodes in the Sports tree.

In the example, create subscriptions to see how the wildcard setting affects the publications that are delivered; see Figure 27 on page 227. Run the publish command in Figure 30 on page 228 to create some

publications.

pub QMA

Figure 38. Publish to QMA

The results are shown in Table 8 on page 223. Notice how setting the WILDCARD property value BLOCK, prevents subscriptions with wildcards from receiving publications to topics within the scope of the wildcard.

Table 11. Publications received on QMA

Subscription	Topic string	Publications received	Notes
SPORTS	Sports/#	Sports Sports/Rugby Sports/Rugby/Leeds	All publications to Football subtree blocked by WILDCARD(BLOCK) on Sports/Football
SARSENAL	Sports/#/Arsenal	-	WILDCARD(BLOCK) on Sports/Football prevents wildcard subscription on Arsenal
SLEEDS	Sports/#/Leeds	Sports/Rugby/Leeds	Default WILDCARD on Sports/Rugby does not prevent wildcard subscription on Leeds.

Note:

Suppose a subscription has a wildcard that matches a topic object with the WILDCARD property value BLOCK. If the subscription also has a topic string to the right of the matching wildcard, the subscription never receives a publication. The set of publications that are not blocked are publications to topics that are parents of the blocked wildcard. Publications to topics that are children of the topic with the BLOCK property value are blocked by the wildcard. Therefore subscription topic strings that include a topic to the right of the wildcard never receive any publications to match.

Setting the WILDCARD property value to BLOCK does not mean you cannot subscribe using a topic string that includes wildcards. Such a subscription is normal. The subscription has an explicit topic that matches the topic with a topic object having a WILDCARD property value BLOCK. It uses wildcards for topics that are parents or children of the topic with the WILDCARD property value BLOCK. In the example in Figure 23 on page 222, a subscription such as Sports/Football/# can receive publications.

Wildcards and cluster topics

Cluster topic definitions are propagated to every queue manager in a cluster. A subscription to a cluster topic at one queue manager in a cluster results in the queue manager creating proxy subscriptions. A proxy subscription is created at every other queue manager in the cluster. Subscriptions using topic strings containing wildcards, combined with cluster topics, can give hard to predict behavior. The behavior is explained in the following example.

In the cluster set up for the example, “Example: Create the Sport publish/subscribe cluster” on page 225, QMB has the same set of subscriptions as QMA, yet QMB received no publications after the publisher published to QMA, see Figure 24 on page 223. Although the Sports/Football and Sports/Rugby topics are cluster topics, the subscriptions defined in fullsubs.tst do not reference a cluster topic. No proxy subscriptions are propagated from QMB to QMA. Without proxy subscriptions, no publications to QMA are forwarded to QMB.

Some of the subscriptions, such as Sports/#/Leeds, might seem to reference a cluster topic, in this case Sports/Rugby. The Sports/#/Leeds subscription actually resolves to the topic object SYSTEM.BASE.TOPIC.

The rule for resolving the topic object referenced by a subscription such as, Sports/#/Leeds is as follows. Truncate the topic string to the first wildcard. Scan left through the topic string looking for the first topic that has an associated administrative topic object. The topic object might specify a cluster name, or define a local topic object. In the example, Sports/#/Leeds, the topic string after truncation is Sports, which has no topic object, and so Sports/#/Leeds inherits from SYSTEM.BASE.TOPIC, which is a local topic object.

To see how subscribing to clustered topics can change the way wildcard propagation works, run the batch script, upsubs.bat. The script clears the subscription queues, and adds the cluster topic subscriptions in fullsubs.tst. Run puba.bat again to create a batch of publications; see Figure 24 on page 223.

Table 9 on page 224 shows the result of adding two new subscriptions to the same queue manager that the publications were published on. The result is as expected, the new subscriptions receive one publication each, and the numbers of publications received by the other subscriptions are unchanged. The unexpected results occur on the other cluster queue manager; see Table 10 on page 224.

Table 12. Publications received on QMA

Subscription	Topic string	Publications received	Notes
SPORTS	Sports/#	Sports Sports/Rugby Sports/Rugby/Leeds	All publications to Football subtree blocked by WILDCARD(BLOCK) on Sports/Football
SARSENAL	Sports/#/Arsenal	-	WILDCARD(BLOCK) on Sports/Football prevents wildcard subscription on Arsenal
SLEEDS	Sports/#/Leeds	Sports/Rugby/Leeds	Default WILDCARD on Sports/Rugby does not prevent wildcard subscription on Leeds.
FARSENAL	Sports/Football/Arsenal	Sports/Football/Arsenal	Arsenal receives a publication because the subscription does not have a wildcard.
FLEEDS	Sports/Rugby/Leeds	Sports/Rugby/Leeds	Leeds would receive a publication in any event.

Table 10 on page 224 shows the results of adding the two new subscriptions on QMB and publishing on QMA. Recall that QMB received no publications without these two new subscriptions. As expected, the two new subscriptions receive publications, because Sports/FootBall and Sports/Rugby are both cluster topics. QMB forwarded proxy subscriptions for Sports/Football/Arsenal and Sports/Rugby/Leeds to QMA, which then sent the publications to QMB.

The unexpected result is that the two subscriptions Sports/# and Sports/#/Leeds that previously received no publications, now receive publications. The reason is that the Sports/Football/Arsenal and Sports/Rugby/Leeds publications forwarded to QMB for the other subscriptions are now available for any subscriber attached to QMB. Consequently the subscriptions to the local topics Sports/# and Sports/#/Leeds receive the Sports/Rugby/Leeds publication. Sports/#/Arsenal continues not to receive a publication, because Sports/Football has its WILDCARD property value set to BLOCK.

Table 13. Publications received on QMB

Subscription	Topic string	Publications received	Notes
SPORTS	Sports/#	Sports/Rugby/Leeds	All publications to Football subtree blocked by WILDCARD(BLOCK) on Sports/Football
SARSENAL	Sports/#/Arsenal	-	WILDCARD(BLOCK) on Sports/Football prevents wildcard subscription on Arsenal
SLEEDS	Sports/#/Leeds	Sports/Rugby/Leeds	Default WILDCARD on Sports/Rugby does not prevent wildcard subscription on Leeds.
FARSENAL	Sports/Football/Arsenal	Sports/Football/Arsenal	Arsenal receives a publication because the subscription does not have a wildcard.
FLEEDS	Sports/Rugby/Leeds	Sports/Rugby/Leeds	Leeds would receive a publication in any event.

In most applications, it is undesirable for one subscription to influence the behavior of another subscription. One important use of the WILDCARD property with the value BLOCK is to make the subscriptions to the same topic string containing wildcards behave uniformly. Whether the subscription is on the same queue manager as the publisher, or a different one, the results of the subscription are the same.

Wildcards and streams

For a new application written to the publish/subscribe API, the effect is that a subscription to * receives no publications. To receive all the Sports publications you must subscribe to Sports/*, or Sports/#, and similarly for Business publications.

The behavior of an existing queued publish/subscribe application does not change when the publish/subscribe broker is migrated to IBM MQ Version 7 and later versions. The **StreamName** property in the **Publish**, **Register Publisher**, or **Subscriber** commands is mapped to the name of the topic the stream has been migrated to.

Wildcards and subscription points

For a new application written to the publish/subscribe API, the effect of the migration is that a subscription to * receives no publications. To receive all the Sports publications you must subscribe to Sports/*, or Sports/#, and similarly for Business publications.

The behavior of an existing queued publish/subscribe application does not change when the publish/subscribe broker is migrated to IBM MQ Version 7 and later versions. The **SubPoint** property in the **Publish**, **Register Publisher**, or **Subscriber** commands is mapped to the name of the topic the subscription has been migrated to.

Example: Create the Sport publish/subscribe cluster

The steps that follow create a cluster, CL1, with four queue managers: two full repositories, CL1A and CL1B, and two partial repositories, QMA and QMB. The full repositories are used to hold only cluster definitions. QMA is designated the cluster topic host. Durable subscriptions are defined on both QMA and QMB.

Note: The example is coded for Windows. You must recode Create qmgrs.bat and create pub.bat to configure and test the example on other platforms.

1. Create the script files.
 - a. Create topics.tst

- b. Create wildsubs.tst
 - c. Create fullsubs.tst
 - d. Create qmgrs.bat
 - e. create pub.bat
2. Run Create qmgrs.bat to create the configuration.
- qmgrs

Create the topics in Figure 23 on page 222. The script in figure 5 creates the cluster topics Sports/Football and Sports/Rugby.

Note: The REPLACE option does not replace the TOPICSTR properties of a topic. TOPICSTR is a property that is usefully varied in the example to test different topic trees. To change topics, delete the topic first.

```
DELETE TOPIC ('Sports')
DELETE TOPIC ('Football')
DELETE TOPIC ('Arsenal')
DELETE TOPIC ('Blackburn')
DELETE TOPIC ('Chelsea')
DELETE TOPIC ('Rugby')
DELETE TOPIC ('Leeds')
DELETE TOPIC ('Wigan')
DELETE TOPIC ('Warrington')
DELETE TOPIC ('St. Helens')

DEFINE TOPIC ('Sports') TOPICSTR('Sports')
DEFINE TOPIC ('Football') TOPICSTR('Sports/Football') CLUSTER(CL1) WILDCARD(BLOCK)
DEFINE TOPIC ('Arsenal') TOPICSTR('Sports/Football/Arsenal')
DEFINE TOPIC ('Blackburn') TOPICSTR('Sports/Football/Blackburn')
DEFINE TOPIC ('Chelsea') TOPICSTR('Sports/Football/Chelsea')
DEFINE TOPIC ('Rugby') TOPICSTR('Sports/Rugby') CLUSTER(CL1)
DEFINE TOPIC ('Leeds') TOPICSTR('Sports/Rugby/Leeds')
DEFINE TOPIC ('Wigan') TOPICSTR('Sports/Rugby/Wigan')
DEFINE TOPIC ('Warrington') TOPICSTR('Sports/Rugby/Warrington')
DEFINE TOPIC ('St. Helens') TOPICSTR('Sports/Rugby/St. Helens')
```

Figure 39. Delete and create topics: topics.tst

Note: Delete the topics, as REPLACE does not replace topic strings.

Create subscriptions with wildcards. The wildcards corresponding the topics with topic objects in Figure 23 on page 222. Create a queue for each subscription. The queues are cleared and the subscriptions deleted when the script is run or rerun.

Note: The REPLACE option does not replace TOPICOBJ or TOPICSTR properties of a subscription. TOPICOBJ or TOPICSTR are the properties that are usefully varied in the example to test different subscriptions. To change them, delete the subscription first.

```

DEFINE QLOCAL(QSPORTS) REPLACE
DEFINE QLOCAL(QSARSENAL) REPLACE
DEFINE QLOCAL(QSLEEDS) REPLACE
CLEAR QLOCAL(QSPORTS)
CLEAR QLOCAL(QSARSENAL)
CLEAR QLOCAL(QSLEEDS)

DELETE SUB (SPORTS)
DELETE SUB (SARSENAL)
DELETE SUB (SLEEDS)
DEFINE SUB (SPORTS) TOPICSTR('Sports/#') DEST(QSPORTS)
DEFINE SUB (SARSENAL) TOPICSTR('Sports+/Arsenal') DEST(QSARSENAL)
DEFINE SUB (SLEEDS) TOPICSTR('Sports+/Leeds') DEST(QSLEEDS)

```

Figure 40. Create wildcard subscriptions: wildsubs.tst

Create subscriptions that reference the cluster topic objects.

Note:

The delimiter, /, is automatically inserted between the topic string referenced by TOPICOBJ, and the topic string defined by TOPICSTR.

The definition, DEFINE SUB(FARSENAL) TOPICSTR('Sports/Football/Arsenal') DEST(QFARSENAL) creates the same subscription. TOPICOBJ is used as a quick way to reference topic string you have already defined. The subscription, when created, no longer refers to the topic object.

```

DEFINE QLOCAL(QFARSENAL) REPLACE
DEFINE QLOCAL(QRLEEDS) REPLACE
CLEAR QLOCAL(QFARSENAL)
CLEAR QLOCAL(QRLEEDS)

DELETE SUB (FARSENAL)
DELETE SUB (RLEEDS)
DEFINE SUB (FARSENAL) TOPICOBJ('Football') TOPICSTR('Arsenal') DEST(QFARSENAL)
DEFINE SUB (RLEEDS) TOPICOBJ('Rugby') TOPICSTR('Leeds') DEST(QRLEEDS)

```

Figure 41. Delete and create subscriptions: fullsubs.tst

Create a cluster with two repositories. Create two partial repositories for publishing and subscribing. Rerun the script to delete everything and start again. The script also creates the topic hierarchy, and the initial wildcard subscriptions.

Note:

On other platforms, write a similar script, or type all the commands. Using a script makes it quick to delete everything and start again with an identical configuration.

```

@echo off
set port.CL1B=1421
set port.CL1A=1420
for %%A in (CL1A CL1B QMA QMB) do call :createQM %%A
call :configureQM CL1A CL1B %port.CL1B% full
call :configureQM CL1B CL1A %port.CL1A% full
for %%A in (QMA QMB) do call :configureQM %%A CL1A %port.CL1A% partial
for %%A in (topics.tst wilddsubs.tst) do runmqsc QMA < %%A
for %%A in (wilddsubs.tst) do runmqsc QMB < %%A
goto:eof

:createQM
echo Configure Queue manager %1
endmqm -p %1
for %%B in (dlt crt str) do %%Bmqm %1
goto:eof

:configureQM
if %1==CL1A set p=1420
if %1==CL1B set p=1421
if %1==QMA set p=1422
if %1==QMB set p=1423
echo configure %1 on port %p% connected to repository %2 on port %3 as %4 repository
echo DEFINE LISTENER(LST%1) TRPTYPE(TCP) PORT(%p%) CONTROL(QMGR) REPLACE | runmqsc %1
echo START LISTENER(LST%1) | runmqsc %1
if full=%4 echo ALTER QMGR REPOS(CL1) DEADQ(SYSTEM.DEAD.LETTER.QUEUE) | runmqsc %1
echo DEFINE CHANNEL(TO.%2) CHLTYPE(CLUSSDR) TRPTYPE(TCP) CONNAME('LOCALHOST(%3)') CLUSTER(CL1) REPLACE | runmqsc %1
echo DEFINE CHANNEL(TO.%1) CHLTYPE(CLUSRCVR) TRPTYPE(TCP) CONNAME('LOCALHOST(%p%)') CLUSTER(CL1) REPLACE | runmqsc %1
goto:eof

```

Figure 42. Create queue managers: qmgrs.bat

Update the configuration by adding the subscriptions to the cluster topics.

```

@echo off
for %%A in (QMA QMB) do runmqsc %%A < wilddsubs.tst
for %%A in (QMA QMB) do runmqsc %%A < upsubs.tst

```

Figure 43. Update subscriptions: upsubs.bat

Run pub.bat, with a queue manager as a parameter, to publish messages containing the publication topic string. Pub.bat uses the sample program **amqspub**.

```

@echo off
@rem Provide queue manager name as a parameter
set S=Sports
set S=6 Sports/Football Sports/Football/Arsenal
set S=6 Sports/Rugby Sports/Rugby/Leeds
for %%B in (6) do echo %%B | amqspub %%B %1

```

Figure 44. Publish: pub.bat

Related information:

Wildcard subscriptions and retained publications

Publication scope

When you configure a publish/subscribe cluster or hierarchy, the scope of a publication further controls whether queue managers forward a publication to remote queue managers. Use the **PUBSCOPE** topic attribute to administer the scope of publications.

If a publication is not forwarded to remote queue managers, only local subscribers receive the publication.

When you use a publish/subscribe cluster, the scope of publications is primarily controlled by the definition of clustered topic objects at certain points in the topic tree. Publication scope must be set to allow the flow of publications to other queue managers in the cluster. You should only restrict publication scope for a clustered topic when you need fine-grain control of specific topics on certain queue managers.

When you use a publish/subscribe hierarchy, the scope of publications is primarily controlled by this attribute in combination with the subscription scope attribute.

The **PUBSCOPE** attribute is used to determine the scope of publications made to a specific topic. You can set the attribute to one of the following values:

QMGR

The publication is delivered only to local subscribers. These publications are called *local publications*. Local publications are not forwarded to remote queue managers and therefore are not received by subscribers connected to remote queue managers.

ALL The publication is delivered to local subscribers and subscribers connected to remote queue managers in a publish/subscribe cluster or hierarchy. These publications are called *global publications*.

ASPARENT

Use the **PUBSCOPE** setting of the parent topic in the topic tree.

Publishers can also specify whether a publication is local or global using the MQPMO_SCOPE_QMGR put message option. If this option is used, it overrides any behavior that has been set using the **PUBSCOPE** topic attribute.

Related concepts:

“Administrative topic objects” on page 219

Using an administrative topic object, you can assign specific, non-default attributes to topics.

Related information:

Configuring distributed publish/subscribe networks

Subscription scope

The scope of a subscription controls whether a subscription on one queue manager receives publications that are published on another queue manager in a publish/subscribe cluster or hierarchy, or only publications from local publishers.

Limiting the subscription scope to a queue manager stops proxy subscriptions from being forwarded to other queue managers in the publish/subscribe topology. This reduces inter-queue manager publish/subscribe messaging traffic.

When you use a publish/subscribe cluster, the scope of subscriptions is primarily controlled by the definition of clustered topic objects at certain points in the topic tree. Subscription scope must be set to

allow the flow of proxy subscriptions to other queue managers in the cluster. You should only restrict subscription scope for a clustered topic when you need fine-grain control of specific topics on certain queue managers.

When you use a publish/subscribe hierarchy, the scope of subscriptions is primarily controlled by this attribute in combination with the publication scope attribute.

The **SUBSCOPE** topic attribute is used to determine the scope of subscriptions made to a specific topic. You can set the attribute to one of the following values:

QMGR

A subscription receives only local publications, and proxy subscriptions are not propagated to remote queue managers.

ALL A proxy subscription is propagated to remote queue managers in a publish/subscribe cluster or hierarchy, and the subscriber receives local and remote publications.

ASPARENT

Use the **SUBSCOPE** setting of the parent topic in the topic tree.

When subscription scope for a topic is set to ALL, either directly or resolved through ASPARENT, individual subscriptions to that topic can restrict their scope to QMGR by specifying MQSO_SCOPE_QMGR when creating the subscription. A subscription to a topic that has a scope of QMGR cannot widen the scope to ALL.

Related concepts:

“Administrative topic objects” on page 219

Using an administrative topic object, you can assign specific, non-default attributes to topics.

Related information:

Configuring distributed publish/subscribe networks

Topic spaces

A topic space is the set of topics on which you can subscribe and publish. A queue manager in a distributed publish/subscribe topology has a topic space that potentially includes topics that have been subscribed and published to on connected queue managers in that topology.

Note: For an overview of topics within a queue manager, such as administrative topic objects, topic strings and topic trees, see “Topics” on page 210. Further references to *topics* in the current article refer to *topic strings* unless otherwise specified.

Topics are initially created in either of the following ways:

- administratively, when you define a topic object or durable subscription.
- dynamically, when an application creates a publication or subscription dynamically to a new topic.

Topics are propagated to other queue managers both through proxy subscriptions, and by creating administrative cluster topic objects. Proxy subscriptions result in publications being forwarded from the queue manager to which a publisher is connected, to the queue managers of subscribers.

Proxy subscriptions are propagated between all queue managers that are connected together by parent-child relationships in a queue manager hierarchy. The result is, you can subscribe on one queue manager to a topic defined on any other queue manager in the hierarchy. As long as there is a connected path between the queue managers, it does not matter how the queue managers are connected.

Proxy subscriptions are also propagated for subscriptions to cluster topics in a publish/subscribe cluster. A cluster topic is a topic that is attached to a topic object that has the **CLUSTER** attribute, or inherits the attribute from its parent. Topics that are not cluster topics are known as local topics and are not replicated to the cluster. No proxy subscriptions are propagated to the cluster from subscriptions to local topics.

To summarize, proxy subscriptions are created for subscribers in two circumstances.

1. A queue manager is a member of a hierarchy, and a proxy subscription is forwarded to the parent and children of the queue manager.
2. A queue manager is a member of a cluster, and the subscription topic string resolves to a topic that is associated with a cluster topic object. When the topic is a *direct routed* cluster topic, proxy subscriptions are forwarded to all members of the cluster. When the topic is a *topic host routed* cluster topic, proxy subscriptions are forwarded only to the queue managers in the cluster that have defined the clustered topic object. For more information see “Publish/subscribe clusters” on page 235.

If a queue manager is a member of a cluster and a hierarchy, proxy subscriptions are propagated by both mechanisms without delivering duplicate publications to the subscriber.

The topics spaces of three publish/subscribe topologies are described in the following list:

- “Case 1. Publish/subscribe clusters.”
- “Case 2. Publish/subscribe hierarchies in version 7 or later versions” on page 250.
- “Case 3. Publish/subscribe hierarchies and streams in Version 6” on page 250.

In separate topics, the following configuration tasks describe how to combine topic spaces.

- Creating a single topic space in a publish/subscribe cluster.
- Adding a version 7 or later queue manager to existing Version 6 topic spaces.
- Combining the topic spaces of multiple clusters.
- Combining and isolating topic spaces in multiple clusters.
- Publishing and subscribing to topic spaces in multiple clusters.

Case 1. Publish/subscribe clusters

In the example, assume that the queue manager is not connected to a publish/subscribe hierarchy.

If a queue manager is a member of a publish/subscribe cluster, its topic space is made up from local topics and cluster topics. Local topics are associated with topic objects without the **CLUSTER** attribute. If a queue manager has local topic object definitions, its topic space is different from another queue manager in the cluster that also has its own locally defined topic objects.

In a publish/subscribe cluster, you cannot subscribe to a topic defined on another queue manager, unless the topic you subscribe to resolves to a cluster topic object.

When the same named definitions of a cluster topic object are required on multiple queue managers, for example when using *topic host routing*, it is important that all definitions match where necessary. For more information, see Creating a single topic space in a publish/subscribe cluster.

A local definition of a topic object, whether the definition is for a cluster topic or a local topic, takes precedence over the same topic object defined elsewhere in the cluster. The locally defined topic is used, even if the object defined elsewhere is more recent.

It is important that a cluster topic object is associated with the same topic string everywhere in the cluster. You cannot modify the topic string with which a topic object is associated. To associate the same topic object with a different topic string, you must delete the topic object and re-create it with the new topic string. If the topic is clustered, the effect is to delete the copies of the topic object stored on the other members of the cluster and then to create copies of the new topic object everywhere in the cluster. The copies of the topic object all refer to the same topic string.

It is possible to accidentally create two definitions of the same named topic object on different queue managers in the cluster, with different topic strings. This can result in confusing behavior, because multiple definitions of the same topic object with different topic strings can produce different results

depending how and where the topic is referenced. See Multiple cluster topic definitions of the same name for more information on this important point.

Case 2. Publish/subscribe hierarchies in version 7 or later versions

In the example, assume that the queue manager is not a member of a publish/subscribe cluster.

In version 7 or later, if a queue manager is a member of a publish/subscribe hierarchy, its topic space consists of all the topics defined locally and on connected queue managers. The topic space of all the queue managers in a hierarchy is the same. There is no division of topics into local topics and global topics.

Set either of the **PUBSCOPE** and **SUBSCOPE** options to QMGR, to prevent a publication on a topic flowing from a publisher to a subscriber connected to different queue managers in the hierarchy.

Suppose you define a topic object Alabama with the topic string USA/Alabama on queue manager QMA. The result is as follows:

1. The topic space at QMA now includes the topic object Alabama and the topic string USA/Alabama.
2. An application or administrator can create a subscription at QMA using the topic object name Alabama.
3. An application can create a subscription to any topic, including USA/Alabama, at any queue manager in the hierarchy. If QMA has not been defined locally, the topic USA/Alabama resolves to the topic object SYSTEM.BASE.TOPIC.

Case 3. Publish/subscribe hierarchies and streams in Version 6

Before version 7, the topic space was divided into separate streams, which included the default stream that was present on all queue managers. Publications cannot flow between different streams. If named streams are used, the topic spaces at different queue managers might be different. Topics are divided into topics in the default stream, and topics in different named streams.

Note: Each named stream forms a separate topic space. To form a connected topology each named stream must exist on the connected queue managers. Suppose stream X is defined on QMA and QMC, but not on QMB. If QMA is the parent of QMB, and QMB is the parent of QMC, no topics in stream X can flow between QMA and QMC.

Setting both of the **PUBSCOPE** and **SUBSCOPE** options either to QMGR or to ALL requires a publisher and subscriber to a topic to exchange only publications for local consumption, or to exchange only publications for global consumption.

From version 7, streams are not available using the publish/subscribe API. If you use queued publish/subscribe on a version 7 queue manager, streams are mapped to different topic objects that can simulate the effect of streams. A stream is simulated by creating a topic object that is the root topic for all the topics in the stream. The queue manager maps publications and subscriptions between the stream and the corresponding root topic of each tree.

Related information:

Configuring distributed publish/subscribe networks

Publication scope

Subscription scope

IBM MQ Multicast

IBM MQ Multicast offers low latency, high fan out, reliable multicast messaging.

Multicast is an efficient form of publish/subscribe messaging as it can be scaled to a high number of subscribers without detrimental effects in performance. IBM MQ enables reliable Multicast messaging by using acknowledgments, negative acknowledgments, and sequence numbers to achieve low latency messaging with high fan out.

IBM MQ Multicast's fair delivery enables near simultaneous delivery, ensuring that no recipient gains an advantage. As IBM MQ Multicast uses the network to deliver messages, a publish/subscribe engine is not needed to fan-out data. After a topic is mapped to a group address, there is no need for a queue manager because publishers and subscribers can operate in a peer-to-peer mode. This allows the load to be reduced on queue manager servers, and the queue manager server is no longer a potential point of failure.

Initial multicast concepts

IBM MQ Multicast can be easily integrated into existing systems and applications by using the Communication Information (COMMINFO) object. Two TOPIC object fields enable the quick configuration of existing TOPIC objects to support or ignore multicast traffic.

Objects needed for multicast

The following information is a brief overview of the two objects needed for IBM MQ Multicast:

COMMINFO object

The COMMINFO object contains the attributes associated with multicast transmission. For more information about the COMMINFO object parameters, see DEFINE COMMINFO.

The only COMMINFO field that **MUST** be set is the name of the COMMINFO object. This name is then used to identify the COMMINFO object to a topic. The **GRPADDR** field of the COMMINFO object must be checked to ensure that the value is a valid multicast group address.

TOPIC object

A topic is the subject of the information that is published in a publish/subscribe message, and a topic is defined by creating a TOPIC object. For more information about the TOPIC object parameters, see DEFINE TOPIC.

Existing topics can be used with multicast by changing the values of the following TOPIC object parameters: **COMMINFO** and **MCAST**.

- **COMMINFO** This parameter specifies the name of the multicast communication information object.
- **MCAST** This parameter specifies whether multicast is allowable at this position in the topic tree. By default, **MCAST** is set to **ASPARENT** meaning that the multicast attribute of the topic is inherited from the parent. Setting **MCAST** to **ENABLED** allows multicast traffic at this node.

Multicast networks and topics

The following information is an overview of what happens to subscriptions with different types of subscription and topic definition. These examples all assume that the TOPIC object **COMMINFO** parameter is set to the name of a valid COMMINFO object:

Topic set to multicast enabled

If the topic string **MCAST** parameter is set to **ENABLED**, subscriptions from multicast capable clients are allowed and a multicast subscription is made unless:

- It is a durable subscription from a multicast capable client.
- It is a non-managed subscription from a multicast capable client.
- It is a subscription from a non-multicast capable client.

In these cases a non-multicast subscription is made and subscriptions are downgraded to normal publish/subscribe.

Topic set to multicast disabled

If the topic string **MCAST** parameter is set to **DISABLED**, a non-multicast subscription is always made and subscriptions are downgraded to normal publish/subscribe.

Topic set to multicast only

If the topic string **MCAST** parameter is set to **ONLY**, subscriptions from multicast capable clients are allowed and a multicast subscription is made unless:

- It is a durable subscription: Durable subscriptions are rejected with reason code 2436 (0984) (RC2436): MQRC_DURABILITY_NOT_ALLOWED
- It is a non-managed subscription: Non-managed subscriptions are rejected with reason code 2046 (07FE) (RC2046): MQRC_OPTIONS_ERROR
- It is a subscription from a non-multicast capable client: These subscriptions are rejected with reason code 2560 (0A00) (RC2560): MQRC_MULTICAST_ONLY
- It is a subscription from a locally bound application: These subscriptions are rejected with reason code 2560 (0A00) (RC2560): MQRC_MULTICAST_ONLY

MQ Telemetry



MQ Telemetry comprises a telemetry (MQXR) service that is part of a queue manager, telemetry clients that you can write yourself or download for free, and command line and explorer administrative interfaces. Telemetry refers to collecting data from and administering a wide range of remote devices. With MQ Telemetry you can integrate the collection of data and control of devices with web applications.

MQ Telemetry is a component of IBM WebSphere MQ Version 7.1, and later. Upgrading for these versions is essentially installing a later version of IBM MQ.

However, from IBM MQ Version 8.0, the Client Software Development Kit (SDK) is no longer supplied as part of the product. Instead, the current version of the SDK is available as the IBM Messaging Telemetry Clients SupportPac.

MQ Telemetry can either be installed with the main product, or installed after Version 7.1, or later, is already installed. For migration information, see Migrating MQ Telemetry to the latest version of the product.

Included in MQ Telemetry are the following components:

Installer

MQ Telemetry is installed using either a GUI or command-line installer.

Telemetry channels

Use telemetry channels to manage the connection of MQTT clients to IBM MQ . Telemetry channels use new IBM MQ objects, such as the `SYSTEM.MQTT.TRANSMIT.QUEUE`, to interact with IBM MQ.

Telemetry (MQXR) service

MQTT clients use the `SYSTEM.MQXR.SERVICE` telemetry service to connect to telemetry channels.

IBM MQ Explorer support for MQ Telemetry

MQ Telemetry can be administered using IBM MQ Explorer.

Documentation

MQ Telemetry documentation is included in the standard IBM MQ product documentation from Version 7.1 . SDK documentation for Java and C clients is provided in the product documentation, and as Javadoc and HTML.

Telemetry concepts

You collect information from the environment all around you to decide what to do. As a consumer, you check what you have in store, before deciding about what food to buy. You want to know how long a journey is going to take if you leave now, before booking a connection. You check your symptoms, before deciding whether to visit the doctor. You check when a bus is going to arrive, before deciding whether to wait. The information for those decisions comes directly from meters and devices, from the written word on paper or from a screen, and from you. Where ever you are, and when ever you need to, you collect information, bring it together, analyze it, and act upon it.

If the sources of information are widely dispersed or inaccessible, it becomes difficult and costly to collect the most accurate information. If there are many changes you want to make, or it is difficult to make the changes, then the changes do not get made, or are made when they are less effective.

What if the costs of collecting information from, and controlling, widely dispersed devices is greatly reduced by connecting the devices with digital technology to the internet? The information can be analyzed using the resources of the internet and the enterprise. You have more opportunities to make informed decisions and act upon them.

Technological trends, and environmental and economic pressures, are driving these changes to happen:

1. The cost of connecting and controlling sensors and actuators is reducing, due to standardization and connection to low cost digital processors.
2. The internet, and internet technologies, are increasingly used to connect devices. In some countries, mobile phones exceed personal computers in the number of connections to internet applications. Other devices are surely following.
3. The internet, and internet technologies, make it much easier for an application to get data. Easy access to data is driving the use of data analytics to turn data from sensors into information that is useful in many more solutions.
4. Intelligent use of resources is often a quicker and cheaper way of reducing carbon emissions and costs. The alternatives: finding new resources, or developing new technologies to use existing resources, might be the long-term solution. In the short term developing new technologies, or finding new resources, is often riskier, slower, and more costly, than improving existing solutions.

Example

An example shows how these trends create new opportunities to interact with the environment intelligently.

The International Convention for the Safety of Life at Sea (SOLAS) requires Automatic Identification System (AIS) to be deployed on many ships. It is required on merchant ships over 300 tons and passenger ships. AIS is primarily a collision avoidance system for coastal shipping. It is used by marine authorities to monitor and control coastal waters.

Enthusiasts around the world are deploying low-cost AIS tracking stations and placing coastal shipping information onto the internet. Other enthusiasts are writing applications that combine information from AIS with other information from the internet. The results are put on Web sites, and published using Twitter and SMS.

In one application, information from AIS stations near Southampton is combined with ship ownership and geographical information. The application feeds live information about ferry arrivals and departures to Twitter. Regular commuters using the ferries between Southampton and the Isle of Wight subscribe to the news feed using Twitter or SMS. If the feed shows their ferry is running late, commuters can delay their departure and catch the ferry when it docks later than its scheduled arrival time.

For more examples, see “Telemetry use cases” on page 255.

Related information:

Installing MQ Telemetry

Administering MQ Telemetry

Migrating MQ Telemetry to the latest version of the product

Developing applications for MQ Telemetry

MQ Telemetry Reference

MQ Telemetry troubleshooting

Introduction to MQ Telemetry



People, businesses, and governments increasingly want to use MQ Telemetry to interact more smartly with the environment we live and work in. MQ Telemetry connects all kinds of devices to the internet and to the enterprise, and reduces the costs of building applications for smart devices.

What is MQ Telemetry?

- It is a feature of IBM MQ that extends the universal messaging backbone provided by IBM MQ to a wide range of remote sensors, actuators and telemetry devices. MQ Telemetry extends IBM MQ so that it can interconnect intelligent enterprise applications, services, and decision makers with networks of instrumented devices.
- The core parts of MQ Telemetry are:

The MQ Telemetry (MQXR) service.

This service runs inside the IBM MQ server, and uses the IBM MQ Telemetry Transport (MQTT) protocol to communicate with telemetry devices.

MQTT applications that you write.

These applications control the information that is carried between the telemetry devices and the IBM MQ queue manager, and any actions that are taken in response to that information. To help create these applications, you use MQTT client libraries.

What can it do for me?

- MQTT is an open messaging transport that allows MQTT implementations to be created for a wide variety of devices.
- MQTT clients can run on small footprint devices that have limited resources.
- MQTT works efficiently on networks where the bandwidth is low, where cost of sending data is expensive or which might be fragile.
- Message delivery is assured and decoupled from the application.
- Application programmers do not need to have communications programming knowledge.

- Messages can be exchanged with other messaging applications. These can be another telemetry application, or an MQI, JMS or enterprise messaging application.

How do I use it?

- Download the IBM Messaging Telemetry Clients SupportPac, which includes the MQTT client libraries.
- Use the IBM MQ Explorer and its associated tools to administer the telemetry feature of IBM MQ.
- Use the client libraries to help you create MQTT applications that connect to a queue manager, and that use publish/subscribe messaging.
- Distribute your application and client library to the device where your application is to run.

How does it work?

- MQTT is a publish subscribe protocol. An MQTT client application can publish messages to an MQTT server, or subscribe for messages that are sent by applications that connect to an MQTT server.
- MQTT client applications use client libraries that implement the MQTT message transport.
- A basic MQTT client application works like a standard MQ client but can run on a much wider variety of platforms and networks.
- The MQ Telemetry (MQXR) service turns an IBM MQ queue manager into an MQTT server.
- When an IBM MQ queue manager acts as the MQTT server, other applications that connect to the queue manager can subscribe for and receive the messages from the MQTT client.
- The queue manager acts as router distributing messages from publishing applications to subscribing applications.
- Messages can be distributed between different types of client applications. For instance, between Telemetry clients and JMS clients.

Note: MQ Telemetry replaces the SCADA nodes that were withdrawn in version 7 of WebSphere Message Broker (now known as IBM Integration Bus) and runs on Windows, Linux, and AIX.

Telemetry use cases



Telemetry is the automated sensing, measurement of data, and control of remote devices. The emphasis is on the transmission of data from devices to a central control point. Telemetry also includes sending configuration and control information to devices.

MQ Telemetry connects small devices by using the MQTT protocol, and connects the devices to other applications by using IBM MQ. MQ Telemetry bridges a gap between devices and the internet making it easier to build "smart solutions". Smart solutions unlock the wealth of information available on the internet, and in enterprise applications, for applications that monitor and control devices.

The following diagrams demonstrate some typical uses of MQ Telemetry:

Telemetry: Smart Electricity	
	<ul style="list-style-type: none"> • MQTT message containing energy usage data sent to service provider. • MQ Telemetry sends CONTROL COMMANDS based on analysis of energy usage data. • For more information, see the following use case: "Telemetry use case: Home energy monitoring and control" on page 258

Telemetry: Smart Health Services	
<ul style="list-style-type: none"> • MQ Telemetry sends Health Data to your Hospital & Doctor. • MQTT message alerts or feedback can be sent based on analysis of Health Data. • For more information, see the following use case: "Telemetry use case: Home patient monitoring" 	

Telemetry: One in a Crowd	
	<ul style="list-style-type: none"> • A simple card transaction is sent to the bank's server. • MQ Telemetry identifies the one person from the thousands, alerting the customer that their card has been used. • MQ Telemetry can use the simplest input of information, and locate that individual.

The use cases described in the subtopics are drawn from actual examples. They illustrate some ways of using telemetry, and some of the common problems that telemetry technology must resolve.

Telemetry use case: Home patient monitoring



In the collaboration between IBM and a healthcare provider on a cardiac patient care system, an implanted cardioverter defibrillator communicates with a hospital. Data about the patient and the implanted device are transferred using RF telemetry to the MQTT device in the home of a patient.

Typically the transfer takes place nightly to a transmitter located at the bedside. The transmitter transfers the data securely over the phone system to the hospital, where the data is analyzed.

The system reduces the number of visits a patient must make to a physician. It detects when the patient or device needs attention, and in the event of an emergency, it alerts the on-call physician.

The collaboration between IBM and the healthcare provider has characteristics that are common to a number of telemetry use cases:

Invisibility

The device requires no user intervention other than supplying power, a telephone line, and being in proximity to the device for part of the day. Its operation is reliable and simple to use.

To remove the need for the patient to set up the device, the device supplier preconfigures the device. The patient only must plug it in. Elimination of configuration by the patient simplifies the operation of the device and reduces the chance the device is configured wrongly.

The MQTT client is embedded as part of the device. The device developer embeds the MQTT client implementation in the device and the developer, or supplier, configures the MQTT client as part of the preconfiguration.

The MQTT client is shipped as a Java SE JAR file, which the developer includes in their Java application. For non-Java environments, such as this one, the device developer can implement a client in a different language using the published MQTT formats and protocol. Alternatively, the developer can use one of the C clients shipped as shared libraries for Windows, Linux and ARM platforms.

Uneven connectivity

Communication between the defibrillator and the hospital has uneven network characteristics. Two different networks are used to solve the different problems of collecting data from the patient, and sending the data to the hospital. Between the patient and the MQTT device, a short-range low-power RF network is used. The transmitter connects to the hospital using a VPN TCP/IP connection over a low-bandwidth phone-line.

It is often impractical to find a way to connect every device directly to an Internet Protocol network. Using two networks, connected by a hub, is a common solution. The MQTT device is a simple hub, storing information from the patient, and forwarding it to the hospital.

Security

The physician must be able to trust the authenticity of the patient data, and the patient wants the privacy of their data to be respected.

In some situations it is sufficient to encrypt the connection, using VPN or TLS. In other situations, it is desirable to keep the data secure even after it has been stored.

Sometimes the telemetry device is not secure. It might be in a shared dwelling, for example. The user of the device must be authenticated to make sure that the data is from the correct patient. The device itself can be authenticated to the server using TLS, and the server authenticated to the device.

The telemetry channel between the device and the queue manager supports JAAS for user authentication and TLS for communication encryption, and device authentication. Access to a publication is controlled by the object authority manager in IBM MQ.

The identifier used to authenticate the user can be mapped to a different identifier, such as a common patient identity. A common identifier simplifies configuring authorization to publication topics in IBM MQ.

Connectivity

The connection between the MQTT device and the hospital uses dial-up, and works with a bandwidth as low as 300 baud.

To operate effectively at 300 baud, the MQTT protocol adds only a few extra bytes to a message in addition to TCP/IP headers.

The MQTT protocol provides single transmission "fire and forget" messaging, which keeps latencies low. It can also use multiple transmissions to guarantee "at least once" and "exactly once" delivery if guaranteed delivery is more important than response time. To guarantee delivery, messages are stored at the device until they have been delivered successfully. If a device is connected wirelessly, guaranteed delivery is especially useful.

Scalability

Telemetry devices are typically deployed in large numbers, from tens of thousands to millions.

Connecting many devices to a system places large demands on a solution. There are business demands such as the cost of the devices and their software, and the administration demands of managing licenses, devices, and users. Technical demands include the load on the network, and on servers.

Opening connections uses more server resource than maintaining the open connections. But in a use case such as this that uses phone lines, the expense of connections means that connections are left open no longer than required. The data transfers are largely of a batched nature. The connections can be scheduled throughout the night to avoid a sudden peak of connections at bedtime.

On the client, the scalability of clients is helped by the minimal client configuration required. The MQTT client is embedded in the device. There is no requirement for a configuration or MQTT client license acceptance step to be built into the deployment of devices to patients.

On the server, MQ Telemetry has an initial target of 50,000 open connections per queue manager.

The connections are managed using the IBM MQ Explorer. The IBM MQ Explorer filters the connections to be displayed to a manageable number. With an appropriately chosen scheme of allocating identifiers to clients, you might filter connections based on geography, or alphabetically by patient name.

Telemetry use case: Home energy monitoring and control



Smart meters collect more detail about energy consumption than traditional meters.

Smart meters are often coupled with a local telemetry network to monitor and control individual appliances in a home. Some are also connected remotely for monitoring and control at a distance.

The remote connection could be set up by an individual, by a power utility, or by a central control point. The remote control point can read power usage and provide usage data. It can provide data to influence usage such as continuous pricing and weather information. It can limit load to improve overall power generation efficiency.

Smart meters are beginning to be deployed widely. The UK government, for instance, is in consultation about deployment of smart meters to every UK home by 2020.

Home metering use cases have a number of common characteristics:

Invisibility

Unless the user wants to be involved in saving energy by using the meter, the meter must not require user intervention. It must not reduce the reliability of the energy supply to individual appliances.

An MQTT client can be embedded in the software deployed with the meter, and does not require separate installation or configuration.

Uneven connectivity

The communication between appliances and the smart meter demands different standards of connectivity than between the meter and the remote connection point.

The connection from the smart meter to appliances must be highly available and conform to network standards for a home area network.

The remote network is likely to use various physical connections. Some of them, such as cellular, have a high transmission cost, and can be intermittent. The MQTT v3 specification is aimed at remote connections, and connections between local adapters and the smart meter.

Connection between power outlets and appliances, and the meter, use a home area network, such as Zigbee. MQTT for sensor networks (MQTT-S), is designed to work with Zigbee and other low bandwidth network protocols. MQ Telemetry does not support MQTT-S directly. It requires a gateway to connect MQTT-S to MQTT v3.

Like home patient monitoring, solutions for home energy monitoring and control require multiple networks, connected using the smart meter as a hub.

Security

There are a number of security issues associated with smart meters. These issues include non-repudiation of transactions, authorization of any control actions that are initiated, and privacy of power consumption data.

To ensure privacy, data transferred between the meter and the remote control point by MQTT can be encrypted using TLS. To ensure authorization of control actions, the MQTT connection between the meter and the remote control point can be mutually authenticated using TLS.

Connectivity

The physical nature of the remote network can vary considerably. It might use an existing broadband connection, or use a mobile network with high call costs, and intermittent availability. For high cost, intermittent, connections MQTT is an efficient and reliable protocol; see Home patient monitoring.

Scalability

Eventually power companies, or central control points, plan to deploy tens of millions of smart meters. Initially, the numbers of meters per deployment are in the tens to hundreds of thousands. This number is comparable to the initial MQTT target of 50,000 open client connections per queue manager.

A critical aspect of the architecture for home energy monitoring and control is to use the smart meter as a network concentrator. Each appliance adapter is a separate sensor. By connecting them to a local hub using MQTT, the hub can concentrate the data flows onto a single TCP/IP session with the central control point, and also store messages for a short period to overcome session outages.

Remote connections must be left open in home energy use cases for two reasons. First, because opening connections takes a long time relative to sending requests. The time to open many connections to send "load-limitation" requests in a short interval is too long. Second, to receive load-limitation requests from the power company, the connection must first be opened by the client. With MQTT, connections are always initiated by the client, and to receive load-limitation requests from the power company, the connection must be left open.

If the rate of opening connections is critical, or the server initiates time-critical requests, the solution is typically to maintain many open connections.

Telemetry use cases: Radio Frequency Identification (RFID)

AIX

Linux

Windows

RFID is the use of an embedded RFID tag to identify and track an object wirelessly. RFID tags can be read up to a range of several meters, and out of the line of sight of the RFID reader. Passive tags are activated by an RFID reader. Active tags transmit without external activation. Active tags must have a power source. Passive tags can include a power source to increase their range.

RFID is used in many applications, and the types of use cases vary enormously. RFID use cases, and home patient monitoring and home energy monitoring and control use cases, have some similarities and differences.

Invisibility

In many use cases, the RFID reader is deployed in large numbers and must work without user intervention. The reader includes an embedded MQTT client to communicate with a central control point.

For example, in a distribution warehouse, a reader uses a motion sensor to detect a pallet. It activates the RFID tags of items on the pallet and sends data and requests to central applications. The data is used to update the location of stock. The requests control what happens to the pallet next, such as moving it to a particular bay. Airlines, and airport baggage systems, are using RFID in this way.

In some RFID use cases, the reader has a standard computing environment, such as Java Platform, Micro Edition (Java ME). In these cases, the MQTT client might be deployed in a distinct configuration step, after manufacture.

Uneven connectivity

The RFID readers might be separated from the local control device that contains an MQTT client, or each reader might embed an MQTT client. Typically, geographical or communications factors indicate the choice of topology.

Security

Privacy and authenticity are security concerns in the attachment of RFID tags. RFID tags are unobtrusive and can be covertly monitored, spoofed, or tampered with.

Solution of RFID security issues increases the opportunity for deployment of new RFID solutions. Although the security exposure is in the RFID tag, and the local reader, using central information processing suggests approaches for countering different threats. For example, tag tampering might be detected by dynamically correlating stock levels against deliveries and dispatches.

Connectivity

RFID applications typically involved both batched store and forward of information gathered from RFID readers and immediate queries. In the distribution warehouse use case, the RFID reader is connected all the time. When a tag is read, it is published along with information about the reader. The warehousing application publishes the response back to the reader.

In the warehousing application the network is typically reliable, and the immediate requests might use "fire and forget" messages for low latency performance. The batched store and forward data might use "exactly once" messaging to minimize administration costs associated with losing data.

Scalability

If the RFID application requires immediate responses, in the order of a second or two, then the RFID readers must stay connected.

Telemetry use cases: Environment sensing



Environment sensing uses telemetry to collect information about river water levels and quality, atmospheric pollutants, and other environmental data.

Sensors are frequently located in remote places, without access to wired communication. Wireless bandwidth is expensive and reliability can be low. Typically, a number of environment sensors in a small geographical area are connected to a local monitoring device in a safe location. The local connections might be wired or wireless.

Invisibility

The sensor devices are likely to be less accessible, lower powered, and deployed in greater numbers, than the central monitoring device. The sensors are sometimes "dumb", and the local monitoring device includes adapters to transform and store sensor data. The monitoring device is likely to incorporate a general-purpose computer that supports Java Platform, Standard Edition (Java SE) or Java Platform, Micro Edition (Java ME). Invisibility is unlikely to be a major requirement when configuring the MQTT client.

Uneven connectivity

The capabilities of sensors, and cost and bandwidth of remote connection, typically results in a local monitoring hub connected to a central server.

Security

Unless the solution is being used in a military or defensive use case, security is not a major requirement.

Connectivity

Many uses do not require continuous monitoring or immediate availability of data. Exception data, such as a flood level alert, does need to be forwarded immediately. Sensor data is aggregated at the local monitor to reduce connection and communication costs, and then transferred using scheduled connections. Exception data is forwarded as soon as it is detected at the monitor.

Scalability

Sensors are concentrated around local hubs, and sensor data is aggregated into packets that are transmitted according to a schedule. Both these factors reduce the load on the central server that would be imposed by using directly connected sensors.

Telemetry use cases: Mobile applications

AIX

Linux

Windows

Mobile applications are applications that run on wireless devices. The devices are either generic application platforms or custom devices.

General platforms include handheld devices such as phones and personal data assistants, and portable devices such as notebook computers. Custom devices use special purpose hardware tailored to specific applications. A device to record "signed-for" parcel delivery is an example of a custom mobile device. Applications on custom mobile devices are often built on a generic software platform.

Invisibility

The deployment of custom mobile applications is managed, and can include configuration of the MQTT client application. Invisibility is unlikely to be a major requirement when configuring the MQTT client.

Uneven connectivity

Unlike the local hub topology of the preceding use cases, mobile clients connect remotely. The client application layer connects directly to an application at the central hub.

Security

With little physical security, the mobile device, and the mobile user must be authenticated. TLS is used to confirm the identity of the device, and JAAS to authenticate the user.

Connectivity

If the mobile application depends on wireless coverage, it must be able to operate offline, and to deal efficiently with an interrupted connection. In this environment, the goal is to stay connected, but the application must be able to store and forward messages. Often the messages are orders, or delivery confirmations, and have important business value. They need to be stored and forwarded reliably.

Scalability

Scalability is not a major issue. The numbers of application clients are likely to not to exceed the thousands, or tens of thousands, in custom mobile application use cases.

Connecting telemetry devices to a queue manager

AIX

Linux

Windows

Telemetry devices connect to a queue manager using an MQTT v3 client. The MQTT v3 client uses TCP/IP to connect to a TCP/IP listener called the telemetry (MQXR) service.

When you connect a telemetry device to a queue manager, the MQTT client initiates a TCP/IP connection using the `MqttClient.connect` method. Like IBM MQ clients, an MQTT client must be connected to the queue manager to send and receive messages. The connection is made at the server using a TCP/IP listener, installed with MQ Telemetry, called the telemetry (MQXR) service. Each queue manager runs a maximum of one telemetry (MQXR) service.

The telemetry (MQXR) service uses the remote socket address set by each client in the `MqttClient.connect` method to allocate the connection to a telemetry channel. A socket address is the combination of TCP/IP host name and port number. Multiple clients that use the same remote socket address are connected to the same telemetry channel by the telemetry (MQXR) service.

If there are multiple queue managers on a server, split the telemetry channels between the queue managers. Allocate the remote socket addresses between the queue managers. Define each telemetry channel with a unique remote socket address. Two telemetry channels must not use the same socket address.

If the same remote socket address is configured for telemetry channels on multiple queue managers, the first telemetry channel to connect, wins. Subsequent channels connecting on the same address fail.

If there are multiple network adapters on the server, split the remote socket addresses between telemetry channels. The allocation of socket addresses is entirely arbitrary, as long as any specific socket address is configured on only one telemetry channel.

Configure IBM MQ to connect MQTT clients using the wizards provided in the MQ Telemetry supplement for IBM MQ Explorer. Alternatively, follow the instructions in [Configuring a queue manager for telemetry on Linux and AIX](#) and [Configuring a queue manager for telemetry on Windows](#) to configure telemetry manually.

Related information:

MQXR properties

Telemetry connection protocols



MQ Telemetry supports TCP/IP IPv4 and IPv6, and TLS.

Telemetry (MQXR) service



The telemetry (MQXR) service is a TCP/IP listener, that is managed as an IBM MQ service. Create the service using an IBM MQ Explorer wizard, or with a `runmqsc` command.

The MQ Telemetry (MQXR) service is called `SYSTEM.MQXR.SERVICE` .

The Telemetry sample configuration wizard, provided in the MQ Telemetry function for IBM MQ Explorer, creates the telemetry service and a sample telemetry channel; see [Verifying the installation of MQ Telemetry by using IBM MQ Explorer](#).

Create the sample configuration from the command line; see [Verifying the installation of MQ Telemetry using the command line](#).

The telemetry (MQXR) service starts and stops automatically with the queue manager. Control the service using the services folder in IBM MQ Explorer. To see the service, you must click the icon to stop IBM MQ Explorer filtering out SYSTEM objects from the display.



`installMQXRService_unix.mqsc` shows an example of how to create the service manually on AIX and Linux.



`installMQXRService_win.mqsc` shows how to create the service manually in Windows.

Telemetry channels



Create telemetry channels to create connections with different properties, such as Java Authentication and Authorization Service (JAAS) or TLS authentication, or to manage groups of clients.

Create Telemetry channels using the **New Telemetry Channel** wizard, supplied in the MQ Telemetryfunction for IBM MQ Explorer. Configure a channel, using the wizard, to accept connections from MQTT clients on a particular TCP/IP port. Since Version 7.1, you can configure MQ Telemetry using the command line program, **runmqsc**.

Create multiple telemetry channels, on different ports, to make large numbers of client connections easier to manage, by splitting the clients into groups. Each telemetry channel has a different name.

You can configure telemetry channels with different security attributes to create different types of connection. Create multiple channels to accept client connections on different TCP/IP addresses. Use TLS to encrypt messages and authenticate the telemetry channel and client; see TLS configuration of MQTT clients and telemetry channels. Specify the user ID to simplify authorizing access to IBM MQ objects. Specify a JAAS configuration to authenticate the MQTT user with JAAS; see MQTT client identification, authorization, and authentication.

MQTT protocol



The IBM MQ Telemetry Transport (MQTT) v3 protocol is designed for exchanging messages between small devices on low bandwidth, or expensive connections, and to send messages reliably. It uses TCP/IP.

The MQTT protocol is published; see IBM MQ Telemetry Transport format and protocol. Version 3 of the protocol uses publish/subscribe, and supports three qualities of service: "fire and forget", "at least once", and "exactly once".

The small size of the protocol headers, and the byte array message payload, keeps messages small. The headers comprise a 2 byte fixed header, and up to 12 bytes of additional variable headers. The protocol uses 12 byte variable headers to subscribe and connect, and only 2 byte variable headers for most publications.

With three qualities of service, you can trade off between low-latency and reliability; see Qualities of service provided by an MQTT client. "Fire and forget" uses no persistent device storage, and only one transmission to send or receive a publication. "At least once", and "exactly once" require persistent storage on the device to maintain the protocol state and save a message until it is acknowledged.

MQTT clients



An MQTT client app is responsible for collecting information from the telemetry device, connecting to the server, and publishing the information to the server. It can also subscribe to topics, receive publications, and control the telemetry device.

Unlike IBM MQ client applications, MQTT client apps are not IBM MQ applications. They do not specify a queue manager to connect to. They are not limited to using specific IBM MQ programming interfaces. Instead, MQTT clients implement the MQTT Version 3 protocol. You can write your own client library to interface to the MQTT protocol in the programming language, and on the platform, of your choice. See IBM MQ Telemetry Transport format and protocol.

To simplify writing MQTT client apps, use the C, Java, and JavaScript client libraries that encapsulate the MQTT protocol for a number of platforms. For links to client API documentation for the MQTT client libraries, see MQTT client programming reference. If you incorporate these libraries in your MQTT apps, a fully functional MQTT client can be as short as 15 lines of code.

To use the telemetry MQTT clients, download the IBM Messaging Telemetry Clients SupportPac, then install the clients in a directory of your own choosing. The sample applications and client libraries are in client-specific directories under *CLIENTPACKDIR/SDK/clients*, where *CLIENTPACKDIR* is the directory in which you decompressed the client pack. Note that the example scripts included in IBM MQ assume that the clients are in the directories specified in Location of telemetry logs, error logs, and configuration files.

The MQTT client app is always responsible for initiating a connection with a telemetry channel. After it is connected, either the MQTT client app or an IBM MQ application can start an exchange of messages.

MQTT client apps and IBM MQ applications publish and subscribe to the same set of topics. An IBM MQ application can also send a message directly to an MQTT client app without the client app first creating a subscription. See Configure distributed queuing to send messages to MQTT clients.

MQTT client apps are connected to IBM MQ using a telemetry channel. The telemetry channel acts as a bridge between the different types of message used by MQTT and IBM MQ. It creates publications and subscriptions in the queue manager on behalf of the MQTT client app. The telemetry channel sends publications that match the subscriptions of an MQTT client app from the queue manager to the MQTT client app.

Sending a message to an MQTT client



IBM MQ applications can send MQTT v3 clients messages by publishing to subscriptions created by clients, or by sending messages directly. MQTT clients can send messages to one another by publishing to topics subscribed to by other clients.

An MQTT client subscribes to a publication, which it receives from IBM MQ

Do the task, “Publishing a message to the MQTT client utility from IBM MQ Explorer” on page 268 to send a publication from IBM MQ to an MQTT client.

The standard way for an MQTT v3 client to receive messages is for it to create a subscription to a topic, or set of topics. In the example code snippet, Figure 45 on page 266, the MQTT client subscribes using the topic string "MQTT Examples". An IBM MQ C application, Figure 46 on page 267, publishes to the topic using the topic string "MQTT Examples". In the code snippet Figure 47 on page 267, the MQTT client receives the publication in the callback method, `messageArrived`.

For further information about how to configure IBM MQ to send publications in response to subscriptions from MQTT clients, see [Publishing a message in response to an MQTT client subscription](#).

An IBM MQ application sends a message directly to an MQTT client

Do the task, “[Sending a message to an MQTT client using IBM MQ Explorer](#)” on page 272 to send a message directly from IBM MQ to an MQTT client.

A message sent in this way to an MQTT client is called an unsolicited message. MQTT v3 clients receive unsolicited messages as publications with a topic name set. The telemetry (MQXR) service sets the topic name to the remote queue name.

For further information about how to configure IBM MQ to send messages directly to MQTT clients, see [Sending a message to a client directly](#).

An MQTT client publishes a message

An MQTT v3 client can publish a message that is received by another MQTT v3 client, but it cannot send an unsolicited message. The code snippet [Figure 48](#) on page 267 shows how an MQTT v3 client, written in Java, publishes a message.

The typical pattern for sending a message to one specific MQTT v3 client, is for each client to create a subscription to its own `ClientIdentifier`. Do the task “[Publishing a message to a specific MQTT v3 client](#)” on page 273 to publish a message from one MQTT client to another MQTT client using `ClientIdentifier` as a topic string.

Example code snippets

The code snippet in [Figure 45](#) shows how an MQTT client written in Java creates a subscription. It also needs a callback method, `messageArrived` to receive publications for the subscription.

```
String    clientId = String.format("%-23.23s",
                                System.getProperty("user.name") + "_" +
                                (UUID.randomUUID().toString()).trim()).replace('-', '_');
MqttClient client = new MqttClient("localhost", clientId);
String topicString = "MQTT Examples";
int       QoS = 1;
client.subscribe(topicString, QoS);
```

Figure 45. MQTT v3 client subscriber

The code snippet in [Figure 46](#) on page 267 shows how an IBM MQ application written in C sends a publication. The code snippet is extracted from the task, [Create a publisher to a variable topic](#)

```

/* Define and set variables to defaults */
/* Omitted lines declaring variables */
char * topicName = ""
char * topicString = "MQTT Examples"
char * publication = "Hello world!";
do {
    MQCONN(qMgrName, &Hconn, &CompCode, &Reason);
    if (CompCode != MQCC_OK) break;
    td.ObjectType = MQOT_TOPIC; /* Object is a topic */
    td.Version = MQOD_VERSION_4; /* Descriptor needs to be V4 */
    strncpy(td.ObjectName, topicName, MQ_TOPIC_NAME_LENGTH);
    td.ObjectString.VSPtr = topicString;
    td.ObjectString.VSLength = (MQLONG)strlen(topicString);
    MQOPEN(Hconn, &td, MQOO_OUTPUT | MQOO_FAIL_IF QUIESCING, &Hobj, &CompCode, &Reason);
    if (CompCode != MQCC_OK) break;
    pmo.Options = MQPMO_FAIL_IF QUIESCING | MQPMO_RETAIN;
    MQPUT(Hconn, Hobj, &md, &pmo, (MQLONG)strlen(publication)+1, publication, &CompCode, &Reason);
    if (CompCode != MQCC_OK) break;
    MQCLOSE(Hconn, &Hobj, MQCO_NONE, &CompCode, &Reason);
    if (CompCode != MQCC_OK) break;
    MQDISC(&Hconn, &CompCode, &Reason);
} while (0);

```

Figure 46. IBM MQ publisher

When the publication arrives, the MQTT client calls the `messageArrived` method of the MQTT application client `MqttCallback` class.

```

public class Callback implements MqttCallback {
    public void messageArrived(MqttTopic topic, MqttMessage message) {
        try {
            System.out.println("Message arrived: \"" + message.toString()
                + "\" on topic \"" + topic.toString() + "\"");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    // ... Other callback methods
}

```

Figure 47. messageArrived method

Figure 48 shows an MQTT v3 publishing a message to the subscription created in Figure 45 on page 266.

```

String address = "localhost";
String clientId = String.format("%-23.23s",
    System.getProperty("user.name") + "_" +
    (UUID.randomUUID().toString()).trim()).replace('-', '_');
MqttClient client = new MqttClient(address, clientId);
String topicString = "MQTT Examples";
MqttTopic topic = client.getTopic(topicString);
String publication = "Hello world!";
MqttMessage message = new MqttMessage(publication.getBytes());
MqttDeliveryToken token = topic.publish(message);

```

Figure 48. MQTT v3 client publisher

Publishing a message to the MQTT client utility from IBM MQ Explorer

AIX

Linux

Windows

Follow the steps in this task to publish a message using IBM MQ Explorer, and subscribe to it with the MQTT client utility. An additional task shows you how to configure a queue manager alias rather than setting the default transmission queue to `SYSTEM.MQTT.TRANSMIT.QUEUE`.

Before you begin

The task assumes that you are familiar with IBM MQ and the IBM MQ Explorer, and that IBM MQ and MQ Telemetry feature are installed.

The user creating the queue manager resources for this task must have sufficient authority to do so. For demonstration purposes, the IBM MQ Explorer user ID is assumed to be member of the `mqm` group.

About this task

In the task, you create a topic in IBM MQ and subscribe to the topic using the MQTT client utility. When you publish to the topic using IBM MQ Explorer, the MQTT client receives the publication.

Procedure

Do one of the following tasks:

- You have installed MQ Telemetry, but you have not started it yet. Do the task: “Start task with no telemetry (MQXR) service yet defined” on page 269.
- You have run IBM MQ telemetry before, but want to use a new queue manager to do the demonstration. Do the task: “Start task with no telemetry (MQXR) service yet defined” on page 269.
- You want to do the task using an existing queue manager that has no telemetry resources defined. You do not want to run the **Define sample configuration** wizard.
 1. Do one of the following tasks to set up telemetry:
 - Configuring a queue manager for telemetry on Linux and AIX
 - Configuring a queue manager for telemetry on Windows
 2. Do the task: “Start task with a running telemetry (MQXR) service” on page 269
- If you want to do the task using an existing queue manager that already has telemetry resources defined, do the task: “Start task with a running telemetry (MQXR) service” on page 269.

What to do next

Do “Sending a message to an MQTT client using IBM MQ Explorer” on page 272 to send a message directly to the client utility.

Start task with no telemetry (MQXR) service yet defined:

Create a queue manager and run the **Define sample configuration** to define sample telemetry resources for the queue manager. Publish a message using IBM MQ Explorer, and subscribe to it with the MQTT client utility.

About this task

When you set up sample telemetry resources using the **Define sample configuration**, the wizard sets the guest user ID permissions. Carefully consider if you want the guest user ID to be authorized in this way. guest on Windows, and nobody on Linux, are given permission to publish and subscribe to the root of the topic tree, and to put messages onto SYSTEM.MQTT.TRANSMIT.QUEUE.

The wizard also sets the default transmission queue to SYSTEM.MQTT.TRANSMIT.QUEUE, which might interfere with applications running on an existing queue manager. It is possible, but laborious, to configure telemetry and not use the default transmission queue; do the follow on task: "Using a queue manager alias" on page 271. In this task, you create a queue manager to avoid the possibility of interfering with any existing default transmission queue.

Procedure

1. Using IBM MQ Explorer, create and start a new queue manager.
 - a. Right-click Queue Managers folder > **New** > **Queue manager** Type a queue manager name > **Finish**.
Make up a queue manager name; for example, MQTTQMGR.
2. Create and start the telemetry (MQXR) service and create a sample telemetry channel.
 - a. Open the Queue Managers\QmgrName\Telemetry folder.
 - b. Click **Define sample configuration...** > **Finish**
Leave the **Launch MQTT Client Utility** check box checked.
3. Create a subscription for MQTT Example using the MQTT client utility.
 - a. Click **Connect**.
The **Client history** records a Connected event.
 - b. Type MQTT Example into the **Subscription\Topic** field > **Subscribe**.
The **Client history** records a Subscribed event.
4. Create MQTTExampleTopic in IBM MQ.
 - a. Right-click the Queue Managers\QmgrName\Topics folder in the **MQ Explorer** > **New** > **Topic**.
 - b. Type MQTTExampleTopic as the **Name** > **Next**.
 - c. Type MQTT Example as the **Topic string** > **Finish**.
 - d. Click **OK** to close the acknowledgment window.
5. Publish Hello World! to the topic MQTT Example using IBM MQ Explorer.
 - a. Click the Queue Managers\QmgrName\Topics folder in the IBM MQ Explorer.
 - b. Right-click MQTTExampleTopic > **Test publication...**
 - c. Type Hello World! into the **Message data** field > **Publish message** > Switch to the MQTT Client Utility window.
The **Client history** records a Received event.

Start task with a running telemetry (MQXR) service:

Create a telemetry channel and a topic. Authorize the user to use the topic and the telemetry transmit queue. Publish a message using IBM MQ Explorer, and subscribe to it with the MQTT client utility.

Before you begin

In this version of the task, a queue manager, *QmgrName*, is defined and running. A telemetry (MQXR) service is defined and running. The telemetry (MQXR) service might have been created manually, or by running the **Define sample configuration** wizard.

About this task

In this task you configure an existing queue manager to send a publication to the MQTT client utility.

Step 1 of the task sets the default transmission queue to `SYSTEM.MQTT.TRANSMIT.QUEUE`, which might interfere with applications running on an existing queue manager. It is possible, but laborious, to configure telemetry and not use the default transmission queue; do the follow on task: "Using a queue manager alias" on page 271.

Procedure

1. Set `SYSTEM.MQTT.TRANSMIT.QUEUE` as the default transmit queue.
 - a. Right-click the Queue Managers*QmgrName* folder > **Properties...**
 - b. Click **Communication** in the navigator.
 - c. Click **Select...** > Select `SYSTEM.MQTT.TRANSMIT.QUEUE` > **OK** > **OK**.
2. Create a telemetry channel `MQTTExampleChannel` to connect the MQTT client utility to IBM MQ, and start the MQTT client utility.
 - a. Right-click the Queue Managers*QmgrName* \Telemetry\Channels folder in the **MQ Explorer**> **New** > **Telemetry channel...**
 - b. Type `MQTTExampleChannel` in the **Channel name** field > **Next** > **Next**.
 - c. Change the **Fixed user ID** on the client authorization panel to the user ID that is going to publish and subscribe to `MQTTExample` > **Next**.
 - d. Leave **Launch Client Utility** checked> **Finish**.
3. Create a subscription for MQTT Example using the MQTT client utility.
 - a. Click **Connect**.
The **Client history** records a Connected event.
 - b. Type MQTT Example into the **Subscription\Topic** field> **Subscribe**.
The **Client history** records a Subscribed event.
4. Create `MQTTExampleTopic` in IBM MQ.
 - a. Right-click the Queue Managers*QmgrName*\Topics folder in the **MQ Explorer**> **New** > **Topic**.
 - b. Type `MQTTExampleTopic` as the **Name** > **Next**.
 - c. Type MQTT Example as the **Topic string** > **Finish**.
 - d. Click **OK** to close the acknowledgment window.
5. If you want a user, not in the `mqm` group, to publish and subscribe to the `MQTTExample` topic, do the following:
 - a. Authorize the user to publish and subscribe to the topic `MQTTExampleTopic`:
`setmqaut -m qMgrName -t topic -n MQTTExampleTopic -p User ID -all +pub +sub`
 - b. Authorize the user to put a message onto the `SYSTEM.MQTT.TRANSMIT.QUEUE`:
`setmqaut -m qMgrName -t q -n SYSTEM.MQTT.TRANSMIT.QUEUE -p User ID -all +put`
6. Publish Hello World! to the topic MQTT Example using IBM MQ Explorer.
 - a. Click the Queue Managers*QmgrName*\Topics folder in the IBM MQ Explorer.
 - b. Right-click `MQTTExampleTopic` > **Test publication...**
 - c. Type Hello World! into the **Message data** field > **Publish message** > Switch to the MQTT Client Utility window.

The **Client history** records a Received event.

Using a queue manager alias:

Publish a message to the MQTT client utility using IBM MQ Explorer without setting the default transmission queue to `SYSTEM.MQTT.TRANSMIT.QUEUE`.

The task is a continuation of the previous task, and uses a queue manager alias to avoid setting the default transmission queue to `SYSTEM.MQTT.TRANSMIT.QUEUE`.

Before you begin

Complete either the task, “Start task with no telemetry (MQXR) service yet defined” on page 269 or the task, “Start task with a running telemetry (MQXR) service” on page 269.

About this task

When an MQTT client creates a subscription, IBM MQ sends its response using `ClientIdentifier`, as the remote queue manager name. In this task, it uses the `ClientIdentifier`, `MyClient`.

If there is no transmission queue or queue manager alias called `MyClient`, the response is placed on the default transmission queue. By setting default transmission queue to `SYSTEM.MQTT.TRANSMIT.QUEUE`, the MQTT client gets the response.

You can avoid setting the default transmission queue to `SYSTEM.MQTT.TRANSMIT.QUEUE` by using queue manager aliases. You must set up a queue manager alias for every `ClientIdentifier`. Typically, there are too many clients to make it practical to use queue manager aliases. Often `ClientIdentifier` is unpredictable, making it impossible to configure telemetry this way.

Nonetheless, in some circumstances you might have to configure the default transmission queue to something other than `SYSTEM.MQTT.TRANSMIT.QUEUE`. The steps in Procedure configure a queue manager alias instead of setting the default transmission queue to `SYSTEM.MQTT.TRANSMIT.QUEUE`.

Procedure

1. Remove `SYSTEM.MQTT.TRANSMIT.QUEUE` as the default transmit queue.
 - a. Right-click the Queue Managers*QmgrName* folder > **Properties...**
 - b. Click **Communication** in the navigator.
 - c. Remove `SYSTEM.MQTT.TRANSMIT.QUEUE` from the **Default transmission queue** field > **OK**.
2. Check that you can no longer create a subscription with the MQTT client utility:
 - a. Click **Connect**.
The **Client history** records a Connected event.
 - b. Type MQTT Example into the **Subscription\Topic** field> **Subscribe**.
The **Client history** records a Subscribe failed and a Connection lost event.
3. Create a queue manager alias for the `ClientIdentifier`, `MyClient`.
 - a. Right-click the Queue Managers*QmgrName*\Queues folder> **New** > **Remote queue definition**.
 - b. Name the definition, `MyClient` > **Next**.
 - c. Type `MyClient` in the **Remote queue manager** field.
 - d. Type `SYSTEM.MQTT.TRANSMIT.QUEUE` in the **Transmission queue** field > **Finish**.
4. Connect the MQTT client utility again.
 - a. Check the **Client identifier** is set to `MyClient`.
 - b. **Connect**
The **Client history** records a Connected event.

5. Create a subscription for MQTT Example using the MQTT client utility.
 - a. Click **Connect**.
The **Client history** records a Connected event.
 - b. Type MQTT Example into the **Subscription\Topic** field > **Subscribe**.
The **Client history** records a Subscribed event.
6. Publish Hello World! to the topic MQTT Example using IBM MQ Explorer.
 - a. Click the Queue Managers*QmgrName*\Topics folder in the IBM MQ Explorer.
 - b. Right-click MQTTExampleTopic > **Test publication...**
 - c. Type Hello World! into the **Message data** field > **Publish message** > Switch to the MQTT Client Utility window.
The **Client history** records a Received event.

Sending a message to an MQTT client using IBM MQ Explorer



Send a message to the MQTT client utility by putting a message onto an IBM MQ queue using IBM MQ Explorer. The task shows you how to configure a remote queue definition to send a message directly to an MQTT client.

Before you begin

Do the task, “Publishing a message to the MQTT client utility from IBM MQ Explorer” on page 268. Leave the MQTT client utility connected.

About this task

The task demonstrates sending a message to an MQTT client using queue rather than publishing to a topic. You do not create a subscription in the client. Step 2 of the task demonstrates that the previous subscription has been deleted.

Procedure

1. Discard any existing subscriptions by disconnecting and reconnecting the MQTT client utility.
The subscription is discarded because, unless you change the defaults, the MQTT client utility connects with a clean session; see Clean sessions.
To make it easier to do the task, type your own `ClientIdentifier`, rather than use the generated `ClientIdentifier` created by the MQTT client utility.
 - a. Click **Disconnect** to disconnect the MQTT client utility from the telemetry channel.
The **Client History** records a Disconnected event
 - b. Change the **Client Identifier** to MyClient.
 - c. Click **Connect**.
The **Client History** records a Connected event
2. Check that the MQTT client utility no longer receives publication for the MQTTExampleTopic.
 - a. Click the Queue Managers*QmgrName*\Topics folder in the IBM MQ Explorer.
 - b. Right-click MQTTExampleTopic > **Test publication...**
 - c. Type Hello World! into the **Message data** field > **Publish message** > Switch to the MQTT Client Utility window.
No event is recorded in the **Client history**.
3. Create a remote queue definition for the client.

Set the `ClientIdentifier`, `MyClient`, as the remote queue manager name in the remote queue definition. Use any name you like as the remote queue name. The remote queue name is passed to an MQTT client as the topic name.

- a. Right-click the Queue Managers*QmgrName*\Queues folder > **New** > **Remote queue definition**.
 - b. Name the definition, `MyClientRemoteQueue` > **Next**.
 - c. Type `MQTTExampleQueue` in the **Remote queue** field.
 - d. Type `MyClient` in the **Remote queue manager** field.
 - e. Type `SYSTEM.MQTT.TRANSMIT.QUEUE` in the **Transmission queue** field > **Finish**.
4. Put a test message onto `MyClientRemoteQueue`.
- a. Right-click `MyClientRemoteQueue` > **Put test message...**
 - b. Type `Hello queue!` into the Message data field > **Put message** > **Close**
- The **Client history** records a Received event.
5. Remove `SYSTEM.MQTT.TRANSMIT.QUEUE` as the default transmit queue.
- a. Right-click the Queue Managers*QmgrName* folder > **Properties...**
 - b. Click **Communication** in the navigator.
 - c. Remove `SYSTEM.MQTT.TRANSMIT.QUEUE` from the **Default transmission queue** field > **OK**.
6. Redo step 4.

`MyClientRemoteQueue` is a remote queue definition that explicitly names the transmission queue. You do not need a to define default transmission queue to send a message to `MyClient`.

What to do next

With the default transmission queue no longer set to `SYSTEM.MQTT.TRANSMIT.QUEUE`, the MQTT Client Utility is unable to create a new subscription unless a queue manager alias is defined for the `ClientIdentifier`, `MyClient`. Restore the default transmission queue to `SYSTEM.MQTT.TRANSMIT.QUEUE`.

Publishing a message to a specific MQTT v3 client



Publish a message from one MQTT v3 client to another, using `ClientIdentifier` as the topic name and IBM MQ as the publish/subscribe broker.

Before you begin

Do the task, “Publishing a message to the MQTT client utility from IBM MQ Explorer” on page 268. Leave the MQTT client utility connected.

About this task

The task demonstrates two things:

1. Subscribing to a topic in one MQTT client, and receiving a publication from another MQTT client.
2. Setting up "point-to-point" subscriptions by using `ClientIdentifier` as the topic string.

Procedure

1. Discard any existing subscriptions by disconnecting and reconnecting the MQTT client utility. The subscription is discarded because, unless you change the defaults, the MQTT client utility connects with a clean session; see Clean sessions.

To make it easier to do the task, type your own `ClientIdentifier`, rather than use the generated `ClientIdentifier` created by the MQTT client utility.

- a. Click **Disconnect** to disconnect the MQTT client utility from the telemetry channel.

The **Client History** records a Disconnected event

- b. Change the **Client Identifier** to MyClient.
- c. Click **Connect**.

The **Client History** records a Connected event

2. Create a subscription to the topic, MyClient
MyClient is the ClientIdentifier of this client.
 - a. Type MyClient into the **Subscription\Topic** field> **Subscribe**.

The **Client history** records a Subscribed event.

3. Start another MQTT client utility.
 - a. Open the Queue Managers*QmgrName*\Telemetry\channels folder.
 - b. Right-click the **PlainText** channel> **Run MQTT Client Utility...**
 - c. Click **Connect**.

The **Client History** records a Connected event

4. Publish Hello MyClient! to the topic MyClient.
 - a. Copy the subscription topic, MyClient, from the MQTT client utility running with the ClientIdentifier, MyClient.
 - b. Paste MyClient into the **Publication\Topic** field of each of the MQTT client utility instances.
 - c. Type Hello MyClient! into the **Publication\message** field.
 - d. Click **Publish** in both instances.

Results

The **Client history** in the MQTT client utility with the ClientIdentifier, MyClient, records two **Received** events and one **Published** event. The other MQTT client utility instance records one **Published** event.

If you see only one **Received** event, check the following possible causes:

1. Is the default transmission queue for the queue manager set to SYSTEM.MQTT.TRANSMIT.QUEUE ?
2. Have you created queue manager aliases or remote queue definitions referencing MyClient in doing the other exercises? In case you have a configuration problem, delete any resources that reference MyClient, such as a queue manager aliases or transmission queues. Disconnect the client utilities, stop, and restart the telemetry (MQXR) service.

Send a message to an IBM MQ application from an MQTT client



An IBM MQ application can receive a message from an MQTT v3 client by subscribing to a topic. The MQTT client connects to IBM MQ using a telemetry channel, and sends a message to the IBM MQ application by publishing to the same topic.

Do the task, “Publishing a message to IBM MQ from an MQTT client” on page 275, to learn how to send a publication from an MQTT client to a subscription defined in IBM MQ.

If the topic is clustered, or distributed using a publish/subscribe hierarchy, the subscription can be on a different queue manager to the queue manager that the MQTT client is connected to.

Publishing a message to IBM MQ from an MQTT client



Create a subscription to a topic using IBM MQ Explorer and publish to the topic using MQTT client utility.

Before you begin

Do the task, “Publishing a message to the MQTT client utility from IBM MQ Explorer” on page 268. Leave the MQTT client utility connected.

About this task

The task demonstrates publishing a message with an MQTT client and receiving the publication using an unmanaged durable subscription created using IBM MQ Explorer.

Procedure

1. Create a durable subscription to the topic string MQTT Example. Do the following steps to create the queue, and subscription using IBM MQ Explorer.
 - a. Right-click the Queue Managers*QmgrName*\Queues folder in the IBM MQ Explorer> **New > Local queue...**
 - b. Type MQTTExampleQueue as the queue name > **Finish**.
 - c. Right-click the Queue Managers*QmgrName*\Subscriptions folder in the IBM MQ Explorer> **New > Subscription...**
 - d. Type MQTTExampleSubscription as the queue name > **Next**.
 - e. Click **Select...** > MQTTExampleTopic > **OK**.

You have already created the topic, MQTTExampleTopic in step 4 on page 269 of “Publishing a message to the MQTT client utility from IBM MQ Explorer” on page 268.
 - f. Type MQTTExampleQueue as the destination name > **Finish**.
2. As an optional step, set the queue up for use by a different user, without mqm authority.

If you are setting up the configuration for users with less authority than mqm, you must give put and get authority to MQTTExampleQueue. Access to the topic and to the transmission queue was configured in “Publishing a message to the MQTT client utility from IBM MQ Explorer” on page 268.

 - a. Authorize a user to put and get to the queue MQTTExampleQueue:

```
setmqaut -m qMgrName -t queue -n MQTTExampleQueue -p User ID -all +put +get
```
3. Publish Hello IBM MQ! to the topic MQTT Example using the MQTT client utility.

If you have not left the MQTT client utility connected, right-click the **PlainText** channel> **Run MQTT Client Utility...** > **Connect**.

 - a. Type MQTT Example into the **Publication\Topic** field.
 - b. Type Hello IBM MQ! into the **Publication\Message** field> **Publish**.
4. Open the Queue Managers*QmgrName*\Queues folder and find MQTTExampleQueue.

The **Current queue depth** field is 1
5. Right-click MQTTExampleQueue > **Browse messages...** and examine the publication.

MQTT publish/subscribe applications



Use topic-based publish/subscribe to write MQTT applications.

When the MQTT client is connected, publications flow in either direction between the client and server. The publications are sent from the client when information is published at the client. Publications are received at the client when a message is published to a topic that matches a subscription created by the client.

The IBM MQ publish/subscribe broker manages the topics and subscriptions created by MQTT clients. The topics created by MQTT clients share the same topic space as topics created by IBM MQ applications.

Publications that match the topic string in an MQTT client subscription are placed on `SYSTEM.MQTT.TRANSMIT.QUEUE` with the remote queue manager name set to the `ClientIdentifier` of the client. The telemetry (MQXR) service forwards the publications to the client that created the subscription. It uses `ClientIdentifier`, which has been set as the remote queue manager name to identify the client.

Typically, `SYSTEM.MQTT.TRANSMIT.QUEUE` must be defined as the default transmission queue. It is possible, but onerous, to configure MQTT not to use the default transmission queue; see *Configure distributed queuing to send messages to MQTT clients*.

An MQTT client can create a persistent session; see “MQTT stateless and stateful sessions” on page 279. Subscriptions created in a persistent session are durable. Publications that arrive for a client with a persistent session are stored in `SYSTEM.MQTT.TRANSMIT.QUEUE`, and forwarded to the client when it reconnects.

An MQTT client can also publish and subscribe to retained publications; see *Retained publications and MQTT clients*. A subscriber to a retained publication topic receives the latest publication to the topic. The subscriber receives the retained publication when it creates a subscription, or when it reconnects to its earlier session.

Telemetry applications



Write telemetry applications using IBM MQ or IBM Integration Bus message flows.

Use JMS, MQI, or other IBM MQ programming interfaces to program telemetry applications in IBM MQ.

The telemetry (MQXR) service converts between MQTT v3 messages and IBM MQ messages. It creates subscriptions and publications on behalf of MQTT clients, and forwards publications to MQTT clients. A publication is the payload of an MQTT v3 message. The payload comprises message headers and a byte array in `jms-bytes` format. The telemetry server maps the headers between an MQTT v3 message and an IBM MQ message; see “Integration of MQ Telemetry with queue managers” on page 277.

Use the `Publication`, `MQInput`, and `JMSInput` nodes to send and receive publications between IBM Integration Bus and MQTT clients.

Using message flows you can integrate telemetry with web sites using HTTP, and with other applications using IBM MQ and WebSphere Adapters.

Integration of MQ Telemetry with queue managers



The MQTT client is integrated with IBM MQ as a publish/subscribe application. It can either publish or subscribe to topics in IBM MQ, creating new topics, or using existing topics. It receives publications from IBM MQ as a result of MQTT clients, including itself, or other IBM MQ applications publishing to the topics of its subscriptions. Rules are applied to decide the attributes of a publication.

Many of the attributes associated with topics, publications, subscriptions, and messages that are provided by IBM MQ, are not supported. “MQTT client to IBM MQ publish/subscribe broker” and “IBM MQ to an MQTT client” on page 278 describe how attributes of publications are set. The settings depend on whether the publication is going to or from the IBM MQ publish/subscribe broker.

In IBM MQ publish/subscribe topics are associated with administrative topic objects. The topics created by MQTT clients are no different. When an MQTT client creates a topic string for a publication the IBM MQ publish/subscribe broker associates it with an administrative topic object. The broker maps the topic string in the publication to the nearest administrative topic object parent. The mapping is the same as for IBM MQ applications. If there is no user created topic, the publication topic is mapped to SYSTEM.BASE.TOPIC. The attributes that are applied to the publication are derived from the topic object.

When an IBM MQ application, or an administrator creates a subscription, the subscription is named. List subscriptions using IBM MQ Explorer, or by using `runmqsc` or PCF commands. All MQTT client subscriptions are named. They are given a name of the form: *ClientIdentifier:Topic name*

MQTT client to IBM MQ publish/subscribe broker

An MQTT client has sent a publication to IBM MQ. The telemetry (MQXR) service converts the publication to an IBM MQ message. The IBM MQ message contains three parts:

1. MQMD
2. RFH2
3. Message

MQMD properties are set to their default values, except where noted in Table 14.

Table 14. MQMD

MQMD field	Type	Value
Format	MQCHAR8	MQFMT_RF_HEADER_2
UserIdentifier	MQCHAR12	Set to one of: MqttClient.ClientIdentifier MqttConnectOptions.UserName A user ID set by the IBM MQ administrator for the telemetry channel.
Priority	MLONG	MQPRI_PRIORITY_AS_Q_DEF (Default for IBM MQ, which is different to JMS that has a default of 4.)
Persistence	MLONG	QoS=0→MQPER_NOT_PERSISTENT QoS=1→MQPER_PERSISTENT QoS=2→MQPER_PERSISTENT

The RFH2 header does not contain an `<msd>` folder to define the type of the JMS message. The telemetry (MQXR) service creates the IBM MQ message as a default JMS message. The default JMS message-type is a `jms-bytes` message. An application can access additional header information as message properties; see Message properties.

RFH2 values are set as shown in Table 15. The Format property is set in the RFH2 fixed header and the other values are set in RFH2 folders.

Table 15. RFH2

RFH2 property	Type/Folder	Header
Format	MQCHAR8	MQFMT_NONE
ClientIdentifier	mqtt/clientId	Copy MqttClient.ClientIdentifier with a length of 1...23 bytes.
QoS	mqtt/qos	Copy QoS from incoming MQTT message.
Message ID	mqtt/msgid	Copy Message ID from incoming MQTT message, if QoS is 1 or 2.
MQIsRetained	mqs/Ret	Set if the original MQTT publication was sent with the RETAIN property set and the message is received as a retained publication.
MQTopicString	mqs/Top	The topic to which the MQTT message was published.

The payload in an MQTT publication is mapped to the contents of an IBM MQ message:

Table 16. Message contents

Message contents	Type	Contents
Buffer	MQBYTE <i>n</i>	Copy of bytes from incoming MQTT message. The length can be zero.

IBM MQ to an MQTT client

A client has subscribed to a publication topic. An IBM MQ application has published to the topic, resulting in a publication being sent to the MQTT subscriber by the IBM MQ publish/subscribe broker. Alternatively, an IBM MQ application has sent an unsolicited message directly to an MQTT client. Table 17 describes how the fixed message headers are set in the message that is sent to the MQTT client. Any other data in the IBM MQ message header, or any other headers, are discarded. The message data in the IBM MQ message is sent as the message payload in the MQTT message, with no alteration. The MQTT message is sent to the MQTT client by the telemetry (MQXR) service.

Table 17. MQTT fixed header properties

MQTT field	Type	Value
DUP	boolean	Set if QoS = 1 or 2, and the message was sent to this client in a previous transmission, and the message has not been acknowledged after a time.
QoS	int	<p>The way the value of QoS in an outgoing publication from the publish/subscribe broker in IBM MQ is set depends on the incoming publication. It depends on whether the incoming publication was sent from an MQTT client, or from an IBM MQ application.</p> <p>MQTT Lower value of the QoS in the incoming publication, and in the QoS requested by the subscriber.</p> <p>IBM MQ Lower value of the QoS derived from the incoming publication: MQPER_NOT_PERSISTENT→QoS=0 MQPER_PERSISTENT→QoS=2</p> <p>and the QoS requested by the subscriber. If the message is sent to the client without a subscription, QoS is set by default to 2. A client can alter this value by subscribing to DEFAULT.QoS with a different QoS.</p>
RETAIN	boolean	Set if the incoming publication has the retained property set.

Table 18 describes how the variable message headers are set in the MQTT message that is sent to the MQTT client.

Table 18. MQTT Variable header properties

MQTT field	Type	Value
Topic name	String	The topic string the message was published with.
Message ID	String	The last 2 bytes of the MQMD.MsgId property of the publication when it is placed in SYSTEM.MQTT.TRANSMIT.QUEUE.
Payload	byte[]	Direct copy of bytes from incoming publication to the publish/subscribe broker. The length can be zero.

MQTT stateless and stateful sessions



MQTT clients can create a stateful session with the queue manager. When a stateful MQTT client disconnects, the queue manager maintains the subscriptions created by the client, and in-flight messages. When the client reconnects, it resolves in-flight message. It sends any messages that are queued for delivery, and receives any messages published for its subscriptions while it was disconnected.

When an MQTT client connects to a telemetry channel it either starts a new session, or resumes an old session. A new session has no outstanding messages that have not been acknowledged, no subscriptions, and no publications awaiting delivery. When a client connects, it specifies whether to start with a clean session, or to resume an existing session; see Clean sessions.

If the client resumes an existing session, it continues as if the connection had not been broken. Publications awaiting delivery are sent to the client, and any message transfers that had not been committed, are completed. When a client in a persistent session disconnects from the telemetry (MQXR) service, any subscriptions the client created remain. Publications for the subscriptions are sent to the client when it reconnects. If it reconnects without resuming the old session, the publications are discarded by the telemetry (MQXR) service.

Session state information is saved by the queue manager in the SYSTEM.MQTT.PERSISTENT.STATE queue.

The IBM MQ administrator can disconnect and purge a session.

When an MQTT client is not connected



When a client is not connected the queue manager can continue to receive publications on its behalf. They are forwarded to the client when it reconnects. A client can create a “Last will and testament”, which the queue manager publishes on behalf of the client, if the client disconnects unexpectedly.

If you want to be notified when the client unexpectedly disconnects, you can register a last will and testament publication; see Last will and testament publication. It is sent by the telemetry (MQXR) service, if it detects the connection to the client has broken without the client requesting it.

A client can publish a retained publication at any time; see Retained publications and MQTT clients. A new subscription to a topic can request to be sent any retained publication associated with topic. If you create the last will and testament as a retained publication, you can use it to monitor the status of a client.

For example, the client publishes a retained publication, when it connects, advertising its availability. At the same time, it creates a retained last will and testament publication that announces its unavailability. In addition, just before it makes a planned disconnection, it publishes its unavailability as a retained publication. To find out whether the client is available, you would subscribe to the topic of the retained publication. You would always receive one of the three publications.

If the client is to receive messages published when it is disconnected, then reconnect the client to its previous session; see “MQTT stateless and stateful sessions” on page 279. Its subscriptions are active until they are deleted, or until the client creates a clean session.

Loose coupling between MQTT clients and IBM MQ applications



The flow of publications between MQTT clients and IBM MQ applications is loosely coupled. Publications might originate from either an MQTT client or an IBM MQ application, and in no set order. Publishers and subscribers are loosely coupled. They interact with each other indirectly through publications and subscriptions. You can also send messages directly to an MQTT client from an IBM MQ application.

MQTT clients and IBM MQ applications are loosely coupled in two senses:

1. Publishers and subscribers are loosely coupled by the association of a publication and a subscription with a topic. Publishers and subscribers are not normally aware of the address or identity of the other source of a publication or subscription.
2. MQTT clients publish, subscribe, receive publications, and process delivery acknowledgments on separate threads.

An MQTT client application does not wait until a publication has been delivered. The application passes a message to the MQTT client, and then the application continues on its own thread. A delivery-token is used to synchronize the application with the delivery of a publication; see Delivery tokens.

After passing a message to the MQTT client, the application has the choice of waiting on the delivery-token. Rather than waiting, the client can provide a callback method that is called when the publication is delivered to IBM MQ. It can also ignore the delivery-token.

Depending on the quality of service associated with the message, the delivery-token is returned immediately to the callback method, or possibly after some considerable time. The delivery-token might even be returned after the client has disconnected and reconnected. If the quality of service is "fire and forget", the delivery-token is returned immediately. In the other two cases, the delivery token is returned only when the client receives acknowledgment that the publication has been sent to subscribers.

Publications sent to an MQTT client as a result of a client subscription, are delivered to the `messageArrived` callback method. `messageArrived` runs on a different thread to the main application.

Sending messages directly to an MQTT client

You can send a message to a particular MQTT client in one of two ways.

1. An IBM MQ application can send a message directly to an MQTT client without a subscription; see Sending a message to a client directly.
2. An alternative approach is to use your `ClientIdentifier` naming convention. Make all MQTT subscribers create subscriptions using their unique `ClientIdentifier` as a topic. Publish to `ClientIdentifier`. The publication is sent to the client that subscribed to the topic `ClientIdentifier`. Using this technique you can send a publication to a particular MQTT subscriber.

MQ Telemetry security



Securing telemetry devices can be important, as the devices are likely to be portable, and used in places that cannot be carefully controlled. You can use VPN to secure the connection from the MQTT device to the telemetry (MQXR) service. MQ Telemetry provides two other security mechanisms, TLS and JAAS.

TLS is principally used to encrypt communications between the device and the telemetry channel, and to authenticate the device is connecting to the correct server; see Telemetry channel authentication using TLS. You can also use TLS to check that the client device is permitted to connect to the server; see MQTT client authentication using TLS.

JAAS is principally used to check that the user of the device is permitted to use a server application; see MQTT client authentication using a password. JAAS can be used with LDAP to check a password using a single sign-on directory.

TLS and JAAS can be used in conjunction to provide two factor authentication. You can restrict the ciphers used by TLS to ciphers that meet FIPS standards.

With at least tens of thousands of users, it is not always practical to provide individual security profiles. Nor is it always practical to use the profiles to authorize individual users to access IBM MQ objects. Instead group users into classes for authorizing publication and subscription to topics, and sending publications to clients.

Configure each telemetry channel to map clients to common client user IDs. Use a common user ID for every client that connects on a specific channel; see MQTT client identity and authorization.

Authorizing groups of users does not compromise authentication of each individual. Each individual user can be authenticated, at the client or server, with their Username and Password, and then authorized at the server using a common user ID.

MQ Telemetry globalization



The message payload in the MQTT v3 protocol is encoded as byte-array. Generally, applications handling text create the message payload in UTF-8. The telemetry channel describes the message payload as UTF-8, but does not do any code page conversions. The publication topic string must be UTF-8.

The application is responsible for converting alphabetic data to the correct code page and numeric data to the correct number encoding.

The MQTT Java client has a convenient `MqttMessage.toString` method. The method treats the message payload as being encoded in the local platform default character set, which is generally UTF-8. It converts the payload to a Java String. Java has a `String` method, `getBytes` that converts a string into a byte array encoded using the local platform default character set. Two MQTT Java programs exchanging text in the message payload, between platforms with the same default character set do so easily and efficiently in UTF-8.

If the default character set of one of the platforms is not UTF-8, then the applications must establish a convention for exchanging messages. For example, the publisher specifies conversion from a string to UTF-8 using the `getBytes("UTF8")` method. To receive the text of a message, the subscriber assumes that the message is encoded in the UTF-8 character set.

The telemetry (MQXR) service describes the encoding of all incoming publications from MQTT clients messages as being UTF-8. It sets MQMD.CodedCharSetId to UTF-8, and RFH2.CodedCharSetId to MQCCSI_INHERIT ; see “Integration of MQ Telemetry with queue managers” on page 277. The format of the publication is set to MQFMT_NONE, so no conversion can be performed by channels, or by MQGET.

Performance and scalability of MQ Telemetry



Consider the following factors when managing large numbers of clients and improving scalability of MQ Telemetry.

Capacity Planning

For information about performance reports for MQ Telemetry, select the MQ Telemetry Performance Evaluations report from the website IBM MQ Family - Performance Reports.

Connections

Costs involved with connections include

- The cost of setting up a connection itself in terms of processor usage and time.
- Network costs.
- Memory used when keeping a connection open but not using it.

There is an extra load incurred when clients stay connected. If a connection is kept open, TCP/IP flows and MQTT messages use the network to check that the connection is still there. Additionally, memory is used in the server for each client connection that is kept open.

If you are sending messages more than one per minute, keep your connection open to avoid the cost of initiating a new connection. If you are sending messages less than one every 10 - 15 minutes, consider dropping your connection to avoid the cost of keeping it open. You might want to keep a TLS connection open, but idle, for longer periods because it is more expensive to set up.

Additionally, consider the capability of the client. If there is a store and forward facility on the client then you might batch up messages and drop the connection between sending the batches. However, if the client is disconnected, then it is not possible for the client to receive a message from the server. Therefore the purpose of your application has a bearing on the decision.

If your system has one client sending many messages, for example file transfers, do not wait for a server response per message. Instead, send all messages and check at the end that they have all been received. Alternatively, use Quality of Service (QoS).

You can vary the QoS by message, delivering unimportant messages using QoS 0 and important messages using a QoS of 2. The message throughput can be around twice as high with a QoS of 0 than with a QoS of 2.

Naming conventions

If you are designing your application for many clients, implement an effective naming convention. In order to map each client to the correct ClientIdentifier, make the ClientIdentifier meaningful. A good naming convention makes it easier for the Administrator to work out which clients are running. A naming convention helps the administrator filter a long list of clients in IBM MQ Explorer, and helps with problem determination; see Client identifier.

Throughput

The length of topic names affects the number of bytes that flow across the network. When publishing or subscribing, the number of bytes in a message might be important. Therefore limit the number of characters in a topic name. When an MQTT client subscribes for a topic IBM MQ gives it a name of the form:

```
ClientIdentifier: TopicName
```

To view all of the subscriptions for an MQTT client, you can use the IBM MQ MQSC **DISPLAY** command:

```
DISPLAY SUB(' ClientID1:*')
```

Defining resources in IBM MQ for use by MQTT clients

An MQTT client connects to an IBM MQ remote queue manager. There are two basic methods for an IBM MQ application to send messages to an MQTT client: set the default transmission queue to `SYSTEM.MQTT.TRANSMIT.QUEUE` or use queue manager aliases. Define the default transmission queue of a queue manager, if there are large numbers of MQTT clients. Using the default transmission queue setting simplifies the administration effort; see [Configure distributed queuing to send messages to MQTT clients](#).

Improving scalability by avoiding subscriptions.

When an MQTT V3 client subscribes to a topic, a subscription is created by the telemetry (MQXR) service in IBM MQ. The subscription routes publications for the client onto `SYSTEM.MQTT.TRANSMIT.QUEUE`. The remote queue manager name in the transmission header of each publication is set to the `ClientIdentifier` of the MQTT client that made the subscription. If there are many clients, each making their own subscriptions, this results in many proxy subscriptions being maintained throughout the IBM MQ publish/subscribe cluster or hierarchy. For information about not using publish/subscribe, but using a point to point based solution instead, see [Sending a message to a client directly](#).

Managing large numbers of clients

To support many concurrently connected clients, increase the memory available for the telemetry (MQXR) service by setting the JVM parameters `-Xms` and `-Xmx`. Follow these steps:

1. Find the `java.properties` file in the telemetry service configuration directory; see [Telemetry \(MQXR\) service configuration directory on Windows](#) or [Telemetry service configuration directory on Linux](#).
2. Follow the directions in the file; a heap of 1 GB is sufficient for 50,000 concurrently connected clients.

```
# Heap sizing options - uncomment the following lines to set the heap to 1G
#-Xmx1024m
#-Xms1024m
```

3. Add other command-line arguments to pass to the JVM running the telemetry (MQXR) service in the `java.properties` file; see [Passing JVM parameters to the telemetry \(MQXR\) service](#).

To increase the number of open file descriptors on Linux, add the following lines to `/etc/security/limits.conf/`, and log in again.

```
@mqm soft nofile 65000
@mqm hard nofile 65000
```

Each socket requires one file descriptor. The telemetry service require some additional file descriptors, so this number must be larger than the number of open sockets required.

The queue manager uses an object handle for each nondurable subscription. To support many active, nondurable subscriptions increase the maximum number of active handles in the queue manager; for example:

```
echo ALTER QMGR MAXHANDS(999999999) | runmqsc qMgrName
```

Figure 49. Alter maximum number of handles on Windows

```
echo "ALTER QMGR MAXHANDS(999999999)" | runmqsc qMgrName
```

Figure 50. Alter maximum number of handles on Linux

Other considerations

When planning your system requirements, consider the length of time taken to restart the system. The planned downtime might have implications for the number of messages that queue up, waiting to be processed. Configure the system so that the messages can be successfully processed in an acceptable time. Review disk storage, memory, and processing power. With some client applications, it might be possible to discard messages when the client reconnects. To discard messages, set `CleanSession` in the client connection parameters; see [Clean sessions](#). Alternatively, publish and subscribe using the best effort Quality of Service, 0, in an MQTT client; see [Quality of service](#). Use non-persistent messages when sending messages from IBM MQ. Messages with these qualities of service are not recovered when the system or connection restarts.

Devices supported by MQ Telemetry



MQTT clients can run on a range of devices, from sensors and actuators, to hand held devices and vehicle systems.

MQTT clients are small, and run on devices constrained by little memory and low processing power. The MQTT protocol is reliable and has small headers, which suits networks constrained by low bandwidth, high cost, and intermittent availability.

MQ Telemetry communicates with telemetry devices through MQTT client applications. These applications use the following resources, all of which implement the MQTT v3 protocol:

- The following client libraries:
 - The *MQTT client for Java*, which is used for building native applications for (for example) Android, OS X, Linux or Windows devices. Applications that use this client library can run on all variations of Java from the smallest CLDC (Connected Limited Device Configuration)/MIDP (Mobile Information Device Profile) through CDC (Connected Device Configuration)/Foundation, J2SE (Java Platform, Standard Edition), and J2EE (Java Platform, Enterprise Edition). IBM jclRM customized class library is also supported. The Java ME platform is generally used on small devices, such as actuators, sensors, mobile phones, and other embedded devices. The Java SE platform is generally installed on higher end embedded devices, such as desktop computers and servers.
 - The *MQTT client for C*, which is used for building native applications for (for example) iOS, OS X, Linux or Windows devices. This client library provides a C reference implementation together with prebuilt native client for Windows and Linux systems. The C reference implementation enables MQTT to be ported to a wide range of devices and platforms. Some Windows systems on Intel, including Windows 7, RedHat, Ubuntu, and some Linux systems on ARM platforms such as Eurotech Viper, implement versions of Linux that run the C client, but IBM does not provide service support for the platforms. You must reproduce problems with the client on a supported platform if you intend to call your IBM support centre.
 - The *MQTT messaging client for JavaScript*, which is used for building browser-based web applications.

The MQTT client libraries are available as part of the IBM Messaging Telemetry Clients SupportPac.

Security in IBM MQ

In IBM MQ, there are several methods of providing security: the authorization service interface; user-written, or third party, channel exits; channel security using Transport Layer Security (TLS), channel authentication records , and message security.

Authorization service interface

Authorization for using MQI calls, commands, and access to objects is provided by the **object authority manager (OAM)**, which by default is enabled. Access to IBM MQ entities is controlled through IBM MQ user groups and the OAM. Administrators can use a command-line interface to grant or revoke authorizations as required.

For more information about creating authorization service components, see [Setting up security on Windows, UNIX and Linux systems](#).

User-written or third party channel exits

Channels can use user-written or third party channel exits. For more information, see [Channel-exit programs for messaging channels](#).

Channel security using TLS

The Transport Layer Security (TLS) protocol provides industry-standard channel security, with protection against eavesdropping, tampering, and impersonation.

TLS uses public key and symmetric techniques to provide message confidentiality and integrity and mutual authentication.

For a comprehensive review of security in IBM MQ including detailed information about TLS, see [Securing](#). For an overview of TLS, including pointers to the commands described in this section, see [Cryptographic security protocols: TLS](#).

Channel authentication records

Use channel authentication records to exercise precise control over the access granted to connecting systems at a channel level. For more information, see [Channel authentication records](#).

Message security

Use Advanced Message Security, which is a separately installed and licensed component of IBM MQ, to provide cryptographic protection to messages sent and receive using IBM MQ. See [Advanced Message Security](#).

Related information:

Securing

Planning for your security requirements

IBM MQ.NET managed client TLS support

The IBM MQ.NET fully managed client provides Transport Layer Security (TLS) support that is based on the Microsoft.NET SSLStreams kit. This is different from the other IBM MQ clients, which are based on GSKit.

You can develop IBM MQ.NET applications to run in managed mode or unmanaged mode.

- In managed mode, .NET applications work within the .NET CLR (Common Language Runtime) without any cross platform invocation such as invoking the C MQI.
- In unmanaged mode, the C MQI is invoked for the underlying MQI operations. Basically, the unmanaged mode interface comprises the .NET wrapper classes on top of the C MQI.

The managed IBM MQ.NET client uses the Microsoft.NET Framework libraries to implement TLS secure socket protocols. The System.NET.Security.SSLStream class from Microsoft is used for implementing Security (TLS) in IBM MQ.NET.

The unmanaged IBM MQ.NET client mode already supports the TLS feature, which is based on C MQI (and GSKit). That is, the TLS operations are handled by the C MQI. In this case, GSKit implements the TLS secure socket protocols.

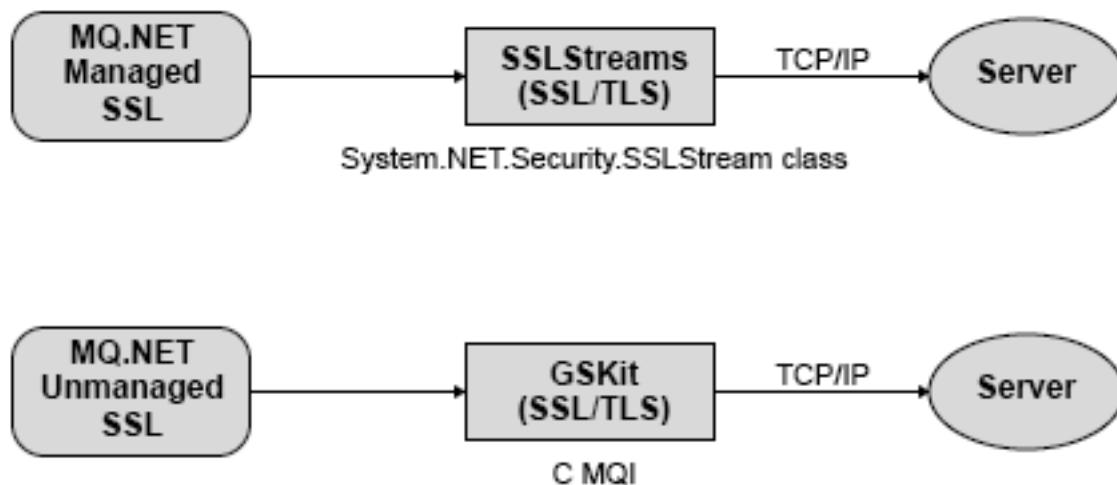


Figure 51. IBM MQ.NET managed and unmanaged TLS comparison

The following table summarizes the differences between the managed and unmanaged implementations:

Table 19. Differences between managed and unmanaged implementations

Mode	Protocols	Implementation	Comments
IBM MQ.NET managed SSL	TLS	System.NET.Security.SSLStream class SSLStream class operates as a stream over a connected TCP socket	TLS 1.0 TLS1.2 (with Microsoft.NET Framework v4.5 only)
IBM MQ.NET unmanaged SSL	TLS	GSKIT and C-MQI	TLS secure socket protocols

Related information:

Secure Sockets Layer (SSL) and Transport Layer Security (TLS) support for .NET

Clients and servers

An introduction to how IBM MQ supports client-server configurations for its applications.

An IBM MQ MQI *client* is a component that allows an application running on a system to issue MQI calls to a queue manager running on another system. The output from the call is sent back to the client, which passes it back to the application.

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 machine (the IBM MQ server machine), and not on the client. An IBM MQ server can also support local IBM MQ applications.

The difference between an IBM MQ server and an ordinary queue manager is that a server has a dedicated communications link with each client. For more information about creating channels for clients and servers, see *Configuring distributed queuing*.

For information about clients in general, see “Overview of IBM MQ MQI clients.”

IBM MQ applications in a client-server environment

When linked to a server, client IBM MQ applications can issue most MQI calls in the same way as local applications. The client application issues an MQCONN call to connect to a specified queue manager. Any additional MQI calls that specify the connection handle returned from the connect request are then processed by this queue manager.

You must link your applications to the appropriate client libraries. See *Building applications for IBM MQ MQI clients*.

Related concepts:

“Transaction management and support” on page 295

An introduction to transaction management and how IBM MQ supports transactions.

“Extending queue manager facilities” on page 297

You can extend queue manager facilities by using user exits, API exits, or installable services.

Overview of IBM MQ MQI 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.

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 an *MQI channel*. An MQI channel starts when the client application issues an **MQCONN** or **MQCONNX** call to connect to the queue manager and ends when the client application issues an **MQDISC** call to disconnect from the queue manager. The input parameters of an MQI call flow in one direction on an MQI channel and the output parameters flow in the opposite direction.

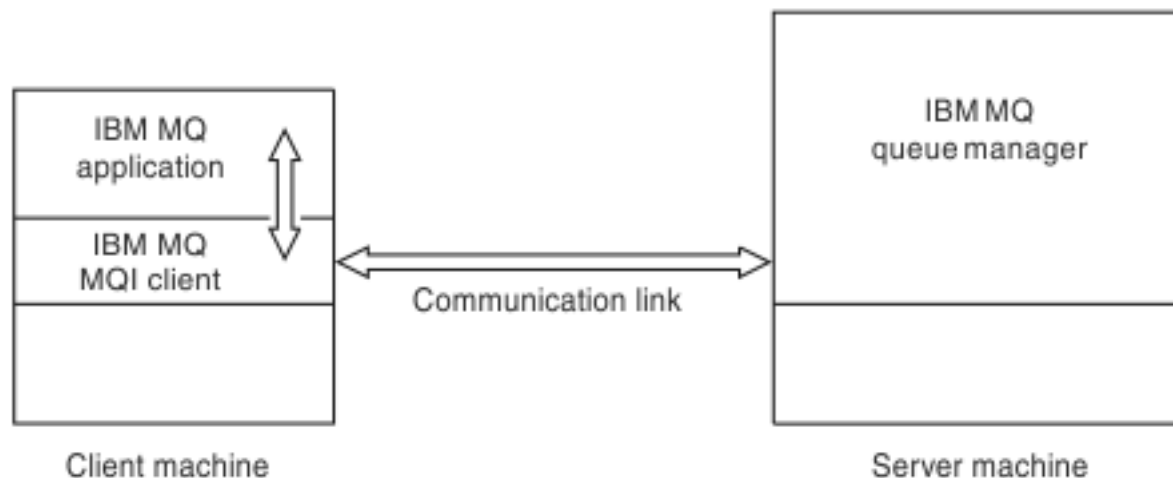


Figure 52. Link between a client and server

The following platforms can be used. The combinations depend on which IBM MQ product you are using and are described in “Platform support for IBM MQ clients” on page 290.

IBM MQ MQI client

UNIX and Linux
Windows
IBM i
HP Integrity NonStop Server

IBM MQ server




UNIX and Linux
Windows
IBM i
z/OS
HP Integrity NonStop Server

The MQI is available to applications running on the client platform; the queues and other IBM MQ objects are held on a queue manager that you have installed on a server.

An application that you want to run in the IBM MQ MQI client environment must first be linked with the relevant client library. When the application issues an MQI call, the IBM MQ MQI client directs the request to a queue manager, where it is processed and from where a reply is sent back to the IBM MQ MQI client.

The link between the application and the IBM MQ MQI client is established dynamically at run time.

You can also develop client applications using the IBM MQ classes for .NET, IBM MQ classes for Java or IBM MQ classes for Java Message Service (JMS). You can use Java and JMS clients on the following platforms:

-  IBM i
-  UNIX
-  Linux
-  Windows

The use of Java and JMS is not described here. For full details on how to install, configure, and use IBM MQ classes for Java and IBM MQ classes for JMS see Using IBM MQ classes for Java and Using IBM MQ classes for JMS.

Related concepts:

“Why use IBM MQ clients?”

Using IBM MQ clients is an efficient way of implementing IBM MQ messaging and queuing.

“How to set up an IBM MQ MQI client” on page 291

Follow these instructions to set up a client.

“What is an extended transactional client?” on page 292

An IBM MQ extended transactional client can update resources managed by another resource manager, under the control of an external transaction manager.

“How the client connects to the server” on page 293

A client connects to a server using MQCONN or MQCONNX, and communicates through a channel.

Why use IBM MQ clients?

Using IBM MQ clients is an efficient way of implementing IBM MQ messaging and queuing.

You can have an application that uses the MQI running on one machine and the queue manager running on a different machine (either physical or virtual). The benefits of doing this are:

- There is no need for a full IBM MQ implementation on the client machine.
- Hardware requirements on the client system are reduced.
- System administration requirements are reduced.
- An IBM MQ application running on a client can connect to multiple queue managers on different systems.
- Alternative channels using different transmission protocols can be used.

Related reference:

“What applications run on an IBM MQ MQI client?”

The full MQI is supported in the client environment.

“Platform support for IBM MQ clients” on page 290

IBM MQ on all supported server platforms accepts client connections from IBM MQ MQI clients on a number of platforms.

What applications run on an IBM MQ MQI client?:

The full MQI is supported in the client environment.

This enables almost any IBM MQ application to be configured to run on an IBM MQ MQI client system by linking the application on the IBM MQ MQI client to the MQIC library, rather than to the MQI library. The exceptions are:

- MQGET with signal
- An application that needs sync point coordination with other resource managers must use an extended transactional client

If read ahead is enabled, to improve non persistent messaging performance, not all MQGET options are available. The table shows the options that are allowed, and whether they can be altered between MQGET calls.

Table 20. MQGET options permitted when read ahead is enabled

	Permitted when read ahead is enabled and can be altered between MQGET calls	Permitted when read ahead is enabled but cannot be altered between MQGET calls ¹	MQGET options that are not permitted when read ahead is enabled ²
MQGET MD values	MsgId ³ CorrelId ³	Encoding CodedCharSetId	
MQGET MQGMO options	MQGMO_WAIT MQGMO_NO_WAIT MQGMO_FAIL_IF QUIESCING MQGMO_BROWSE_FIRST ⁴ MQGMO_BROWSE_NEXT ⁴ MQGMO_BROWSE_MESSAGE _UNDER_CURSOR ⁴	MQGMO_SYNCPOINT_IF_PERSISTENT MQGMO_NO_SYNCPOINT MQGMO_ACCEPT_TRUNCATED_MSG MQGMO_CONVERT MQGMO_LOGICAL_ORDER MQGMO_COMPLETE_MSG MQGMO_ALL_MSGS_AVAILABLE MQGMO_ALL_SEGMENTS_AVAILABLE MQGMO_MARK_BROWSE_HANDLE MQGMO_MARK_BROWSE_CO_OP MQGMO_UNMARK_BROWSE_CO_OP MQGMO_UNMARK_BROWSE_HANDLE MQGMO_UNMARKED_BROWSE_MSG MQGMO_PROPERTIES_FORCE_MQRFH2 MQGMO_NO_PROPERTIES MQGMO_PROPERTIES_IN_HANDLE MQGMO_PROPERTIES_COMPATIBILITY	MQGMO_SET_SIGNAL MQGMO_SYNCPOINT MQGMO_MARK_SKIP _BACKOUT MQGMO_MSG_UNDER _CURSOR ⁴ MQGMO_LOCK MQGMO_UNLOCK
MQGMO values		MsgHandle	






1. If these options are altered between MQGET calls an MQRC_OPTIONS_CHANGED reason code is returned.
2. If these options are specified on the first MQGET call then read ahead is disabled. If these options are specified on a subsequent MQGET call a reason code MQRC_OPTIONS_ERROR is returned.
3. The client applications need to be aware that if the MsgId and CorrelId values are altered between MQGET calls messages with the previous values might have already been sent to the client and remain in the client read ahead buffer until consumed (or automatically purged).
4. The first MQGET call determines whether messages are to be browsed or got from a queue when read ahead is enabled. If the application attempts to use a combination of browse and get an MQRC_OPTIONS_CHANGED reason code is returned.
5. MQGMO_MSG_UNDER_CURSOR is not possible with read ahead. Messages can be browsed or got when read ahead is enabled but not a combination of both.

An application running on an IBM MQ MQI client can connect to more than one queue manager concurrently, or use a queue manager name with an asterisk (*) on an MQCONN or MQCONNX call (see the examples in Connecting IBM MQ MQI client applications to queue managers).

Platform support for IBM MQ clients:

IBM MQ on all supported server platforms accepts client connections from IBM MQ MQI clients on a number of platforms.

IBM MQ installed as a *Base product and Server* on all supported server platforms can accept connections from the IBM MQ MQI clients on the following platforms:








-  HP Integrity NonStop Server
-  IBM i
-  Linux
-  UNIX
-  Windows

Client connections are subject to differences in coded character set identifier (CCSID) and communications protocol.

How to set up an IBM MQ MQI client

Follow these instructions to set up a client.

To set up an IBM MQ MQI client you must have an IBM MQ server already installed and working, to which your client will connect. The steps involved in setting up a client are:

1. Check that you have a suitable platform for an IBM MQ MQI client and that the hardware and software satisfy the requirements. Platform support is described in “Platform support for IBM MQ clients” on page 290.
2. Decide how you are going to install IBM MQ on your client workstation, and then follow the instructions for your particular combination of client and server platforms. Installation is described in the following topics:
 -  Installing an IBM MQ client on AIX
 -  Installing on HP Integrity NonStop Server
 -  Installing an IBM MQ client on HP-UX
 -  Installing an IBM MQ client on Linux
 -  Installing an IBM MQ client on Solaris
 -  Installing an IBM MQ client on Windows
 -  Installing an IBM MQ client on IBM i
3. Ensure that your communication links are configured and connected. Configuration of communication links is described in Configuring connections between the server and client.
4. Check that your installation is working correctly. See the verification section of the installation procedure for the platform, or platforms, that your enterprise use.
5. When you have the verified IBM MQ MQI client installation, consider whether you must secure your client. Client security is described in Setting up IBM MQ MQI client security.
6. Set up the channels between the IBM MQ MQI client and server that are required by the IBM MQ applications you want to run on the client. Setting up channels is described in Defining MQI channels. There are some additional considerations if you are using TLS. These considerations are described in Specifying that an MQI channel uses TLS. You might need to use an IBM MQ MQI client configuration file or IBM MQ environment variables to set up the channels. IBM MQ environment variables are described in Using IBM MQ environment variables.
7. IBM MQ applications are fully described in Developing applications.
8. There are some differences from a queue manager environment to consider when designing, building, and running applications in the IBM MQ MQI client environment. For information about these differences, see:
 - Using the message queue interface (MQI) in a client application
 - Building applications for IBM MQ MQI clients
 - Connecting IBM MQ MQI client applications to queue managers
 - Resolving problems with IBM MQ MQI clients

What is an extended transactional client?

An IBM MQ extended transactional client can update resources managed by another resource manager, under the control of an external transaction manager.

If you are not familiar with the concepts of transaction management, see “Transaction management and support” on page 295.

Note that the XA transactional client is now supplied as part of IBM MQ.

A client application can participate in a unit of work that is managed by a queue manager to which it is connected. Within the unit of work, the client application can put messages to, and get messages from, the queues that are owned by that queue manager. The client application can then use the **MQCMIT** call to commit the unit of work or the **MQBACK** call to back out the unit of work. However, within the same unit of work, the client application cannot update the resources of another resource manager, the tables of a Db2 database, for example. Using an IBM MQ extended transactional client removes this restriction.


An IBM MQ extended transactional client is an IBM MQ MQI client with some additional function. Using this function a client application, within the same unit of work, can perform the following tasks:

- Put messages to, and get messages from, queues that are owned by the queue manager to which it is connected
- Update the resources of a resource manager other than an IBM MQ queue manager

This unit of work must be managed by an external transaction manager that is running on the same system as the client application. The unit of work cannot be managed by the queue manager to which the client application is connected. This means that the queue manager can act only as a resource manager, not as a transaction manager. It also means that the client application can commit or back out the unit of work using only the application programming interface (API) provided by the external transaction manager. The client application cannot, therefore, use the MQI calls, **MQBEGIN**, **MQCMIT**, and **MQBACK**.

The external transaction manager communicates with the queue manager as a resource manager using the same MQI channel as used by the client application that is connected to the queue manager. However, in a recovery situation following a failure, when no applications are running, the transaction manager can use a dedicated MQI channel to recover any incomplete units of work in which the queue manager was participating at the time of the failure.

In this section, an IBM MQ MQI client that does not have the extended transactional function is referred to as an IBM MQ base client. You can consider, therefore, an IBM MQ extended transactional client to consist of an IBM MQ base client with the addition of the extended transactional function.

Note:  IBM MQ MQI client on IBM i does not support the IBM MQ extended transactional function.

Related reference:








“Platform support for extended transactional clients”


Extended transactional clients are available for all Multiplatforms that support a base client. The clients are not available for z/OS.

Platform support for extended transactional clients:

Extended transactional clients are available for all Multiplatforms that support a base client. The clients are not available for z/OS.

A client application that is using an extended transactional client can connect to a queue manager of the following IBM MQ Version 9.0 products only:

-  IBM MQ for AIX
-  IBM MQ for HP-UX
-  IBM MQ client for HP Integrity NonStop Server
-  IBM MQ for IBM i
-  IBM MQ for Linux
-  IBM MQ for Solaris
-  IBM MQ for Windows

 Although there are no extended transactional clients that run on z/OS, a client application that is using an extended transactional client can connect to a queue manager that runs on z/OS.

For each platform, the hardware and software requirements for the extended transactional client are the same as those requirements for the IBM MQ base client. A programming language is supported by an extended transactional client if it is supported by the IBM MQ base client and by the transaction manager you are using.

For information about the external transaction managers for all platforms except HP Integrity NonStop Server, see System Requirements for IBM MQ.

For HP Integrity NonStop Server, see Planning your IBM MQ client environment on HP Integrity NonStop Server.

How the client connects to the server

A client connects to a server using MQCONN or MQCONNX, and communicates through a channel.

An application running in the IBM MQ client environment must maintain an active connection between the client and server machines.

The connection is made by an application issuing an MQCONN or MQCONNX call. Clients and servers communicate through *MQI channels*, or, when using sharing conversations, conversations each share an MQI channel instance. When the call succeeds, the MQI channel instance or conversation remains connected until the application issues a MQDISC call. This is the case for every queue manager that an application needs to connect to.

Related concepts:

“Client and queue manager on the same machine”

You can also run an application in the IBM MQ MQI client environment when your machine also has a queue manager installed.

“Clients on different platforms”

Here is another example of an IBM MQ MQI client and server system. In this example, the server machine communicates with three IBM MQ MQI clients on different platforms.

“Using different versions of client and server software” on page 295

If you are using previous versions of IBM MQ products, make sure that code conversion from the CCSID of your client is supported by the server.

Client and queue manager on the same machine:

You can also run an application in the IBM MQ MQI client environment when your machine also has a queue manager installed.

In this situation, you have the choice of linking to the queue manager libraries or the client libraries, but remember that if you link to the client libraries, you still need to define the channel connections. This can be useful during the development phase of an application. You can test your program on your own machine, with no dependency on others, and be confident that it will still work when you move it to an independent IBM MQ MQI client environment.

Clients on different platforms:

Here is another example of an IBM MQ MQI client and server system. In this example, the server machine communicates with three IBM MQ MQI clients on different platforms.

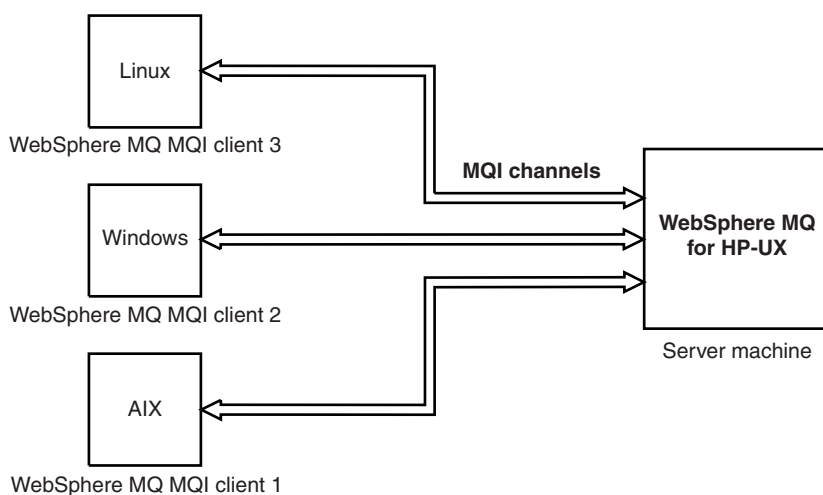


Figure 53. IBM MQ server connected to clients on different platforms

Other more complex environments are possible. For example, an IBM MQ client can connect to more than one queue manager, or any number of queue managers connected as part of a queue-sharing group.

Using different versions of client and server software:

If you are using previous versions of IBM MQ products, make sure that code conversion from the CCSID of your client is supported by the server.

An IBM WebSphere MQ Version 7.0, or later, client can connect to all supported versions of queue manager. If you are connecting to an earlier version queue manager, you cannot use features and structures from a later version of the product in your IBM MQ application on the client.

An IBM WebSphere MQ Version 7.0, or later, queue manager accepts connections from all supported versions of client.

See the programming languages supported in Developing applications for more information.

Transaction management and support

An introduction to transaction management and how IBM MQ supports transactions.

A *resource manager* is a computer subsystem that owns and manages resources that can be accessed and updated by applications. Here are examples of resource managers:

- An IBM MQ queue manager, with resources that are its queues
- A Db2 database, with resources that are its tables

When an application updates the resources of one or more resource managers, there might be a business requirement to ensure that certain updates all complete successfully as a group, or none of them complete. The reason for this kind of requirement is that the business data would be left in an inconsistent state if some of these updates completed successfully, but others did not.

Updates to resources that are managed in this way are said to occur within a *unit of work*, or a *transaction*. An application program can group a set of updates into a unit of work.

During a unit of work, an application issues requests to resource managers to update their resources. The unit of work ends when the application issues a request to commit all the updates. Until the updates are committed, none of them become visible to other applications that are accessing the same resources. Alternatively, if the application decides that it cannot complete the unit of work for any reason, it can issue a request to back out all the updates it has requested up to that point. In this case, none of the updates ever become visible to other applications. These updates are usually logically related and must all be successful for data integrity to be preserved. If one update succeeds while another fails, data integrity is lost.

When a unit of work completes successfully, it is said to *commit*. Once committed, all updates made within that unit of work are made permanent and irreversible. However, if the unit of work fails, all updates are instead *backed out*. This process, where units of work are either committed or backed out with integrity, is known as *sync point coordination*.

The point in time when all the updates within a unit of work are either committed or backed out is called a *sync point*. An update within a unit of work is said to occur *within sync point control*. If an application requests an update that is *outside of sync point control*, the resource manager commits the update immediately, even if there is a unit of work in progress, and the update cannot be backed out later.

The computer subsystem that manages units of work is called a *transaction manager*, or a *point coordinator*.

A *local* unit of work is one in which the only resources updated are those of the IBM MQ queue manager. Here sync point coordination is provided by the queue manager itself using a single-phase commit process.

A *global* unit of work is one in which resources belonging to other resource managers, such as XA-compliant databases, are also updated. Here, a two-phase commit procedure must be used and the unit of work can be coordinated by the queue manager itself, or externally by another XA-compliant transaction manager such as IBM TXSeries®, or BEA Tuxedo.

A transaction manager is responsible for ensuring that all updates to resources within a unit of work complete successfully, or none of them complete. It is to a transaction manager that an application issues a request to commit or back out a unit of work. Examples of transaction managers are CICS and WebSphere Application Server, although both of these possess other function as well.

Some resource managers provide their own transaction management function. For example, an IBM MQ queue manager can manage units of work involving updates to its own resources and updates to Db2 tables. The queue manager does not need a separate transaction manager to perform this function, although one can be used if it is a user requirement. If a separate transaction manager is used, it is referred to as an *external transaction manager*.

For an external transaction manager to manage a unit of work, there must be a standard interface between the transaction manager and every resource manager that is participating in the unit of work. This interface allows the transaction manager and a resource manager to communicate with each other. One of these interfaces is the *XA Interface*, which is a standard interface supported by a number of transaction managers and resource managers. The XA Interface is published by The Open Group in *Distributed Transaction Processing: The XA Specification*.

When more than one resource manager participates in a unit of work, a transaction manager must use a *two-phase commit* protocol to ensure that all the updates within the unit of work complete successfully or none of them complete, even if there is a system failure. When an application issues a request to a transaction manager to commit a unit of work, the transaction manager does the following:

Phase 1 (Prepare to commit)

The transaction manager asks each resource manager participating in the unit of work to ensure that all the information about the intended updates to its resources is in a recoverable state. A resource manager normally does this by writing the information to a log and ensuring that the information is written through to hard disk. Phase 1 completes when the transaction manager receives notification from each resource manager that the information about the intended updates to its resources is in a recoverable state.

Phase 2 (Commit)

When Phase 1 is complete, the transaction manager makes the irrevocable decision to commit the unit of work. It asks each resource manager participating in the unit of work to commit the updates to its resources. When a resource manager receives this request, it must commit the updates. It does not have the option to back them out at this stage. Phase 2 completes when the transaction manager receives notification from each resource manager that it has committed the updates to its resources.

The XA Interface uses a two-phase commit protocol.

For more information, see Transactional support scenarios.

IBM MQ also provides support for the Microsoft Transaction Server (COM+). Using the Microsoft Transaction Server (COM+) provides information on how to set up IBM MQ to take advantage of COM+ support.

Extending queue manager facilities

You can extend queue manager facilities by using user exits, API exits, or installable services.

User exits

User exits provide a mechanism for you to insert your own code into a queue manager function. The user exits supported include:

Channel exits

These exits change the way that channels operate. Channel exits are described in Channel-exit programs for messaging channels.

Data conversion exits

These exits create source code fragments that can be put into application programs to convert data from one format to another. Data conversion exits are described in the Writing data-conversion exits.

The cluster workload exit

The function performed by this exit is defined by the provider of the exit. Call definition information is given in MQ_CLUSTER_WORKLOAD_EXIT - Call description.

API exits

API exits let you write code that changes the behavior of IBM MQ API calls, such as MQPUT and MQGET, and then insert that code immediately before or immediately after those calls. The insertion is automatic; the queue manager drives the exit code at the registered points. For more information about API exits, see Using and writing API exits.

Installable services

Installable services have formalized interfaces (an API) with multiple entry points.

An implementation of an installable service is called a *service component*. You can use the components supplied with IBM MQ, or you can write your own component to perform the functions that you require.

Currently, the following installable services are provided:

Authorization service

The authorization service allows you to build your own security facility.

The default service component that implements the service is the object authority manager (OAM). By default, the OAM is active, and you do not have to do anything to configure it. You can use the authorization service interface to create other components to replace or augment the OAM. For more information about the OAM, see Setting up security on Windows, UNIX and Linux systems.

Name service

The name service enables applications to share queues by identifying remote queues as though they were local queues.

You can write your own name service component. You might want to do this if you intend to use the name service with IBM MQ, for example. To use the name service you must have either a component that is either user-written, or supplied by a different software vendor. By default, the name service is inactive.

IBM MQ Java language interfaces

IBM MQ provides two alternative application programming interfaces (APIs) for use in Java applications: IBM MQ classes for Java Message Service and IBM MQ classes for Java.

IBM supports, and is an active participant of, open standards and within the messaging area the API standard is the Java Message Service (JMS). From Version 8.0, IBM MQ implements the JMS 2.0 standard, which introduces a new simplified API together with features such as shared subscriptions. In addition, WebSphere Application Server Liberty has support for JMS 2.0 with both the default messaging provider and IBM MQ.

Within IBM MQ there are two alternative APIs for use in Java applications:

IBM MQ classes for JMS

IBM MQ classes for Java Message Service (JMS) is the JMS provider that is supplied with IBM MQ. The Java Platform, Enterprise Edition Connector Architecture (JCA) provides a standard way of connecting applications running in a Java EE environment to an Enterprise Information System (EIS) such as IBM MQ or Db2.

IBM MQ classes for Java

IBM MQ classes for Java enable you to use IBM MQ in a Java environment. IBM MQ classes for Java allow a Java application to connect to IBM MQ as an IBM MQ client, or connect directly to an IBM MQ queue manager.

Note: IBM MQ classes for Java are functionally stabilized at the level shipped in IBM MQ Version 8.0. for more information, see “Stabilization of IBM MQ classes for Java” on page 18.

From IBM MQ Version 8.0, the IBM MQ classes for Java and IBM MQ classes for JMS are built with Java 7. The Java 7 runtime environment supports running earlier class file versions.

Related information:

Why should I use IBM MQ classes for JMS?

Why should I use IBM MQ classes for Java?

Using JMS 2.0 functionality

The JMS model

IBM MQ classes for JMS

IBM MQ classes for JMS is the JMS provider that is supplied with IBM MQ. IBM MQ classes for JMS implements the interfaces defined in the `javax.jms` package, and also provides two sets of extensions to the JMS API. Both Java Platform, Standard Edition (Java SE) and Java Platform, Enterprise Edition (Java EE) applications can use IBM MQ classes for JMS.

The JMS specification defines a set of interfaces that applications can use to perform messaging operations. From Version 8.0, IBM MQ supports the JMS 2.0 version of the JMS standard. This implementation offers all the features of the classic API but requires fewer interfaces and is simpler to use. For more information, see The JMS model and the JMS 2.0 specification at Java.net.

The `javax.jms` package specifies the details of the JMS interfaces, and a JMS provider implements these interfaces for a specific messaging product. IBM MQ classes for JMS is a JMS provider that implements the JMS interfaces for IBM MQ.

The flow of logic within a JMS application starts with `ConnectionFactory` and `Destination` objects. The application uses a `ConnectionFactory` object to create a `Connection` object, which represents the active connection from the application to a messaging server. The application uses the `Connection` object to create a `Session` object, which is a single threaded context for producing and consuming messages. The application can then use the `Session` object and a `Destination` object to create a `MessageProducer` object,

which the application uses to send messages to the specified destination. The destination is either a queue or a topic in the messaging system and is encapsulated by the Destination object. The application can also use the Session object and a Destination object to create a MessageConsumer object, which the application uses to receive messages that have been sent to the specified destination.

The JMS specification expects ConnectionFactory and Destination objects to be administered objects. An administrator creates and maintains administered objects in a central repository, and a JMS application retrieves these objects using the Java Naming and Directory Interface (JNDI). The repository of administered objects can range from a simple file to a Lightweight Directory Access Protocol (LDAP) directory.

IBM MQ classes for JMS supports the use of administered objects. An application can use all the features of IBM MQ classes for JMS that are exposed through IBM MQ without having any IBM MQ -specific information hard coded into the application itself. This arrangement provides the application with a degree of independence from the underlying IBM MQ configuration. To achieve this independence, the application can use JNDI to retrieve connection factories and destinations that are stored as administered objects, and use only the interfaces defined in the javax.jms package to perform messaging operations. An administrator can use the IBM MQ JMS administration tool or IBM MQ Explorer to create and maintain administered objects in a central repository. An application server, however, typically provides its own repository for administered objects and its own tools for creating and maintaining the objects. A Java EE application can therefore use JNDI to retrieve administered objects either from the applications server repository or from a central repository.

IBM MQ classes for JMS also provides extensions to the JMS API. Previous releases of IBM MQ classes for JMS contain extensions that are implemented in MQConnectionFactory, MQQueue, and MQTopic objects. These objects have properties and methods that are specific to IBM MQ. The objects can be administered objects, or an application can create the objects dynamically at run time. From IBM WebSphere MQ Version 7.0, IBM MQ classes for JMS maintains these extensions, and you can continue to use, without change, any applications that use these extensions. These extensions are known as the IBM MQ JMS extensions. Note that, in this set of documentation, objects that are created dynamically by an application at run time are not considered to be administered objects.

In addition to the IBM MQ JMS extensions, from IBM WebSphere MQ Version 7.0, IBM MQ classes for JMS provides a more generic set of extensions to the JMS API. These extensions are known as the IBM JMS extensions, and have the following broad objectives:

- To provide a greater level of consistency across IBM JMS providers
- To make it easier to write a bridge application between two IBM messaging systems
- To make it easier to port an application from one IBM JMS provider to another

The main focus of these extensions concerns creating and configuring connection factories and destinations dynamically at run time, but the extensions also provide function that is not directly related to messaging, such as function for problem determination.

A connection factory, queue, or topic object created using the javax.jms interface or either set of JMS extensions can be addressed using any of these APIs; that is, it can be cast to any of the interfaces. To maintain application portability at the highest level, use the most generic API that is suitable for your requirements.

Both Java SE and Java EE applications can use IBM MQ classes for JMS. On the Java EE platform, IBM MQ classes for JMS supports two types of communication between a component of an application and an IBM MQ queue manager:

Outbound communication

Using the JMS API directly, an application component creates a connection to a queue manager, and then sends and receives messages.

For example, the application component can be an application client, a servlet, a JavaServer Page (JSP), an enterprise Java bean (EJB), or a message driven bean (MDB). In this type of communication, the application server container provides only low-level functions in support of messaging operations, such as connection pooling and thread management.

Inbound communication

A message arriving at a destination is delivered to an MDB, which then processes the message.

Java EE applications use MDBs to process messages asynchronously. An MDB acts as a JMS message listener and is implemented by an `onMessage()` method, which defines how a message is processed. An MDB is deployed in the EJB container of an application server. The precise way in which an MDB is configured depends on which application server you are using, but the configuration information must specify which queue manager to connect to, how to connect to the queue manager, which destination to monitor for messages, and the transactional behavior of the MDB. This information is then used by the EJB container. When a message satisfying the selection criteria of the MDB arrives at the specified destination, the EJB container uses IBM MQ classes for JMS to retrieve the message from the queue manager, and then delivers the message to the MDB by calling its `onMessage()` method.

Architecture and overview of features

From IBM WebSphere MQ Version 7.0, IBM MQ classes for JMS contains a number of enhancements compared to previous releases. Some of these enhancements are as a result of changes to the implementation of IBM MQ classes for JMS, and some are as a result of IBM MQ classes for JMS making use of changes to the underlying IBM MQ function.

Layered architecture

Before IBM WebSphere MQ Version 7.0, the implementation of IBM WebSphere MQ classes for JMS was entirely specific to IBM WebSphere MQ. Other IBM products that provided messaging systems also included JMS providers, but these JMS providers had little or nothing in common with the implementation of IBM MQ classes for JMS.

From IBM WebSphere MQ Version 7.0, IBM MQ classes for JMS has had a layered architecture as shown in Figure 54 on page 301. The top layer of code is a common layer that can be used by any IBM JMS provider. When an application calls a JMS method, any processing of the call that is not specific to a messaging system is performed by the common layer, which also provides a consistent response to the call. Any processing of the call that is specific to a messaging system is delegated to a lower layer. In the following diagram, the IBM MQ messaging provider is shown in the lower layer, together with two further messaging providers (Messaging provider A and Messaging provider B.)

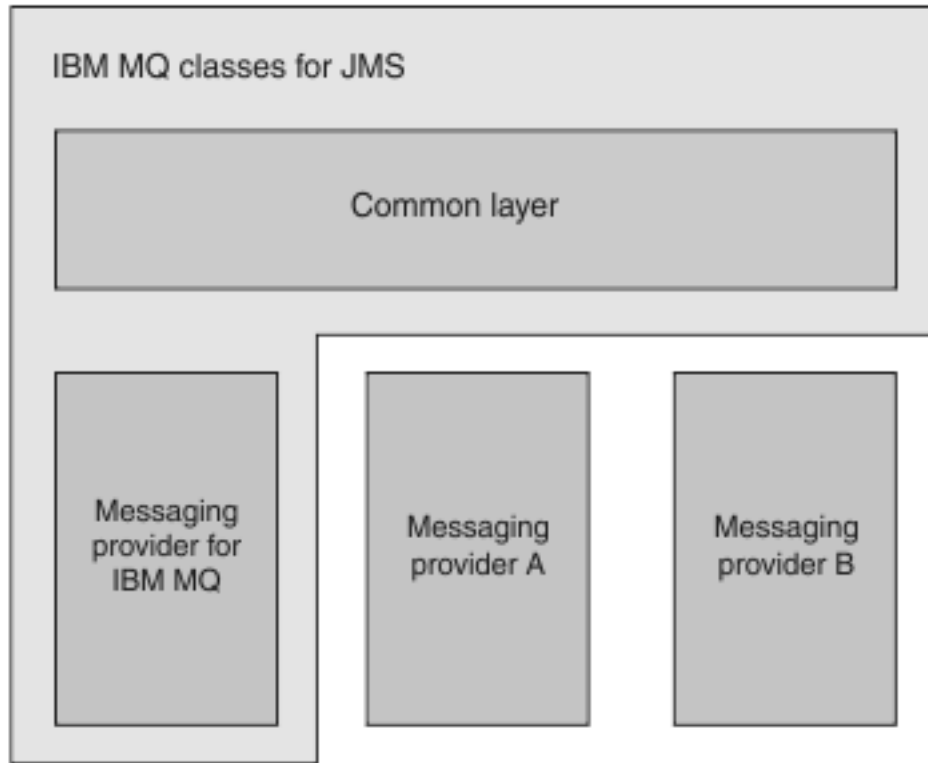


Figure 54. The layered architecture for IBM JMS providers

A layered architecture fulfills following objectives:

- To improve the consistency of behavior of the various IBM JMS providers
- To make it easier to write a bridge application between two IBM messaging systems
- To make it easier to port an application from one IBM JMS provider to another

This implementation of IBM MQ classes for JMS also includes a set of extensions to the JMS API. These extensions are known as the IBM JMS extensions. The main focus of these extensions concerns the creation and configuration of connection factories and destinations dynamically at run time.

An application that uses the IBM JMS extensions starts by creating a `JmsFactoryFactory` object, specifying as a parameter a constant that identifies the chosen messaging system. The application uses the `JmsFactoryFactory` object to create connection factories and destinations that have the correct specialized classes for the chosen messaging system.

The application can then configure the connection factories and destinations by setting their properties. The IBM JMS extensions provide a set of methods to set properties. These methods are independent of any messaging system. Each data type has its own set method, and each property is identified by a name, which is defined as a static final member of the `WMQConstants` class. When an application calls one of these methods, one of the parameters on the call is the name of the property, and the other parameter is the value of the property.

For example, if IBM MQ is the messaging system, one of the properties of a connection factory is the name of the queue manager to connect to. Using the IBM JMS extensions, an application sets the name of the queue manager to JUPITER by calling the following method:

```
JmsConnectionFactory myCF;
...
myCF.setStringProperty(WMQConstants.WMQ_QUEUE_MANAGER, "JUPITER");
```

By contrast, an application can carry out the same function by calling the following method:

```
MQConnectionFactory myCF;  
...  
myCF.setQueueManager("JUPITER");
```

This method is an IBM MQ JMS extension and is specific to IBM MQ as the messaging system. The use of this method might therefore make the application potentially less easy to port to another IBM JMS provider.

Relationship between IBM MQ classes for JMS and IBM MQ classes for Java

Before IBM WebSphere MQ Version 7.0, IBM WebSphere MQ classes for JMS was implemented almost entirely as a layer of code on top of IBM WebSphere MQ classes for Java. This arrangement caused some confusion among application developers because setting fields or calling methods in the `MQEnvironment` class could cause unwanted and unexpected effects on the runtime behavior of code that was written using IBM MQ classes for JMS. In addition, the implementation of IBM MQ classes for JMS had some constraints in areas where the JMS API is not a natural fit on top of IBM MQ classes for Java, which might have led to some issues regarding runtime performance.

From IBM WebSphere MQ Version 7.0, the implementation of IBM MQ classes for JMS is no longer dependent on IBM MQ classes for Java. IBM MQ classes for Java and IBM MQ classes for JMS became peers that use a common Java interface to the MQI. This arrangement allows more scope for optimizing performance, and means that setting fields or calling methods in the `MQEnvironment` class has no effect on the runtime behavior of code that is written using IBM MQ classes for JMS. Figure 55 on page 303 shows the relationship between IBM MQ classes for JMS and IBM MQ classes for Java in previous releases of IBM WebSphere MQ and in IBM WebSphere MQ Version 7.0 and later releases.

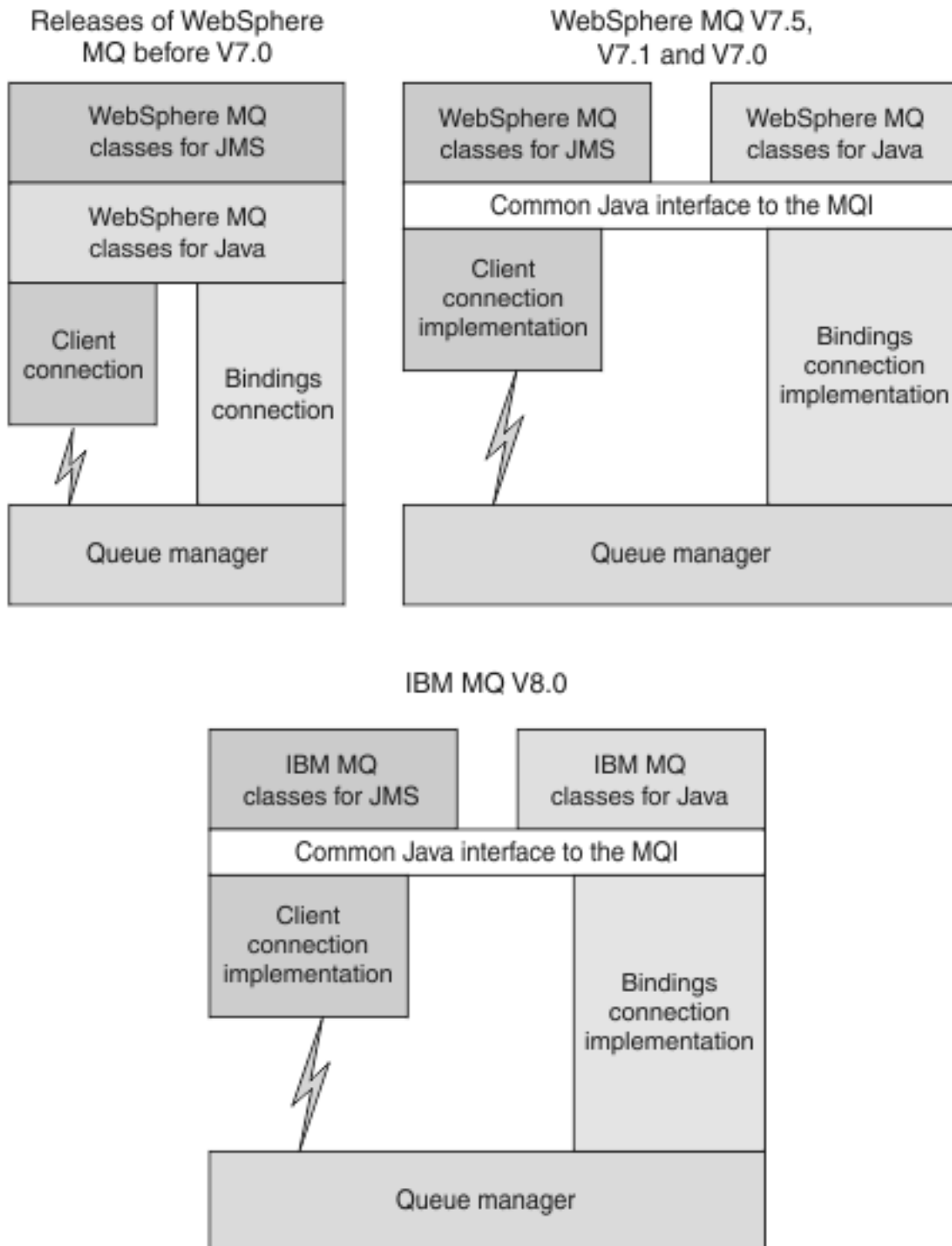


Figure 55. The relationship between IBM MQ classes for JMS and IBM MQ classes for Java

To maintain compatibility with releases before IBM WebSphere MQ Version 7.0, channel exit classes written in Java can still use the IBM MQ classes for Java interfaces, even if the channel exit classes are called from IBM MQ classes for JMS. However, using the IBM MQ classes for Java interfaces means that

your applications are still dependent on the IBM MQ classes for Java JAR file, `com.ibm.mq.jar`. If you do not want `com.ibm.mq.jar` in your class path, you can use the set of interfaces in the `com.ibm.mq.exits` package instead.

From IBM WebSphere MQ Version 7.0, you can create and configure JMS administered objects with the IBM MQ Explorer.

Publish/subscribe messaging

IBM WebSphere MQ Version 7.0, and subsequent releases, contain embedded publish/subscribe function.

IBM MQ classes for JMS applications can use the embedded publish/subscribe function, and can use it instead of using IBM Integration Bus for publish/subscribe messaging with IBM MQ as the transport. Configuring IBM MQ classes for JMS to use the new function is simpler than configuring IBM MQ classes for JMS to use IBM MQ publish/subscribe or IBM Integration Bus. Administrators and application developers no longer need to manage publication queues, subscriber queues, subscription stores, and subscriber cleanup. In addition, `ConnectionFactory` and `Topic` objects have a smaller number of properties.

The embedded publish/subscribe function also provides some additional features such as retained publications and a choice of two wildcard schemes for specifying a range of topics to which an application wants to subscribe.

An application can still use a real-time connection to a broker of IBM Integration Bus for publish/subscribe messaging. This support is unchanged.

Applications that use IBM MQ publish/subscribe can use the embedded publish/subscribe function without change when the queue manager to which they are connected is upgraded. Properties that are set by an application, but are not required by the embedded publish/subscribe function, are ignored.

IBM MQ messaging provider

The IBM MQ messaging provider has three modes of operation:

- IBM MQ messaging provider normal mode
- IBM MQ messaging provider normal mode with restrictions
- IBM MQ messaging provider migration mode

The IBM MQ messaging provider normal mode uses all the features of an IBM MQ queue manager to implement JMS. This mode is optimized to use the JMS 2.0 API and functionality.

The IBM MQ messaging provider normal mode with restrictions uses the JMS 2.0 API, but not the new IBM MQ Version 8.0 features such as shared subscriptions, delayed delivery, or asynchronous send.

The IBM MQ messaging provider migration mode is based on IBM WebSphere MQ Version 6.0 function and uses only features that were available in the IBM WebSphere MQ Version 6.0 queue manager to implement JMS. You can connect to IBM WebSphere MQ Version 7.0, or later, queue managers that use IBM MQ messaging provider migration mode but you cannot use any of the Version 7.0 optimizations. This mode allows connections to either of the following queue manager versions:

1. IBM WebSphere MQ Version 7.0 or later, queue manager in bindings or client mode, but this mode uses only those features that were available to a IBM WebSphere MQ Version 6.0 queue manager.
2. IBM WebSphere MQ Version 6.0 or earlier queue manager in client mode.

If you want to connect to IBM Integration Bus by using IBM MQ Enterprise Transport, use the IBM MQ messaging provider migration mode. If you use IBM MQ Real-Time Transport, the IBM MQ messaging provider migration mode is automatically selected because you have explicitly selected properties in the

connection factory object. Connection to IBM Integration Bus using the IBM MQ Enterprise Transport follows the general rules for mode selection that is described in Configuring the JMSPROVIDERVERSION property.

Asynchronous message consumption

Asynchronous message consumption has been supported since IBM WebSphere MQ Version 7.0. An application can register a callback function for a destination. When a suitable message is sent to the destination, IBM MQ calls the function and passes the message as a parameter. The function then processes the message asynchronously. In releases of IBM WebSphere MQ before Version 7.0, this feature was available only when using IBM WebSphere MQ classes for JMS.

IBM MQ classes for JMS was changed to use this feature in IBM WebSphere MQ Version 7.0. Since IBM WebSphere MQ Version 7.0, the implementation of JMS message listeners is a more natural fit with IBM MQ, and IBM MQ classes for JMS no longer has to poll a destination to check whether a suitable message has been sent to the destination. The performance of JMS message listeners is improved as a result, particularly when an application uses multiple message listeners in a session to monitor multiple destinations. Message throughput is increased, and the time that is taken to deliver a message to a message listener after it has arrived at a destination is reduced.

Message driven beans (MDBs) have similar performance improvements. In addition, because of another enhancement to IBM MQ function, multiple MDBs that are consuming messages from the same destination now experience reduced contention on the messages.

Message selection

Except for selecting messages by message identifier or correlation identifier, all message selection in releases of IBM MQ prior to Version 7.0 was done by IBM MQ classes for JMS. In IBM WebSphere MQ Version 7.0 and later, all message selection is done by the queue manager.

As a result, message throughput is increased for applications that consume messages using message selection. The performance improvement is greater for an application that connects in client mode because only those messages that satisfy the selection criteria are transported over the network, and IBM MQ classes for JMS handles only those messages that it delivers to the application.

Sharing a communications connection

Before IBM WebSphere MQ Version 7.0, if an IBM MQ client application connected to a queue manager more than once using the same MQI channel, each instance of the MQI channel required a separate TCP connection. In IBM WebSphere MQ Version 7.0 and later, each connection to the queue manager using the same MQI channel can share a single TCP connection. This arrangement means that fewer network resources are required and the total time that is taken to create multiple connections to the queue manager is reduced, particularly when using TLS because the TLS handshake takes place only once at the start of the TCP connection.

IBM MQ classes for JMS uses this enhancement. For an application that connects to a queue manager in client mode, IBM MQ classes for JMS might create more than one connection to a queue manager using the MQI channel with the name that is specified as a property of the ConnectionFactory object. Each of these connections to the queue manager can now share a single TCP connection.

Read ahead on client connections

If an application uses a client connection to consume nonpersistent messages from a destination, the destination can be configured so that IBM MQ classes for JMS uses a buffer to store the messages of interest before delivering them to the application. This optimization is called *read ahead* and can be used by applications that consume messages synchronously by calling the `receive()` method, and by message

listeners and MDBs, which consume messages asynchronously. Read ahead is particularly effective for destinations with a large number of messages that need to be consumed rapidly.

Read ahead does not apply to persistent messages because, if persistent messages were read into a buffer, the queue manager would no longer be able to recover the messages following a failure. However, an application that consumes messages from a destination with a mixture of persistent and nonpersistent messages can still use read ahead. The order of the messages is preserved, but the runtime benefits of read ahead apply only to the nonpersistent messages.

When you are deciding whether to use read ahead, consider the following points:

- If an application is consuming messages from a destination that is configured for read ahead, and the application ends for any reason, any nonpersistent messages that are currently stored in the buffer are discarded.
- If all the following conditions are true, messages sent to a queue in a session might not be received in the order in which they were sent:
 - An application uses two message consumers in the same session to consume the messages from the queue.
 - Each message consumer uses a different Destination object for the queue.
 - Any or both of the Destination objects are configured for read ahead.

Sending messages

When an application sends messages to a destination, the destination can be configured so that when the application calls `send()`, IBM MQ classes for JMS forwards the message to the queue manager and returns control back to the application without determining whether the queue manager has received the message safely. IBM MQ classes for JMS can work in this way only for nonpersistent messages and for persistent messages that are sent in a transacted session.

For persistent messages sent in a transacted session, the application ultimately determines whether the queue manager has received the messages safely when it calls `commit()`. For any messages sent in a session that is not transacted, the `SENDCHECKCOUNT` property of the `ConnectionFactory` object specifies how many messages are to be sent before IBM MQ classes for JMS checks that the queue manager has received the messages safely.

This optimization is of most benefit to an application that connects to a queue manager in client mode and needs to send a sequence of messages in rapid succession, but does not require immediate feedback from the queue manager for each message sent.

Channel exits

From IBM WebSphere MQ Version 7.0, when called from IBM MQ classes for JMS, channel exit programs that are written in C or C++ behave in the same way as when they are called from an IBM MQ MQI client. The performance of channel exit classes that are written in Java is improved from earlier releases, and you can write channel exit classes using a new set of interfaces in the `com.ibm.mq.exits` package instead of using the interfaces in IBM MQ classes for Java.

Message properties

A JMS message consists of a set of header fields, a set of properties, and a body that contains the application data. As a minimum, an IBM MQ message consists of a message descriptor and the application data.

When an IBM MQ classes for JMS application sends a JMS message, IBM MQ classes for JMS maps the JMS message into an IBM MQ message. Some of the JMS header fields and properties are mapped into

fields in the message descriptor, and some are mapped into fields in an additional IBM MQ header that is called an MQRFH2 header. When an IBM MQ classes for JMS application receives a JMS message, IBM MQ classes for JMS performs the reverse mapping.

An application that is using the MQI to receive messages from an IBM MQ classes for JMS application must therefore be able to handle an MQRFH2 header. If the application cannot handle an MQRFH2 header, the TARGCLIENT property of the Destination object can be set to tell IBM MQ classes for JMS not to include an MQRFH2 header in the IBM MQ messages. However, by excluding the MQRFH2 header, the information that is held in some of the JMS header fields and properties is lost.

Similarly, an application that is using the MQI to send messages to an IBM MQ classes for JMS application must include an MQRFH2 header in each message. If an MQRFH2 header is not included, IBM MQ classes for JMS can set only those JMS header fields and properties that can be derived from the fields in a message descriptor.

IBM WebSphere MQ Version 7.0 and later provides some additional support for applications that use the MQI to receive messages from, and send messages to, IBM MQ classes for JMS applications.

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:

1. The message is delivered with a message descriptor, an MQRFH2 header that contains data that is derived from JMS header fields and properties, and the application data.
2. The message is delivered with a message descriptor, the application data, and a set of message properties.

In option 2, each message property represents a JMS header field or property that was originally mapped by IBM MQ classes for JMS into a field in an MQRFH2 header. After the MQGET call, the application can use the MQINQMP call to get the values of the message properties. Using option 2 instead of option 1 to receive a message simplifies the application logic in the following ways:

- The application does not have to parse the variable portion of the MQRFH2 header, which contains the JMS header field and property data that is encoded in an XML-like format.
- The application does not have to convert the character data in the variable portion of the MQRFH2 header.

Correspondingly, before an application calls MQPUT to send a message to an IBM MQ classes for JMS application, the application can use the MQSETMP call to set the values of message properties instead of constructing an MQRFH2 header.

Serviceability

From IBM WebSphere MQ Version 7.0, IBM MQ classes for JMS contains a number of improvements that are related to serviceability.

Tracing

IBM MQ classes for JMS contains a class that an application can use to control tracing. An application can start and stop tracing, specify the required level of detail in a trace, and customize trace output in various ways.

Logging

IBM MQ classes for JMS maintains a log file, which contains messages about errors that you need to correct. The messages are written in plain text. IBM MQ classes for JMS contains a class that an application can use to specify the location of the log file and its maximum size.

First Failure Support Technology (FFST)

If a serious failure occurs, IBM MQ classes for JMS generates an FFST report in an FDC file. The FFST report contains information that IBM Service can use to diagnose the problem more quickly.

Version information

IBM MQ classes for JMS contains a class that an application can use to query the version of IBM MQ classes for JMS.

Exception messages

Exception messages have been enhanced to provide more information about the causes of errors and the actions that are required to correct errors.

Application servers

The integration of the serviceability features of IBM MQ classes for JMS with the serviceability features of WebSphere Application Server was improved.

MQC is replaced by MQConstants

From IBM WebSphere MQ Version 7.0, the `com.ibm.mq.constants` package replaces the MQC interface. This package contains the class MQConstants, which implements a number of interfaces. MQConstants contains definitions of all the constants that were in the MQC interface and a number of new constants. The interfaces in this package closely follow the names of the constants header files that are used in IBM MQ.

For example, the interface CMQC contains a constant MQOO_INPUT_SHARED; this interface and constant correspond to the header file `cmqc.h` and the constant MQOO_INPUT_SHARED.

`com.ibm.mq.constants` can be used with both IBM MQ classes for Java and IBM MQ classes for JMS.

MQC is still present, and has the constants it previously had. However, for any new applications, you must use the `com.ibm.mq.constants` package.

Related information:

IBM MQ messaging provider modes of operation

Exceptions in IBM MQ classes for JMS

ASYNCEXCEPTION

IBM MQ client for HP Integrity NonStop Server

► NSS Client

IBM MQ now provides a client to support the HP Integrity NonStop Server platform.

Overview

For an overview of IBM MQ clients, including the client for the HP Integrity NonStop Server platform, see “Overview of IBM MQ MQI clients” on page 287.

For details specific to the IBM MQ client for HP Integrity NonStop Server, see “IBM MQ client for HP Integrity NonStop Server technical overview” on page 313, and “IBM MQ client for HP Integrity NonStop Server supported environments and features” on page 313.

Planning

See Planning your IBM MQ client environment on HP Integrity NonStop Server.

Installing

Help about installing the IBM MQ client for HP Integrity NonStop Server.

- Choosing what to install, see IBM MQ client components for HP Integrity NonStop Server.

- Planning your installation, see [Planning your installation on HP Integrity NonStop Server](#)
 - File system
- Hardware and software requirements, see [Hardware and software requirements on HP Integrity NonStop Server systems](#).
- Verifying that you have the correct software, see [Verifying system software prerequisites](#).
- Preparing your system, see [Setting up the user and group on HP Integrity NonStop Server](#).
- Installing the client, see [Installing the IBM MQ client on HP Integrity NonStop Server](#).
- Verifying your installation, see [Testing communication between a client and a server on HP Integrity NonStop Server](#).
- Uninstalling, see [Uninstalling the IBM MQ client for HP Integrity NonStop Server](#).

Commands

The following control commands are applicable to the IBM MQ client for HP Integrity NonStop Server OSS and Guardian environments:

- dspmqtrc
- dspmqver
- endmqtrc
- mqrc
- runmqras
- runmqsc
- runmqtmc
- strmqtrc

The following control command is applicable to the IBM MQ client for HP Integrity NonStop Server OSS environment:

- dspmqtrc

The following MQSC command includes a Product Identifier MQNC for the IBM MQ client for HP Integrity NonStop Server:

- DISPLAY CHSTATUS (see the Product Identifier values table)

The following Programmable Command Formats are applicable to the IBM MQ client for HP Integrity NonStop Server:

- Change, Copy, and Create Authentication Information Object
- Change, Copy, and Create Channel
- Change, Copy, and Create Namelist
- Change, Copy, and Create Process
- Change, Copy, and Create Queue
- Change Queue Manager
- Clear Queue
- Delete Authentication Information Object
- Delete Channel
- Delete Namelist
- Delete Process
- Delete Queue
- Escape
- Escape (Response)

- Inquire Authentication Information Object
- Inquire Authentication Information Object (Response)
- Inquire Authentication Information Object Names
- Inquire Authentication Information Object Names (Response)
- Inquire Channel
- Inquire Channel (Response)
- Inquire Channel Names
- Inquire Channel Names (Response)
- Inquire Channel Status
- Inquire Channel Status (Response)
- Inquire Cluster Queue Manager
- Inquire Cluster Queue Manager (Response)
- Inquire Namelist
- Inquire Namelist (Response)
- Inquire Namelist Names
- Inquire Namelist Names (Response)
- Inquire Process
- Inquire Process (Response)
- Inquire Process Names
- Inquire Process Names (Response)
- Inquire Queue
- Inquire Queue (Response)
- Inquire Queue Manager
- Inquire Queue Manager (Response)
- Inquire Queue Names
- Inquire Queue Names (Response)
- Ping Channel
- Ping Queue Manager
- Refresh Cluster
- Refresh Queue Manager
- Reset Channel
- Reset Cluster
- Reset Queue Manager
- Reset Queue Statistics
- Reset Queue Statistics (Response)
- Resolve Channel
- Resume Queue Manager Cluster
- Start Channel
- Stop Channel
- Suspend Queue Manager Cluster

Security

To secure your IBM MQ client for HP Integrity NonStop Server environment, see:

- Information about how the IBM MQ client for HP Integrity NonStop Server identifies itself to the queue manager added to Planning authentication for a client application.

- Setting up security on HP Integrity NonStop Server
 - OpenSSL
 - Entropy Daemon
- TLS security protocols in IBM MQ
- Working with TLS on HP Integrity NonStop Server
 - Certificate management
 - Personal certificate store
 - Certificate truststore
 - Pass phrase stash file
 - Certificate revocation list file

Transaction Management Facility

For information about the Transaction Management Facility (TMF), refer to the following sections and topics.

- Planning your IBM MQ client environment on HP Integrity NonStop Server
 - Preparing the HP Integrity NonStop Server environment
 - IBM MQ and HP NonStop TMF
 - Using HP NonStop TMF
 - Using global units of work
 - Avoiding long running transactions
 - Information about queue manager configuration to expire global units of work after a pre-configured interval of inactivity added to Expiring global units of work.
- Configuring IBM MQ client for HP Integrity NonStop Server
 - Gateway process overview
 - Configuring Gateway to run under Pathway
 - TMF and TMF/Gateway stanzas
 - Configuring the client initialization file
 - Granting permissions to channels
- Administering IBM MQ client for HP Integrity NonStop Server
 - Manually starting the TMF/Gateway from Pathway
 - Stopping the TMF/Gateway from Pathway
- IBM MQ client for HP Integrity NonStop Server troubleshooting

Developing applications

For information about developing applications for your IBM MQ client on the HP Integrity NonStop Server platform, see:

- Building your procedural application on HP Integrity NonStop Server
 - OSS and Guardian headers and public libraries
 - Preparing C programs in HP Integrity NonStop Server
 - Preparing COBOL programs
 - Preparing pTAL programs
- For information about coding in pTAL, see Coding in pTAL.
- For information about preparing JMS programs for the IBM MQ client for HP Integrity NonStop Server, see Preparing JMS programs for the IBM MQ client for HP Integrity NonStop Server.

New messages

The following messages are new for the IBM MQ client for HP Integrity NonStop Server:

- AMQ5xxx: Installable services
 - AMQ5370
 - AMQ5371
 - AMQ5372
 - AMQ5373
 - AMQ5374
 - AMQ5375
 - AMQ5376
 - AMQ5377
 - AMQ5378
 - AMQ5379
 - AMQ5380
 - AMQ5390
 - AMQ5391
 - AMQ5392
 - AMQ5393
 - AMQ5394
 - AMQ5395
 - AMQ5396
 - AMQ5397
 - AMQ5398
 - AMQ5399
- AMQ9xxx: Remote
 - AMQ9816
 - AMQ9817
 - AMQ9818
 - AMQ9819
 - AMQ9820
 - AMQ9821
 - AMQ9823
 - AMQ9824

Modified API reason codes

The following existing API reason codes now include HP Integrity NonStop Server:

- 2354 (0932) (RC2354): MQRC_UOW_ENLISTMENT_ERROR
- 2355 (0933) (RC2355): MQRC_UOW_MIX_NOT_SUPPORTED
- 2072 (0818) (RC2072): MQRC_SYNCPOINT_NOT_AVAILABLE
- 2003 (07D3) (RC2003): MQRC_BACKED_OUT

Samples

For information about the techniques demonstrated by the sample programs for the IBM MQ client for HP Integrity NonStop Server, see [Samples for IBM MQ client for HP Integrity NonStop Server](#).

Troubleshooting and support

For troubleshooting and support information for the IBM MQ client for HP Integrity NonStop Server, see the following topics:

- Troubleshooting IBM MQ client for HP Integrity NonStop Server
- Error logs on HP Integrity NonStop Server
- Using trace on HP Integrity NonStop Server
- FFST: IBM MQ client for HP Integrity NonStop Server

IBM MQ client for HP Integrity NonStop Server technical overview

► NSS Client

Describes where you can find the IBM MQ client for HP Integrity NonStop Server SupportPac, and gives a technical overview of the HP Integrity NonStop Server operating system.

Technical overview of the HP Integrity NonStop Server operating system

The HP Integrity NonStop Server is an operating system that is designed for the highest possible availability, with no planned, or unplanned downtime even with multiple hardware or software failures. It is linearly scalable, for example, if you add 20 percent more hardware, you get 20 percent more usable performance. To maintain data integrity, the operating system has its own transaction manager, and a transactional file system.

The HP Integrity NonStop Server operating system is typically used by:

- Financial institutions, for example, for ATM networks, online banking support, credit authorizations, stock exchange switches, trading, and bank to bank transactions.
- Manufacturing, for example, for web store back ends, inventory, and process control.
- Telecommunications, for example, for exchanges, emergency, and other network services.

For more information about HP Integrity NonStop Server, see HP NonStop systems.

IBM MQ client for HP Integrity NonStop Server supported environments and features

► NSS Client

Provides details about the IBM MQ client for HP Integrity NonStop Server, and describes supported client API and environments and client functionality specific to HP Integrity NonStop Server systems.

Supported client API and environments

IBM MQ client for HP Integrity NonStop Server supports the following execution environments:

Table 21.

	OSS	Guardian
C	✓	✓
JMS	✓	
COBOL	✓	✓

Table 21. (continued)

	OSS	Guardian
pTAL	✓	✓

Functional summary

Some aspects of client functionality are specific to the host operating system. The following summary describes the aspects of client functionality specific to the IBM MQ client for HP Integrity NonStop Server:

- C (native), PTAL, COBOL (native)
 - Network Protocol: TCP (IPv4 and IPv6)
 - Transport Type: Client only
 - Transport Security: TLS
 - Transactional Support: Two-phase commit coordinated by the Transaction Management Facility (TMF) (requires connection to a queue manager that is at IBM WebSphere MQ Version 7.1 or later)
 - Addressing mode: 32 bit
- Java Message Service (JMS)
 - Network Protocol: TCP (IPv4 and IPv6)
 - Transport Type: Client only (Bindings, Direct, and Direct HTTP are not supported)
 - Transport Security: TLS
 - Transactional Support: Single-phase commit
 - Execution: Standalone (Application Support Facility (ASF) and Java Connector Architecture (JCA) are not supported)
 - Exits: Java language only (native exits that are written in other languages are not supported)
 - IBM MQ Headers and PCF: The following classes are not supported: `com.ibm.mq.headers.*` and `com.ibm.mq.pcf.*`

IBM MQ for z/OS concepts

▶ z/OS

Some of the concepts used by IBM MQ for z/OS are unique to the z/OS platform. For example, the logging mechanism, the storage management techniques, unit of recovery disposition, and queue-sharing groups are provided only with IBM MQ for z/OS. Use this topic for further information about these concepts.

Related concepts:

“The queue manager on z/OS”

Before you can let your application programs use IBM MQ on your z/OS system, you must install the IBM MQ for z/OS product and start a queue manager. The queue manager owns and manages the set of resources that are used by IBM MQ.

► z/OS “The channel initiator on z/OS” on page 317

The channel initiator provides and manages resources that enable IBM MQ distributed queuing. IBM MQ uses *Message Channel Agents* (MCAs) to send messages from one queue manager to another.

“Terms and tasks for managing IBM MQ for z/OS” on page 320

Use this topic as an introduction to the terminology, and tasks that are specific to IBM MQ for z/OS.

“Shared queues and queue-sharing groups” on page 322

You can use shared queues and queue-sharing groups, to implement high availability of IBM MQ resources. Shared queues and queue-sharing groups are functions unique to IBM MQ for z/OS on the z/OS platform.

“Intra-group queuing” on page 369

This section describes intra-group queuing, an IBM MQ for z/OS function unique to the z/OS platform. This function is only available to queue managers defined to a queue-sharing group.

“Storage management on z/OS” on page 384

IBM MQ for z/OS requires permanent and temporary data structures and uses page sets and memory buffers to store this data. These topics give more details on how IBM MQ uses these page sets and buffers.

“Logging in IBM MQ for z/OS” on page 390

IBM MQ maintains *logs* of data changes and significant events as they occur. These logs can be used to recover data to a previous state if required.

“Recovery and restart on z/OS” on page 412

Use the links in this topic to find out about the features of IBM MQ for z/OS for restart and recovery.

“Security concepts in IBM MQ for z/OS” on page 429

Use this topic to understand the importance of security for IBM MQ, and the implications of not having adequate security settings on your system.

“Availability on z/OS” on page 436

IBM MQ for z/OS has many features for high availability. This topic describes some of the considerations for availability.

“Unit of recovery disposition on z/OS” on page 440

Certain transactional applications can use a GROUP, rather than a QMGR, unit of recovery disposition when connected to a queue manager in a queue-sharing group (QSG) by specifying the QSG name when they connect instead of the queue manager name. This allows transaction recovery to be more flexible and robust by removing the requirement to reconnect to the same queue manager in the QSG.

Related reference:

“Defining your system on z/OS” on page 401

IBM MQ for z/OS uses many default object definitions, and provides sample JCL to create those default objects. Use this topic to understand these default objects, and the sample JCL.

“Monitoring and statistics on IBM MQ for z/OS” on page 439

IBM MQ for z/OS has a set of facilities for monitoring the queue manager, and gathering statistics.

The queue manager on z/OS

► z/OS

Before you can let your application programs use IBM MQ on your z/OS system, you must install the IBM MQ for z/OS product and start a queue manager. The queue manager owns and manages the set of resources that are used by IBM MQ.

The queue manager

A *queue manager* is a program that provides messaging services to applications. Applications that use the Message Queue Interface (MQI) can put messages on queues and get messages from queues. The queue manager ensures that messages are sent to the correct queue or are routed to another queue manager. The queue manager processes both the MQI calls that are issued to it, and the commands that are submitted to it (from whatever source). The queue manager generates the appropriate completion codes for each call or command.

The resources managed by the queue manager include:

- Page sets that hold the IBM MQ object definitions and message data
- Logs that are used to recover messages and objects in the event of queue manager failure
- Processor storage
- Connections through which different application environments (CICS, IMS, and Batch) can access the IBM MQ API
- The IBM MQ channel initiator, which allows communication between IBM MQ on your z/OS system and other systems

The queue manager has a name, and applications can connect to it using this name.

Figure 56 illustrates a queue manager, showing connections to different application environments, and the channel initiator.

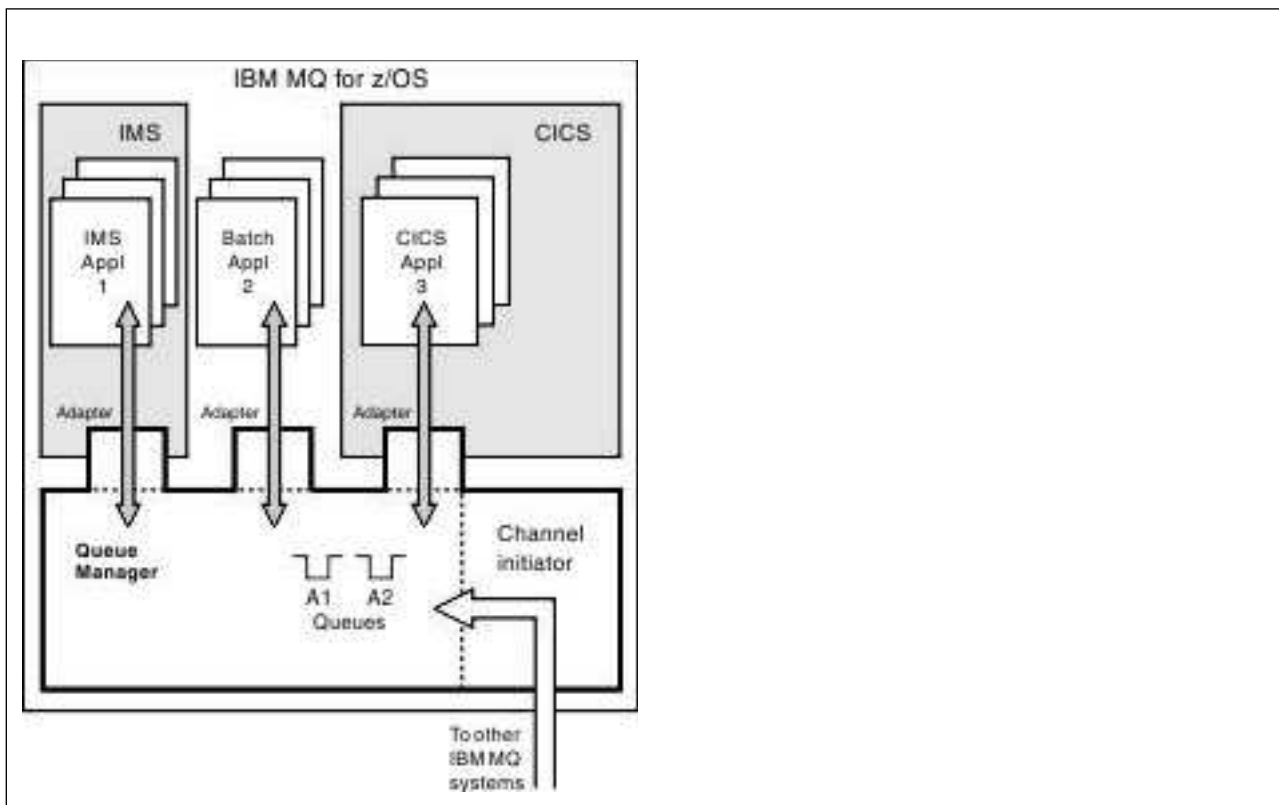


Figure 56. Overview of IBM MQ for z/OS

The queue manager subsystem on z/OS

On z/OS, IBM MQ runs as a z/OS subsystem that is started at IPL time. In the subsystem, the queue manager is started by executing a JCL procedure that specifies the z/OS data sets that contain

information about the logs, and that hold object definitions and message data (the page sets). The subsystem and the queue manager have the same name, of up to four characters. All queue managers in your network must have unique names, even if they are on different systems, sysplexes, or platforms.

The channel initiator on z/OS



The channel initiator provides and manages resources that enable IBM MQ distributed queuing. IBM MQ uses *Message Channel Agents* (MCAs) to send messages from one queue manager to another.

To send messages from queue manager A to queue manager B, a *sending* MCA on queue manager A must set up a communications link to queue manager B. A *receiving* MCA must be started on queue manager B to receive messages from the communications link. This one-way path consisting of the sending MCA, the communications link, and the receiving MCA is known as a *channel*. The sending MCA takes messages from a transmission queue and sends them down a channel to the receiving MCA. The receiving MCA receives the messages and puts them on to the destination queues.

In IBM MQ for z/OS, the sending and receiving MCAs all run inside the channel initiator (the channel initiator is also known as the *mover*). The channel initiator runs as a z/OS address space under the control of the queue manager. There can be only a single channel initiator connected to a queue manager and it is run inside the same z/OS image as the queue manager. There can be thousands of MCA processes running inside the channel initiator concurrently.

Figure 57 on page 318 shows two queue managers within a sysplex. Each queue manager has a channel initiator and a local queue. Messages sent by queue managers on AIX and Windows are placed on the local queue, from where they are retrieved by an application. Reply messages are returned by a similar route.

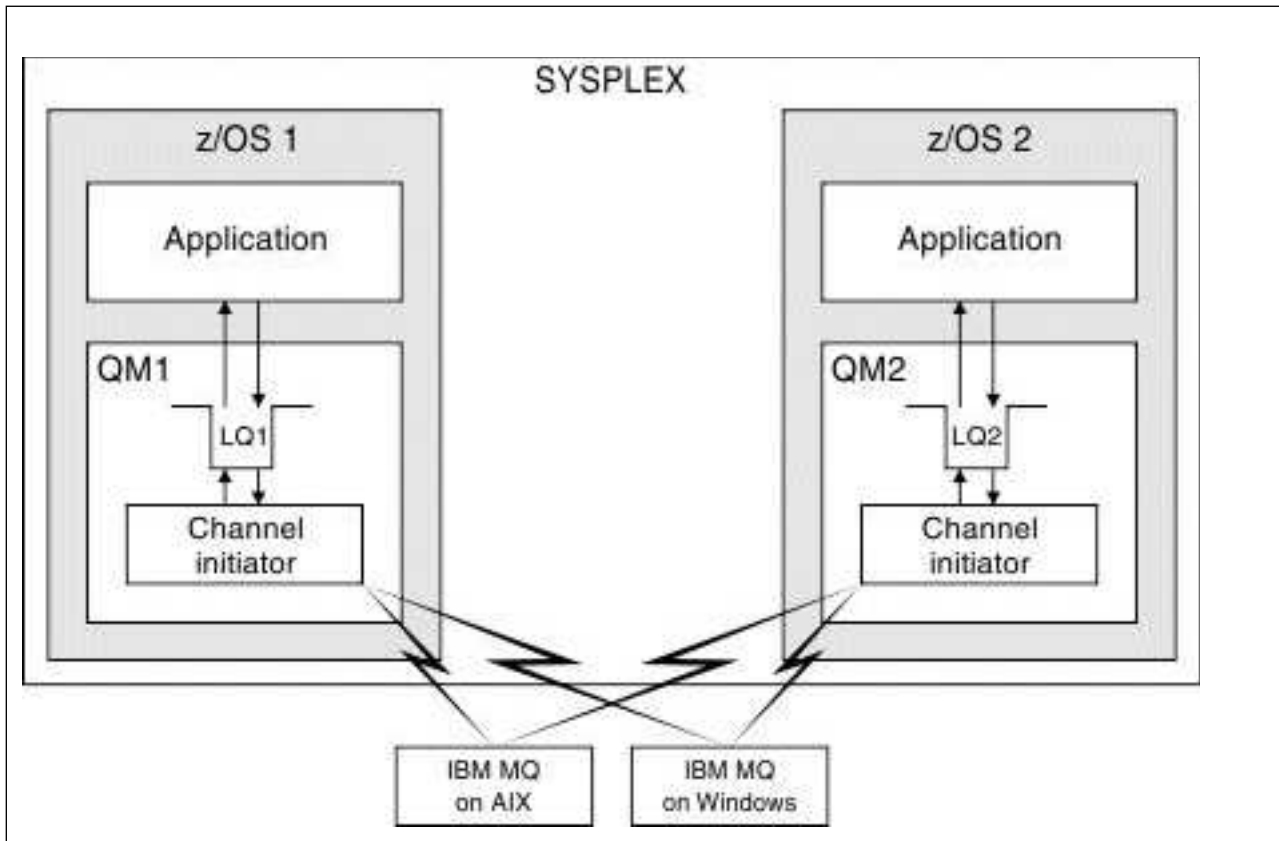


Figure 57. Communication between queue managers

The channel initiator also contains other processes concerned with the management of the channels. These processes include:

Listeners

These processes listen for inbound channel requests on a communications subsystem such as TCP, and start a named MCA when an inbound request is received.

Supervisor

This manages the channel initiator address space, for example it is responsible for restarting channels after a failure.

Name server

This is used to resolve TCP names into addresses.

TLS tasks


These are used to perform encryption and decryption and check certificate revocation lists.

SMF records for the channel initiator

The channel initiator (CHINIT) can produce SMF statistics records and accounting records with information on tasks and channels.

The CHINIT can produce SMF statistics records and accounting records with the following types of information:

- The tasks: dispatcher, adapter, Domain Name Server (DNS), and SSL. These tasks form what is called CHINIT statistics.
- Channels: provides accounting information similar to that available with the DIS CHSTATUS command. This is called channel accounting.

 IBM MQ for Multiplatforms provides similar information by writing PCF messages to the SYSTEM.ADMIN.STATISTICS.QUEUE. See Channel statistics message data for further information on how statistics information is recorded on Multiplatforms.

Statistics data

You can use this information to find out the following information:

- Whether you need more of the CHINIT tasks, such as number of SSL TCBs and how much CPU is used by these tasks.
- The average time for requests on these tasks.
- The longest duration request in the interval, and the time of day this occurred, for DNS and SSL tasks. You can correlate this time of day with problems you may experience with the channel.

Accounting data

You can use this information to monitor channel usage and find out the following information:

- The channels with the highest throughput.
- The rate at which messages were sent, and the rate of sending data in MB/second.
- The achieved batch size. If the achieved batch size is close to the batch size specified for the channel, the channel might be close to its limit for sending messages.

You use the START TRACE and STOP TRACE commands to control the collection of the accounting trace and the statistics trace. You can use the STATCHL and STATACLS options on the channel and queue manager to control whether channels produce SMF data.

Note: You do not have to set OPMODE to NEWFUNC in order to enable the collection of SMF records for the channel initiator.

Terms and tasks for managing IBM MQ for z/OS

z/OS

Use this topic as an introduction to the terminology, and tasks that are specific to IBM MQ for z/OS.

Some of the terms and tasks required for managing IBM MQ for z/OS are specific to the z/OS platform. The following list contains some of these terms and tasks.

- Shared queues
- Page sets and buffer pools
- Logging
- Tailoring the queue manager environment
- Restart and recovery
- Security
- Availability
- Manipulating objects
- Monitoring and statistics
- Application environments

Shared queues

Queues can be *non-shared*, owned by and accessible to only one queue manager, or *shared*, owned by a *queue-sharing group*. A queue-sharing group consists of a number of queue managers, running within a single z/OS sysplex, that can access the same IBM MQ object definitions and message data concurrently. Within a queue-sharing group, the shareable object definitions are stored in a shared Db2 database. The shared queue messages are held inside one or more coupling facility structures (CF structures). If the message data is too large to store directly in the structure (more than 63 KB in size), or if the message is large enough that installation-defined rules select it for offloading, the message control information is still stored in the coupling facility entry, but the message data is offloaded to a shared message data set (SMDS) or to a shared Db2 database. The shared message data sets, the shared Db2 database, and the coupling facility structures are resources that are jointly managed by all of the queue managers in the group.

Pages sets and buffer pools

When a message is put on to a non-shared queue, the queue manager stores the data on a page set in such a way that it can be retrieved when a subsequent operation gets a message from the same queue. If the message is removed from the queue, space in the page set that holds the data is later freed for reuse. As the number of messages held on a queue increases, so the amount of space used in the page set increases, and as the number of messages on a queue reduces, the space used in the page set reduces.

To reduce the performance cost of writing data to and reading data from the page sets, the queue manager buffers the updates into processor storage. The amount of storage used to buffer the page set access is controlled through IBM MQ objects called *buffer pools*.

For more information about page sets and buffer pools, see [Storage management](#).

Logging

Any changes to objects held on page sets, and operations on persistent messages, are recorded as log records. These log records are written to a log data set called the *active log*. The name and size of the active log data set is held in a data set called the *bootstrap data set* (BSDS).

When the active log data set fills up, the queue manager switches to another log data set so that logging can continue, and copies the content of the full active log data set to an *archive log* data set. Information about these actions, including the name of the archive log data set, is held in the bootstrap data set. Conceptually, there is a ring of active log data sets that the queue manager cycles through; when an active log is filled, the log data is offloaded to an archive log, and the active log data set is available for reuse.

For more information about the log and bootstrap data sets, see “Logging in IBM MQ for z/OS” on page 390.

Tailoring the queue manager environment

When the queue manager is started, a set of initialization parameters that control how the queue manager operates are read. In addition, data sets containing IBM MQ commands are read, and the commands they contain are executed. Typically, these data sets contain definitions of the system objects required for IBM MQ to run, and you can tailor these to define or initialize the IBM MQ objects necessary for your operating environment. When these data sets have been read, any objects defined by them are stored, either on a page set or in Db2.

For more information about initialization parameters and system objects, see “Defining your system on z/OS” on page 401.

Recovery and restart

At any time during the operation of IBM MQ, there might be changes held in processor storage that have not yet been written to the page set. These changes are written out to the page set that is the least recently used by a background task within the queue manager.

If the queue manager terminates abnormally, the recovery phase of queue manager restart can recover the lost page set changes because persistent message data is held in log records. This means that IBM MQ can recover persistent message data and object changes right up to the point of failure.

If a queue manager that is a member of a queue-sharing group encounters a coupling facility failure, the persistent messages on that queue can be recovered only if you have backed up your coupling facility structure.

For more information about recovery and restart, see “Recovery and restart on z/OS” on page 412.

Security

You can use an external security manager, such as Security Server (previously known as RACF) to protect the resources that IBM MQ owns and manages from access by unauthorized users. You can also use Transport Layer Security (TLS) for channel security. TLS is included as part of the IBM MQ product.

For more information about IBM MQ security, see “Security concepts in IBM MQ for z/OS” on page 429.

Availability

There are several features of IBM MQ that are designed to increase system availability in the event of queue manager or communications subsystem failure. For more information about these features, see “Availability on z/OS” on page 436.

Manipulating objects

When the queue manager is running, you can manipulate IBM MQ objects either through a z/OS console interface, or through an administration utility that uses ISPF services under TSO. Both mechanisms enable you to define, alter, or delete IBM MQ objects. You can also control and display the status of various IBM MQ and queue manager functions.

You can also manipulate IBM MQ objects using the IBM MQ Version 7 Explorer, a graphical user interface that provides a visual way of working with queues, queue managers, and other objects.

For more information about these facilities, see *Issuing commands*.

Monitoring and statistics

Several facilities are available to monitor your queue managers and channel initiators. You can also collect statistics for performance evaluation and accounting purposes.

For more information about these facilities, see “Monitoring and statistics on IBM MQ for z/OS” on page 439.

Application environments

When the queue manager has started, applications can connect to it and start using the IBM MQ API. These can be CICS, IMS, Batch, or WebSphere Application Server applications. IBM MQ applications can also access applications on CICS and IMS systems that are not aware of IBM MQ, using the CICS and IMS bridges.

For more information about these facilities, see “IBM MQ and other z/OS products” on page 443.

For information about writing IBM MQ applications, see the following documentation:

- Developing applications
- Using C++
- Using IBM MQ classes for Java

Shared queues and queue-sharing groups

▶ z/OS

You can use shared queues and queue-sharing groups, to implement high availability of IBM MQ resources. Shared queues and queue-sharing groups are functions unique to IBM MQ for z/OS on the z/OS platform.

This section describes the attributes and benefits, and offers information about how several queue managers can share the same queues and the messages on those queues.

Related concepts:

“What is a shared queue?”

A shared queue is a type of local queue. The messages on that queue can be accessed by one or more queue managers that are in a sysplex.

“What is a queue-sharing group?” on page 325

A group of queue managers that can access the same shared queues is called a queue-sharing group. Each member of the queue-sharing group has access to the same set of shared queues.

“Where are shared queue messages held?” on page 326

Each message on a shared queue is represented by an entry in a z/OS coupling facility list structure. If the message data is too large to fit in the same entry, it is offloaded either to a shared message data set (SMDS) or to Db2.

“Advantages of using shared queues” on page 343

Shared queue allows for IBM MQ applications to be scalable, highly available, and allows workload balancing to be implemented.

“Distributed queuing and queue-sharing groups” on page 363

Distributed queuing and queue-sharing groups are two techniques that you can use to increase the availability of your application systems. Use this topic to find further information about these techniques.

“Influencing workload distribution with shared queues” on page 367

Use this topic to understand the factors that affect workload distribution with shared queues in a queue-sharing group.

Related reference:

“Where to find more information about shared queues and queue-sharing groups” on page 369

Use the table in this topic to find more information about how IBM MQ for z/OS uses shared queues and queue-sharing groups.

What is a shared queue?

▶ z/OS

A shared queue is a type of local queue. The messages on that queue can be accessed by one or more queue managers that are in a sysplex.

A shared queue

The queue managers that can access the same set of shared queues form a group called a *queue-sharing group*.

Any queue manager can access messages

Any queue manager in the queue-sharing group can access a shared queue. This means that you can put a message on to a shared queue on one queue manager, and get the same message from the queue from a different queue manager. This provides a rapid mechanism for communication within a queue-sharing group that does not require channels to be active between queue managers.

IBM WebSphere MQ Version 7.1 and later supports the offloading of messages to Db2 or a shared message data set (SMDS). The offloading of messages of any size is configurable.

In earlier versions of IBM MQ, large messages (> 63 KB) have a placeholder stored in the coupling facility (4 K), and their message data stored in Db2.

Figure 58 on page 324 shows three queue managers and a coupling facility, forming a queue-sharing group. All three queue managers can access the shared queue in the coupling facility.

An application can connect to any of the queue managers within the queue-sharing group. Because all the queue managers in the queue-sharing group can access all the shared queues, the application does not depend on the availability of a specific queue manager; any queue manager in the queue-sharing group can service the queue.

This gives greater availability because all the other queue managers in the queue-sharing group can continue processing the queue if one of the queue managers has a problem.

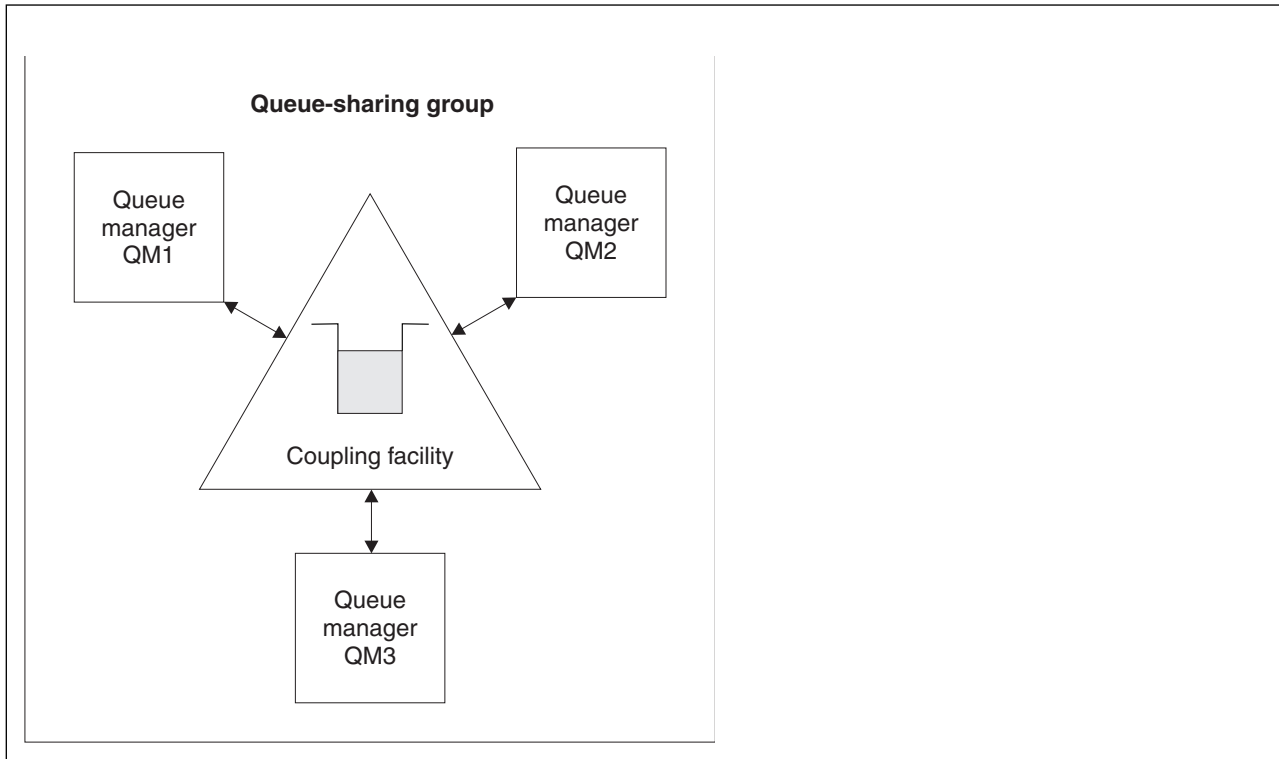


Figure 58. A queue-sharing group

Queue definition is shared by all queue managers

Shared queue definitions are stored in the Db2 database table OBJ_B_QUEUE. Because of this, you need to define the queue only once and then it can be accessed by all the queue managers in the queue-sharing group. This means that there are fewer definitions to make.

By contrast, the definition of a non-shared queue is stored on page set zero of the queue manager that owns the queue (as described in Page sets).

You cannot define a shared queue if a queue with that name has already been defined on the page sets of the defining queue manager. Likewise, you cannot define a local version of a queue on the queue manager page sets if a shared queue with the same name exists.

What is a queue-sharing group?

z/OS

A group of queue managers that can access the same shared queues is called a queue-sharing group. Each member of the queue-sharing group has access to the same set of shared queues.

Queue-sharing groups have a name of up to four characters. The name must be unique in your network, and must be different from any queue manager names.

Figure 59 illustrates a queue-sharing group that contains two queue managers. Each queue manager has a channel initiator and its own local page sets and log data sets.

Each member of the queue-sharing group must also connect to a Db2 system. The Db2 systems must all be in the same Db2 data-sharing group so that the queue managers can access the Db2 shared repository used to hold shared object definitions. These are definitions of any type of IBM MQ object (for example, queues and channels) that are defined only once and then any queue manager in the group can use them. These are called *global* definitions and are described in Private and global definitions.

More than one queue-sharing group can reference a particular data-sharing group. You specify the name of the Db2 subsystem and which data-sharing group a queue manager uses in the IBM MQ system parameters at startup.

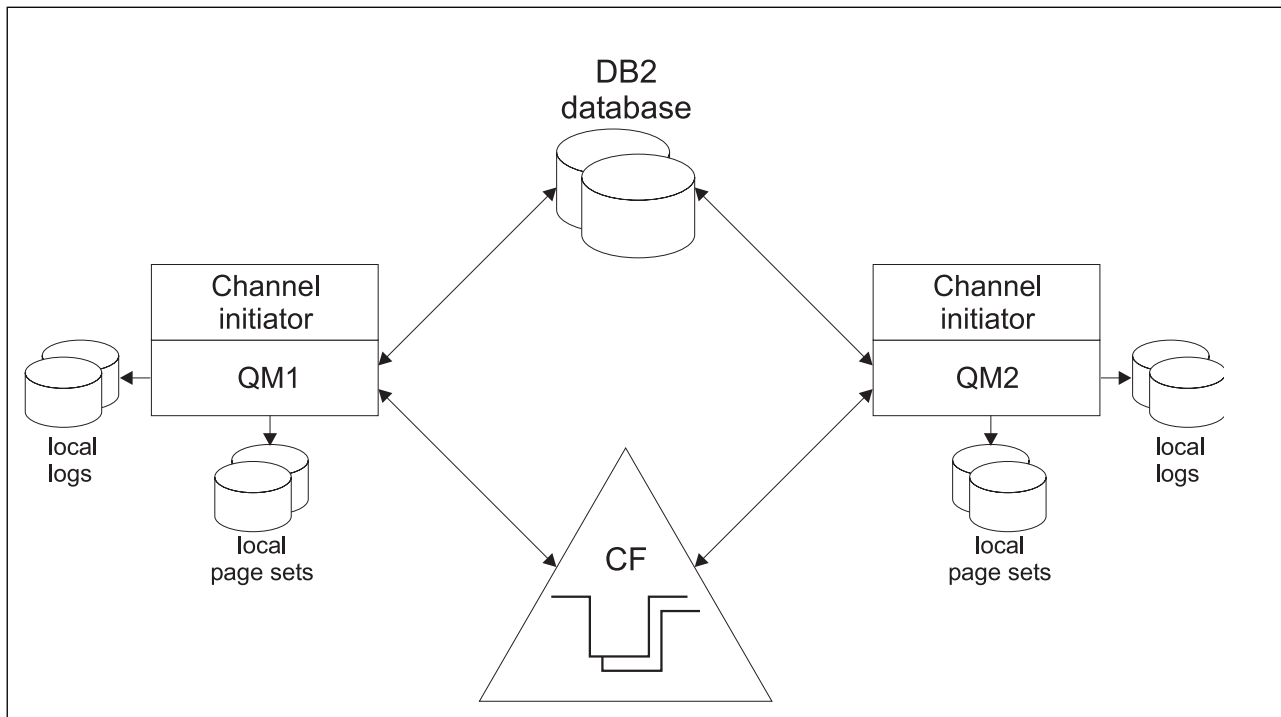


Figure 59. The components of queue managers in a queue-sharing group

When a queue manager has joined a queue-sharing group, it has access to the shared objects defined for that group, and you can use that queue manager to define new shared objects within the group. If shared queues are defined within the group, you can use this queue manager to put messages to and get messages from those shared queues. Any queue manager in the group can retrieve the messages held on a shared queue.

You can enter an MQSC command once, and have it executed on all queue managers within the queue-sharing group as if it had been entered at each queue manager individually. The *command scope* attribute is used for this. This attribute is described in Directing commands to different queue managers.

When a queue manager runs as a member of a queue-sharing group it must be possible to distinguish between IBM MQ objects defined privately to that queue manager and IBM MQ objects defined globally that are available to all queue managers in the queue-sharing group. The *queue-sharing group disposition* attribute is used for this. This attribute is described in Private and global definitions.

You can define a single set of security profiles that control access to IBM MQ objects anywhere within the group. This means that the number of profiles you have to define is greatly reduced.

A queue manager can belong to only one queue-sharing group, and all queue managers in the group must be in the same sysplex. You specify which queue-sharing group the queue manager belongs to in the system parameters at startup.

Where are shared queue messages held?

z/OS

Each message on a shared queue is represented by an entry in a z/OS coupling facility list structure. If the message data is too large to fit in the same entry, it is offloaded either to a shared message data set (SMDS) or to Db2.

If the CF structure has been configured to use System Class Memory (SCM), IBM MQ can use this with no additional configuration. See IBM MQ V8 Features and Enhancements, Chapter 8.

Shared queue message storage

Messages that are put onto shared queues are not stored on page sets and do not use buffer pools.

The messages in shared queues have entries on list structures in the z/OS coupling facility (CF). Many queue managers in the same sysplex can access those messages using the CF list structure.

The message data for small shared queue messages is normally included in the coupling facility entry. For larger messages, the message data can be stored either in a shared message data set (SMDS), or as one or more binary large objects (BLOBs) in a Db2 table which is shared by a Db2 data sharing group. Message data exceeding 63 KB is always offloaded to SMDS or Db2. Smaller messages can also optionally be offloaded in the same way to save space in the coupling facility structure. See “Specifying offload options for shared messages” on page 328 for more details.

Messages put on a shared queue are referenced in a coupling facility structure until they are retrieved by an MQGET. Coupling facility operations are used to:

- Search for the next retrievable message
- Lock uncommitted messages on shared queues
- Notify interested queue managers about the arrival of committed messages

MQPUT and MQGET operations on persistent messages are recorded on the log of the queue manager performing that operation. This minimizes the risk of data loss in the event of a coupling facility failure.

The coupling facility

The messages held on shared queues are referenced inside a coupling facility. The coupling facility lies outside any of the z/OS images in the sysplex and is typically configured to run on a different power

supply. The coupling facility is therefore resilient to software failures and you can configure it so that it is resilient to hardware failures or power-outages. This means that messages stored in the coupling facility are highly available.

Each coupling facility list structure used by IBM MQ is dedicated to a specific queue-sharing group, but a coupling facility can hold structures for more than one queue-sharing group. Queue managers in different queue-sharing groups cannot share data. Up to 32 queue managers in a queue-sharing group can connect to a coupling facility list structure at the same time.

A single coupling facility list structure can contain up to 512 shared queues. The total amount of message data stored in the structure is limited by the structure capacity. However, with **CFLEVEL(5)** you can use the offload parameters to offload data for messages less than 63 KB thereby increasing the number of messages which can be stored in the structure, although each message still requires at least a coupling facility entry plus at least 512 bytes of data.

The size of the list structure is restricted by the following factors:

- It must lie within a single coupling facility.
- It might share the available coupling facility storage with other structures for IBM MQ and other products.

Coupling facility list structures can have storage class memory associated with them. In certain situations this storage class memory can be useful when used with shared queues. See “Use of storage class memory with shared queues” on page 345 for more information.

Planning the CF structure size

If you require guidance on the sizing of your CF structures you can use the MP16: IBM MQ for z/OS Capacity planning and tuning supportpac. You can also use the web-based tool CFSizer, which is provided by IBM to assist with CF sizes.

The CF structure object

The queue manager's use of a coupling facility structure is specified in a CF structure (CFSTRUCT) IBM MQ object.

These structure objects are stored in Db2.

When using z/OS commands or definitions relating to a coupling facility structure, the first four characters of the name of the queue-sharing group are required. However, an IBM MQ CFSTRUCT object always exists within a single queue-sharing group, and so its name does not include the first four characters of the name of the queue-sharing group. For example, CFSTRUCT(MYDATA) defined in queue-sharing group starting with SQ03 would use coupling facility list structure SQ03MYDATA.

CF structures have a CFLEVEL attribute that determines their functional capability:

- 1, 2 - can be used for nonpersistent messages less than 63 KB
- 3 - can be used for persistent and nonpersistent messages less than 63 KB
- 4 - can be used for persistent and nonpersistent messages up to 100 MB
- 5 - can be used for persistent and nonpersistent messages up to 100 MB and selectively offloaded to shared message data sets (SMDS) or Db2.

Backup and recovery of the coupling facility

You can back up coupling facility list structures using the IBM MQ command `BACKUP CFSTRUCT`. This puts a copy of the persistent messages currently within the CF structure onto the active log data set of the queue manager making the backup, and writes a record of the backup to Db2.

If coupling facility fails, you can use the IBM MQ command `RECOVER CFSTRUCT`. This uses the backup record from Db2 to locate and restore persistent messages from the backup of the CF structure. Any activity since the last backup is replayed using the logs of all the queue managers in the queue-sharing group, and the CF structure is then restored up to the point before the failure.

See the `BACKUP CFSTRUCT` and `RECOVER CFSTRUCT` commands for more details.

Related concepts:

“Specifying offload options for shared messages”

You can choose where the message data for a shared queue message is stored, either in a Db2 table or a shared message data set (SMDS). You can also select which messages are offloaded, based on the size of the message and the current usage of the coupling facility structure (CF).

“Managing your shared message data set (SMDS) environment” on page 330

If you select shared message data sets to offload large messages then you must also be aware of the information that IBM MQ uses to manage these data sets and the commands used to work with this information. Use this topic to understand how to manage shared message data sets.

Specifying offload options for shared messages:

You can choose where the message data for a shared queue message is stored, either in a Db2 table or a shared message data set (SMDS). You can also select which messages are offloaded, based on the size of the message and the current usage of the coupling facility structure (CF).

The message data for shared queues can be offloaded from the coupling facility and stored in either a Db2 table or in an IBM MQ managed data set called a *shared message data set* (SMDS).

For messages larger than the coupling facility entry size of 63 KB, offloading message data to a SMDS can have a significant performance improvement compared with offloading to Db2.

Every shared queue message is still managed using a list entry in a coupling facility structure, but when the message data is offloaded to the SMDS, the coupling facility entry only contains some control information and a list of references to the relevant disk blocks where the message is stored. Using this mechanism means the amount of coupling facility element storage required for each message is only a fraction of the actual size of the message.

Selecting where the shared queue messages are stored

The selection of SMDS or Db2 shared message storage is controlled with the **OFFLOAD(SMDS|DB2)** parameter on the **CFSTRUCT** definition. **OFFLOAD(SMDS)** is the default value.

This parameter also requires the **CFSTRUCT** to use **CFLEVEL(5)** or greater. Only queue managers at IBM WebSphere MQ Version 7.1 or higher can connect to a CF structure at this level.

It is only possible to alter a structure up to **CFLEVEL(5)** if all queue managers in the queue-sharing group are at IBM WebSphere MQ Version 7.1 or higher.

The **OFFLOAD** parameter is only valid from **CFLEVEL(5)**. See `DEFINE CFSTRUCT` for more details.

OFFLOAD(DB2) is supported primarily for migration purposes.

Selecting which shared queue messages are offloaded

Message data is offloaded to SMDS or Db2 based on the size of the message data, and the current usage of the coupling facility structure. There are three rules, and each rule specifies a matching

pair of parameters. These parameters are a corresponding coupling facility structure usage threshold percentage (**OFFLDnTH**) and a message size limit (**OFFLDnSZ**).

The current implementation of the three rules is specified using the following pairs of keywords:

- OFFLD1TH and OFFLD1SZ
- OFFLD2TH and OFFLD2SZ
- OFFLD3TH and OFFLD3SZ

Rule pair	Default value	Description
Rule pair 1	OFFLD1TH(70) and OFFLD1SZ(32K)	If the coupling facility structure is more than 70% full offload data for messages exceeding 32 KB
Rule pair 2	OFFLD2TH(80) and OFFLD2SZ(4K)	If the coupling facility structure is more than 80% full offload data for messages exceeding 4 KB
Rule pair 3	OFFLD3TH(90) and OFFLD3SZ(0K)	If the coupling facility structure is more than 90% full offload data for messages exceeding 0 KB (all messages)

Important: Messages exceeding 63.75 KB are always offloaded. The value 64K for OFFLD x SZ is used to indicate that the rule is **not** in effect.

Each message which is offloaded still requires 0.75 KB of storage in the coupling facility.

The three offload rules which can be specified for each structure are intended to be used as follows.

- Performance
 - When there is plenty of space in the application structure, message data should only be offloaded if it is too large to store in the structure, or if it exceeds some lower message size threshold such that the performance value of storing it in the structure is not worth the amount of structure space that it would need.
 - If a specific message size threshold is required, it is conventionally specified using the first offload rule.
- Capacity
 - When there is very little space in the application structure, the maximum amount of message data should be offloaded so as to make the best use of the remaining space.
 - The third offload rule is conventionally used to indicate that when the structure is nearly full, most messages should be offloaded, so the entries in the application structure will be typically of the minimum size (requiring about 0.75K bytes).
 - The usage threshold parameter should be chosen based on the application structure size and the maximum anticipated backlog. For example, if the maximum anticipated backlog is 1M messages, then the amount of structure storage required for this number of messages is about 0.75G bytes. This means for example that if the structure is about 10G bytes, the usage threshold for offloading all messages must be set to 92% or lower.
 - Structure space is divided into elements and entries, and even though there may be enough space overall, one of these may run out before the other. The system provides AUTOALTER capabilities to adjust the ratio when necessary, but this is not very sensitive, so the amount of space actually available may be somewhat less. It may be better therefore to aim to use not more than 90% of the maximum structure space, so in the previous example, the usage threshold for offloading all messages would be better set around 80%.
- Cushioned transition:

- As the amount of space left in the coupling facility structure decreases, it would be undesirable to have a large sudden change in the performance characteristics. It is also undesirable for coupling facility management to have a sudden threshold change in the typical ratio of entries to elements being used.
- The second offload rule is conventionally used to provide some intermediate cushion between the performance and capacity biased offload rules. It can be set to cause a significant increase in offload activity when the space used in the coupling facility structure exceeds an intermediate threshold. This means that the remaining space is used up more slowly, and gives the coupling facility automatic alter processing more time to adapt to the higher usage levels.

If the coupling facility structure cannot be expanded, and there is a need to store at least some predetermined number of messages, the third rule can be modified as necessary to ensure that offloading of data for all messages starts at an appropriate threshold to ensure space is reserved for that predetermined number of messages.

For example, if the coupling facility structure size is 4 GB, and the predetermined number of messages is 1 million, then $1,000,000 * 0.75 \text{ KB}$ are needed, which is 768 MB, 18.75% of 4 GB. In this case the threshold for offloading all messages needs to be set around 80% rather than 90%. This gives parameters OFFLD3TH(80) and OFFLD3SZ(0K) . The other offload parameters would also need to be adjusted.

If it is found that offloading very small messages has a significant performance impact, but the relative impact is less for larger messages, then the usage thresholds for the other rules can be reduced to offload larger messages earlier, leaving more space in the structure for small messages before they need to be offloaded.

For example, if messages exceeding 32KB occur frequently but the elapsed time performance for offloading them (as determined from RMF statistics or application performance) is very similar to that for keeping them in the coupling facility, then the threshold for the first rule could be set to 0% to offload all such messages. This gives parameters OFFLD1TH(0) and OFFLD1SZ(32K). Again the other offload parameters would need to be adjusted.

If there are many messages around specific intermediate sizes, such as 16 KB and 6 KB, then it might be useful to change the message size option for the second rule so that the larger ones get offloaded at a fairly low usage threshold, saving a significant amount of space, but the smaller ones still get stored only in the coupling facility.

Managing your shared message data set (SMDS) environment:

If you select shared message data sets to offload large messages then you must also be aware of the information that IBM MQ uses to manage these data sets and the commands used to work with this information. Use this topic to understand how to manage shared message data sets.

SMDS objects

The properties and status of each shared message data set are tracked in a shared SMDS object which can be updated via any queue manager in the queue-sharing group.

There is one shared message data set for each queue manager that can access each coupling facility application structure. The shared message data set is identified by the owning queue manager name, specified using the SMDS keyword, and by the application structure name, specified using the CFSTRUCT keyword.

The SMDS object is stored in an array (with one entry per queue manager in the group) which forms an extension of the corresponding CFSTRUCT object stored in Db2.

There is no command to DEFINE or DELETE the SMDS object because it is created or deleted as part of the CFSTRUCT object, but there is a command to ALTER it to change settings for an individual owning queue manager.

For further information on SMDS commands, see “SMDS related commands” on page 342

SMDSCONN information

It is possible for a shared message data set to be in a normal state, but for one or more queue managers to be unable to connect to it, for example because of a problem with a security definition or with direct access device connectivity. It is therefore necessary for each queue manager to keep track of connection status, and availability information for each shared message data set, indicating for example whether it can currently connect to it, and if not why not.

The SMDSCONN information represents a queue manager connection to a shared message data set. As for the shared message data set itself, it is identified by the queue manager which owns the shared message data set (as specified on the SMDS keyword for the shared object itself) combined with the CFSTRUCT name.

There is no parameter to identify the connecting queue manager because commands addressed to a specific queue manager can only refer to SMDSCONN information for that same queue manager.

The SMDSCONN information entries are maintained in main storage in the owning queue manager, and are re-created when the queue manager is restarted. However, if a connection from an individual queue manager has been explicitly stopped, this information is also stored as a flag in a connection array in the corresponding CFSTRUCT or SMDS object, so that it persists across a queue manager restart.

Status and availability information

Status information indicates the state of a resource or connection (for example, whether it is not yet being used, is in normal use or is in need of recovery). It is usually described using the STATUS keyword. The possible values depend on the type of object.

Status information is normally updated automatically, for example when an error is detected while using the resource or connection. However, in some cases a command can also be used to update the status, to allow for cases when it is not possible for a queue manager to determine the correct status automatically.

Availability information indicates whether the resource or connection can be used, and is usually primarily determined by the status information. For the resource or connection types used in shared message data set support, three levels of availability are implemented:

Available

This means that the resource is available to be used normally. This does not necessarily mean that it is in use at present (which can be determined instead from the STATUS value). For a data set, if it requires restart processing, this allows the owning queue manager to open it, but other queue managers must wait until the data set is back in the ACTIVE state.

Unavailable because of error

This means that the resource has been made unavailable automatically because of an error and is not expected to be available again until some form of repair or recovery processing has been performed. However, attempts to make it available again are permitted without operator intervention. Such an attempt can also be triggered by a command to mark the resource as enabled, or a command which changes the status in such a way as to indicate that recovery processing has been completed.

The reason that the resource has been made unavailable is normally obvious from the related STATUS value, but in some cases there may be other reasons to make the resource unavailable, in which case a separate REASON value is provided to indicate the reason.

Unavailable because of operator command

This means that access to the resource has been explicitly disabled by a command. It can only be made available by using a command to enable it again.

SMDS availability

For the shared SMDS object, the availability is described by the ACCESS keyword, with the possible values ENABLED, SUSPENDED and DISABLED.

The availability can be updated using a **RESET SMDS** command for the relevant shared object from any queue manager in the group to set ACCESS(ENABLED) or ACCESS(DISABLED).

If the availability was previously ACCESS(SUSPENDED), changing it to ACCESS(ENABLED) will trigger a new attempt to use the shared message data set, but if the previous error is still present, the availability will be reset back to ACCESS(SUSPENDED).

SMDSCONN availability

For a local SMDSCONN information entry, the availability is described by the AVAIL keyword, with the possible values NORMAL, ERROR or STOPPED. The availability can be updated using a **START SMDSCONN** or **STOP SMDSCONN** command addressed to a specific queue manager to enable or disable its connection.

If the availability was previously AVAIL(ERROR), changing it to AVAIL(NORMAL) will trigger a new attempt to use the shared message data set, but if the previous error is still present, the availability will be reset back to AVAIL(ERROR).

Shared message data set shared status and availability

The availability of each shared message data set is managed within the group using shared status information, which can be displayed using the **DISPLAY CFSTATUS** command with TYPE(SMDS). This displays status information for each queue manager that has activated a data set for each structure. Each data set can be in one of the following states:

NOTFOUND

This means that the corresponding data set has not yet been activated. This status only appears when a specific queue manager is specified, as data sets which have not been activated are skipped when all queue managers are selected.

NEW The data set is being opened and initialized for the first time, ready to be made active.

ACTIVE

This means that the data set is fully available and should be allocated and opened by all active queue managers for the structure.

FAILED

This means the data set is not available at all (except for recovery processing) and must be closed and deallocated by all queue managers.

INRECOVER

This means that media recovery (using RECOVER CFSTRUCT) is in progress for this data set.

RECOVERED

This indicates that a command has been issued to switch a failed data set back to the active state, but further restart processing is required which is not yet complete, so the data set can only be opened by the owning queue manager for restart processing.

EMPTY

The data set contains no messages. The data set is put into this state if it is closed normally by

the owning queue manager, at a time when it does not contain any messages. It can also be put into **EMPTY** state when the previous data set contents are to be discarded because the application structure has been emptied (using **RECOVER CFSTRUCT** with **TYPE PURGE** or, for a nonrecoverable structure only, by deleting the previous instance of the structure). The next time the data set is opened by its owning queue manager, the space map is reset to empty, and the status is changed to **ACTIVE**. As the previous data set contents are no longer required, a data set in this state can be replaced with a newly allocated data set, for example to change the space allocation or move it to another volume.

The command output includes the date and time at which recovery logging was enabled, if any, and the date and time at which the data set failed, if it is not currently active.

A shared message data set can be put into a **FAILED** state either by a **RESET SMDS** command or automatically when any of the following types of error are detected:

- The data set cannot be allocated or opened by the owning queue manager.
- Validation of the data set header fails after it has been successfully opened by any queue manager.
- A permanent I/O error occurs when the owning queue manager is reading or writing data.
- A permanent I/O error occurs when another queue manager is reading data from a data set which had successfully completed open processing and validation.

When a data set is in the **FAILED** or **INRECOVER** state, it is not available for normal use, so if the availability state is **ACCESS(ENABLED)** it is changed to **ACCESS(SUSPENDED)**.

If a data set has been put into the **FAILED** state but no media recovery is required, for example because the data was still valid but the storage device was temporarily offline, then the **RESET SMDS** command can be used to request changing the status directly to the **RECOVERED** state.

When the data set enters the **RECOVERED** state, either on completion of recovery processing or as a result of the **RESET SMDS** command, then it is ready to be used again once restart processing has been completed. If it was in the **ACCESS(SUSPENDED)** state, it is automatically switched back to the **ACCESS(ENABLED)** state, which allows the owning queue manager to perform restart processing. When restart processing completes, the state is changed to **ACTIVE** and all other queue managers can then connect to the data set again.

Shared message data set connection status and availability

Each queue manager maintains local status and availability information for its connection to each shared message data set owned by itself and by other queue managers in the group. This information can be displayed using the **DISPLAY SMDSCONN** command.

If it is unable to access a shared message data set in the **ACTIVE** state which belongs to another queue manager it flags the connection as being unavailable from its own point of view.

If the error definitely indicates a problem with the data set itself, the queue manager also automatically changes the shared status to indicate that the data set is now in a **FAILED** state. However, if the error could be caused by an environmental problem, such as not being authorized to open the data set, the queue manager issues error messages and treats the data set as being unavailable, but it does not modify the shared data set status. If the environmental error turns out to be a problem with the data set anyway (for example it has been allocated on a device which cannot be accessed by some of the queue managers) then an operator can use the **RESET SMDS** command specifying **STATUS(FAILED)** to allow the data set to be recovered or repaired as necessary.

If a connection to a shared message data set could not be established but the data set appears to be valid, a new attempt to use it can be triggered by issuing a **START SMDSCONN** command for the owning queue manager.

If there is an operational need to terminate the connection between a specific queue manager and a data set temporarily, but the data set itself is not damaged, then the data set can be closed and deallocated using the **STOP SMDSCONN** command. If the data set is in use, the queue manager will close it normally (although any requests for data in that data set will be rejected with a return code). If it is the owned data set, the queue manager will save the space map during CLOSE processing, avoiding the need for restart processing.

If a data set needs to be taken out of service temporarily from all queue managers (for example to move it) but is not damaged, then it is best to use **STOP SMDSCONN** for the relevant data set with the option **CMDSCOPE(*)** to stop the queue managers using it first, as this will avoid the need for restart processing when the data set is brought back into service. In contrast, if the data set is marked as **FAILED** this tells queue managers that they must stop using it immediately, which means that the space map will not be saved and will need to be rebuilt by restart processing.

Access to any shared message data sets previously in the **ACCESS(SUSPENDED)** state will be retried if the queue manager is restarted.

Shared message data set recovery logging

Persistent shared messages are logged for media recovery purposes. This means that the messages can be recovered after any failure of coupling facility structures or shared message data sets, provided that the recovery logs are still intact. Persistent messages can also be re-created from the recovery logs at another site for disaster recovery purposes.

When the message data is written to a shared message data set, each block written to the data set is logged separately followed by the message entry (including the data map) as written to the coupling facility. The recovery process always recovers the coupling facility structure, but it does not need to recover individual shared message data sets except when the data set status is **FAILED**, or when the status is **ACTIVE** but the data set header record is no longer valid, indicating that the data set has been re-created. A data set is not selected for recovery if its status is **ACTIVE** and the data set header is still valid, nor if its status is **EMPTY**, indicating that no messages were stored in it at the time of the failure.

Shared message data set backups

When **BACKUP CFSTRUCT** is used to make a backup of the shared messages in an application structure, any data for persistent messages stored in shared message data sets is backed up at the same time, as for persistent shared messages previously stored in DB.

Shared message data set recovery

If a shared message data set is corrupted or lost, then it needs to be put into the **FAILED** state to stop the queue managers from using it until it has been repaired. This normally happens automatically, but can also be done using the **RESET SMDS** command specifying **STATUS(FAILED)**.

If the shared message data set contained any persistent messages, these can be recovered using the **RECOVER CFSTRUCT** command. This command first restores any persistent message data for that shared message data set from the most recent **BACKUP CFSTRUCT** command, then applies all logged changes since that time. If no **BACKUP CFSTRUCT** command has been performed since the time that the data set was first activated, it is reset to empty then all changes since activation are applied.

If the **CFSTRUCT** contents and all of the shared message data sets are unavailable, for example in a disaster recovery situation, they can all be recovered in a single **RECOVER CFSTRUCT** command.

If a shared message data set is damaged but recovery was not active for the **CFSTRUCT**, or the log containing the latest **BACKUP CFSTRUCT** is unavailable or unusable, then the messages offloaded to that

data set cannot be recovered. In this case, the **RECOVER CFSTRUCT** command with the parameter **TYPE(PURGE)** can be used to mark the shared message data set as empty and delete any messages from the structure which had data stored in that data set.

When the **RECOVER CFSTRUCT** command is issued, the shared message data set status is changed from **FAILED** to **INRECOVER**. If recovery completes successfully, the status is automatically changed to **RECOVERED**, otherwise it changes back to **FAILED**.

When the data set is changed to the **RECOVERED** state, this tells the owning queue manager that it can now try to open the data set and perform restart processing.

Shared message data set recovery and syncpoints

The shared message data set recovery process reapplies the changes for all complete log records up to the end of the log, regardless of syncpoints.

If changes were made within syncpoint, restart or recovery processing for the **CFSTRUCT** may result in backing out of uncommitted requests, so some of the recovered changes may not actually be used, but there is no harm in recovering them anyway.

It is also possible that an uncommitted **MQPUT** message may have been written to the structure but the corresponding data may not have been written to the data set or the log (as I/O completion is only forced at the start of syncpoint processing). This is harmless because restart processing will back out the message entry in the structure, so the fact that it refers to unrecovered data does not matter.

Shared message data set restart processing

If a queue manager connection to a **CFSTRUCT** terminates normally, the queue manager writes out the free block space map for each shared message data set to a checkpoint area within the data set, just before the data set is closed. The space map can then be read in again at connection restart time, provided that neither the **CFSTRUCT** nor the shared message data set require any recovery processing before the next restart.

However, if a queue manager terminates abnormally, or the structure or data set require any recovery processing, then additional processing is required to rebuild the space map dynamically when the queue manager connection to the structure is restarted.

Provided that the data set itself did not need to be recovered, queue manager restart simply scans the current contents of the structure to locate references to message data owned by the current queue manager, and marks the relevant data blocks as owned in the space map. Other queue managers can continue to use the structure and read the data owned by the restarting queue manager while the space map is being rebuilt.

Shared message data set restart after recovery

If a shared message data set had to be recovered from a backup, then all nonpersistent messages stored in the data set will have been lost, and if the data set was recovered using **TYPE(PURGE)** then all messages stored in the data set will have been lost. Until recovery has completed, the data set will be marked as **FAILED** or **INRECOVER** so any attempt to read one of the affected messages from another queue manager returns an error code indicating that the data set is temporarily unavailable.

When the data set has been recovered, the status is changed to **RECOVERED**, which allows the owning queue manager to open it for restart processing, but the data set remains unavailable to other queue managers. Queue manager restart scans the structure to rebuild the space map for any remaining messages. The scan also checks for messages for which the data has been lost, and deletes them from the structure (or if necessary flags them as lost, to be deleted later).

The data set status is automatically changed from RECOVERED to ACTIVE when this restart scan completes, at which point other queue managers can start using it again.

Shared message data set usage information

The DISPLAY USAGE command now also shows information about shared message data set space and buffer pool usage for any currently open shared message data sets. This information is displayed if either the new option TYPE(SMDS) or the existing option TYPE(ALL) is specified.

Shared message data performance and capacity considerations

Monitoring data set usage

The current percentage full of each owned shared message data set can be displayed by the **DISPLAY USAGE** command with the option **TYPE(SMDS)**.

The queue manager will normally automatically expand a shared message data set when it reaches 90% full, provided that the option **DSEXPAND(YES)** is in effect for the SMDS definition. This applies when either the SMDS option is set to **DSEXPAND(YES)** or the SMDS option is set to **DSEXPAND(DEFAULT)** and the CFSTRUCT default option is set to **DSEXPAND(YES)**.

If the expansion attempt fails because no secondary allocation size was specified when the data set was created (giving message IEC070I with reason code 203) the queue manager repeats the expansion request using an override secondary allocation of approximately 20% of the current size.

When a data set is expanded, the new data set extents are formatted as part of the expansion processing, which can take tens of seconds, or even minutes for very large extents. The new space becomes available for use after formatting is complete and the catalog has been updated to show the new high used control interval.

If new messages are being created very rapidly, it is possible for the existing data set to become full before expansion processing completes. In this case, any request which could not allocate space is temporarily suspended until the expansion attempt completes and the new space becomes available for use. If the expansion was successful the request is retried automatically.

If an expansion attempt fails, because of a lack of available space or because the maximum extents have already been reached, a message is issued giving the reason for the failure, then the override option for the affected SMDS is automatically altered to **DSEXPAND(NO)** to prevent further expansion attempts. In this case, there is a risk that the data set may become full, in which case further action may be needed as described in Data set becomes full.

Monitoring application structure usage

The usage level of an application structure can be displayed using the MVS **DISPLAY XCF,STRUCTURE** command specifying the full name of the application structure (including the queue-sharing group prefix). The IXC360I response message shows current usage of elements and entries.

When the structure usage exceeds the **FULLTHRESHOLD** value specified in the CFRM policy, the system issues message IXC585E and may perform automatic **ALTER** actions if specified, which may either alter the entry to element ratio or increase the structure size.

Optimising buffer pool sizes

Each buffer in a shared buffer pool is used to read or write a contiguous range of pages for one message of up to the logical block size. If the message spills over into further blocks, each range of pages in a separate block requires a separate buffer.

Buffers containing message data after a write or read operation are retained in storage and reused using a least-recently-used (LRU) cache scheme so that a request to read the same data again shortly afterwards will not need to go to disk. This provides a significant optimization when

shared messages are written and then read back soon afterwards by applications running on the same system. If messages owned by another queue manager are browsed for selection purposes then retrieved, this also avoids the need to reread the message from disk.

This means that the number of buffers required for each application structure is one for each concurrent API request which reads or writes large messages for that application structure plus some number of additional buffers which will be used to save recently accessed data in order to optimize subsequent read accesses.

For shared buffer pools, if there are insufficient buffers, API requests will simply wait if a buffer is not immediately available. However, this situation should be avoided as it can cause significantly degraded performance.

The statistics from the **DISPLAY USAGE** command for shared buffer pools show whether there have been any buffer waits within the current statistics interval, and also shows the lowest number of free buffers (or a negative value indicating the maximum number of threads which waited for a buffer at any time), the number of buffers which have saved data, and the percentage of the times that a buffer request has successfully found saved data on the LRU chain ("LRU hits") instead of having to read it ("LRU misses")¹.

- If there have been any waits, the number of buffers should be increased.
- If there are many unused buffers, the number of buffers may be reduced to make more storage available in the region for other purposes.
- If there are many buffers containing saved data but the proportion of reads which were hits against that saved data is very small, the number of buffers may be reduced if the storage could be better used for other purposes. The number of buffers should not however be reduced by more than the lowest number of free buffers, as that could trigger waits, and it should preferably be high enough that the lowest free buffer count is normally well above zero.

Deleting shared message data sets

The **DELETE CFSTRUCT** command (which is only allowed when all shared queues in the structure are empty and closed) does not delete the shared message data sets themselves, but they can be deleted in the usual way after this command has completed. If the same data set is to be reused as a shared message data set, it must be reformatted first to reset it to the empty state.

Exception situations for shared message data sets

There are a number of exception situations which can occur during normal use, even when no software or hardware error is present.

Data set becomes full

If a data set becomes full but cannot be expanded, or the expansion attempt fails, applications using the corresponding queue manager to write large messages to the corresponding application structure will receive error 2192, **MQRC_STORAGE_MEDIUM_FULL** (also known as **MQRC_PAGESET_FULL**).

A data set could become full because of a failure in the application which is supposed to process the data, causing a large backlog of messages to accumulate. If so, expanding the data set any further will only be a temporary solution, and it is important to get the processing application going again as soon as possible.

If more space can be made available the **ALTER SMDS** command can then be used to set **DSEXPAND(YES)** or **DSEXPAND(DEFAULT)** (assuming that YES has been set or assumed as the **DSEXPAND** default for the CFSTRUCT definition) to trigger a retry. If the reason for the failure was however that maximum extents had been reached, the new expansion attempt will be rejected with a

1. $(\text{Hits} / (\text{Hits} + \text{Misses})) * 100$

message and **DSEXPAND(NO)** will be set again. In this case, the only way to expand it any further is to reallocate it, which involves making it temporarily unavailable, as described next.

Data set needs to be moved or reallocated

If a data set needs to be moved or expanded but is otherwise in normal use, it can be taken out of use temporarily to allow it to be moved or reallocated. Any API request which attempts to use the data set while it is unavailable will receive the reason code `MQRC_DATA_SET_NOT_AVAILABLE`.

1. Use the **RESET SMDS** command to mark the data set as **ACCESS(DISABLED)**. This will cause it to be closed normally and deallocated by all currently connected queue managers.
2. Move or reallocate the data set as necessary, copying the old contents to the newly allocated data set, for example using the Access Method Services (AMS) **REPRO** command.

Do not attempt to preformat the new data set before copying the old data into it, as this would result in the copied data being appended to the end of the formatted data set.

3. Use the **RESET SMDS** command to mark the data set as **ACCESS(ENABLED)** again, to bring it back into use.

If the old contents are smaller than the size of the new data set, the rest of the space will be preformatted automatically when the new data set is opened.

If the old contents were larger than the size of the new data set then the queue manager has to scan the messages in the coupling facility structure and rebuild the space map to ensure that none of the active data has been lost. If any reference is found to a data block which is outside the new extents, the data set is marked as **STATUS(FAILED)** and must be repaired by replacing the data set with one of the correct size and either copying the old data set into it again or using **RECOVER CFSTRUCT** to recover any persistent messages.

Coupling facility structure is low on space

If the coupling facility structure is running out of space, causing message `IXC585E`, it is worth checking whether the offload rules have been set to ensure that the maximum amount of data is being offloaded in this case. If not, the offload rules can be modified using the **ALTER CFSTRUCT** command.

Error situations for shared message data sets

There are a number of problems to be aware of, which can only be caused by errors and not occur in normal operational situations.

Owned data set cannot be opened

If the queue manager which owns a shared message data set cannot allocate it or open it, or the data set attributes are not supported, the queue manager sets an appropriate **SMDSCONN** status value of **ALLOCFAIL** or **OPENFAIL** and sets the **SMDSCONN** availability to **AVAIL(ERROR)**. It also sets the SMDS availability to **ACCESS(SUSPENDED)**. When the error has been corrected, use the **RESET SMDS** command to set **ACCESS(ENABLED)** to trigger a retry, or issue the **START SMDSCONN** command to the owning queue manager.

Read-only data set cannot be opened

If a queue manager cannot allocate or open a shared message data set owned by another queue manager and marked as **STATUS(ACTIVE)**, it assumes that this is probably due to a specific problem with its connection to the data set (represented by the **SMDSCONN** object) rather than a problem with the data set itself.

It marks the **SMDSCONN** as **STATUS(ALLOCFAIL)** or **STATUS(OPENFAIL)** as appropriate and marks the **SMDSCONN** availability as **AVAIL(ERROR)** to prevent further attempts to use it.

If the problem can be corrected without affecting the status of the data set itself, use the **START SMDSCONN** command to trigger a retry.

If the problem turns out to be a problem with the data set itself, then the **RESET SMDS** command can be used to mark the data set as **STATUS(FAILED)** until it has been recovered. When the data set has been recovered, the action of changing the status back to **STATUS(ACTIVE)** will cause other queue managers to be notified. If the **SMDSCONN** is marked as **AVAIL(ERROR)**, it will automatically be changed back to **AVAIL(NORMAL)** to trigger a new attempt to open the data set.

Data set header is corrupt

If the data set was successfully opened but the format of the header information is incorrect, the queue manager closes and deallocates the data set and sets the status set to **STATUS(FAILED)** and the availability to **ACCESS(SUSPENDED)**. This allows **RECOVER CFSTRUCT** to be used to recover the contents.

If the error arose because the data set contained residual data from another use and had not been subsequently preformatted, then preformat the data set and use the **RESET SMDS** command to change the status to **STATUS(RECOVERED)**.

Otherwise, the data set must be recovered.

Data set is unexpectedly empty

If the queue manager opens a data set which is marked as **STATUS(ACTIVE)** but finds that it is uninitialized or newly preformatted but otherwise valid, the queue manager closes and deallocates the shared message data set then sets the status to **STATUS(FAILED)** and the availability to **ACCESS(SUSPENDED)**.

Data set has permanent I/O errors

If a data set has permanent I/O errors after successful **OPEN** processing, it probably needs recovery. The queue manager will mark the data set as **STATUS(FAILED)** so that all currently connected queue managers will close and deallocate it.

Data set has recoverable I/O errors

If there are hardware problems with the data set, it is possible that this might result in recoverable I/O errors which are not reflected back to the queue manager but which cause significant performance degradation, and also indicate a risk of permanent I/O errors in the near future.

In this case, the data set may be taken off line for recovery by using the **RESET SMDS** command to mark it as **STATUS(FAILED)**. This will cause it to be closed and deallocated by all queue managers, so for example it could be moved to a new volume before being made available again.

When a data set is made unavailable in this way, the space map is not saved so the queue manager connection restart processing will need to scan the coupling facility structure to locate messages in the data set and rebuild the space map before the data set can be made available again. As an alternative, if the shared message data set is still usable, it set can be made unavailable more gently by using the **RESET SMDS** command to mark the data set **ACCESS(DISABLED)** until it is ready to be made available again.

Data set contents are incorrect

The queue manager cannot detect directly that a data set contains incorrect data or is not up to date, for example because a volume including that data set had to be restored from backups. However, it performs integrity checks which make it very unlikely that any such errors could result in incorrect message data being seen by application programs.

For integrity checking purposes, each message block in the data set is prefixed with a copy of the corresponding coupling facility entry ID, including a unique time stamp, which is checked whenever the message block is read, before the message data is passed to the user program. If the message block prefix does not match the entry ID (and the coupling facility entry was not deleted in the mean time) the message block is assumed to be damaged and unusable.

If the damaged message was persistent, the data set is marked as **STATUS(FAILED)** and the structure contents must be recovered using the **RECOVER CFSTRUCT** command. If the damaged message was non-persistent, there is no way to recover it, so a diagnostic message is issued and the corresponding coupling facility message entry is deleted.

If no saved space map is available when the data set is opened, it is rebuilt by scanning the coupling facility structure for references to data in the data set. During this scan, the queue manager performs a number of actions:

1. The queue manager determines the location of the most recent message (if any) currently remaining in the data set.
2. The queue manager then reads that message from the data set to ensure that the block prefix matches the message entry id

These actions ensure that the queue manager detects any case where the data set is down-level, and marks the data set as **FAILED**. This check does however tolerate the case where the data set was restored from a previous copy and either no new messages had been added since then or all messages added since that copy had been subsequently read and deleted.

To protect against down-level data in the case where the data set was closed normally, the queue manager performs a number of actions:

1. The queue manager saves a copy of the space map time stamp in the SMDS object within Db2 when the data set is closed normally.
2. The queue manager then checks the space map time stamp is the same, when the data set is opened again

If the time stamp does not match, this suggests that a down-level copy of the data set might have been used, so the queue manager ignores the existing space map and rebuilds it, which will succeed only if no message data was actually lost.

Note: These integrity checks do not guarantee to detect a down-level or damaged data set in all theoretically possible cases. For example, they will not detect a case where the start of a message block is valid but the rest of the data has been partly overwritten.

Recovery scenarios for shared message data sets

This section described shared message data set recovery scenarios.

Data set recovery where no data was lost

In some cases, the correct contents of a failed data set can be restored without needing actual recovery. One example is where a data set contains residual data from a previous use and has not been preformatted again, which can be fixed by preformatting it. Another case is when a data set has been moved, but there was an error in the process of copying the data across, which can be fixed by copying the data again correctly.

In such cases, the corrected data set can be made available again by using the **RESET SMDS** command to set **STATUS(RECOVERED)**. If the availability is currently **ACCESS(SUSPENDED)** this will automatically set it back to **ACCESS(ENABLED)**.

When the owning queue manager is notified that the data set has been recovered, it scans the structure contents to reconstruct the space map, then changes the status to **STATUS(ACTIVE)**. The other queue managers can then start reading the data set again.

Data set recovery with TYPE(NORMAL)

If the contents of a data set have been lost, but the application structure was defined with **RECOVER(YES)** and the appropriate recovery logs are available, the **RECOVER CFSTRUCT** command can be used to recover any persistent messages stored in the structure including persistent

message data offloaded to shared message data sets. This command restores the current state using information logged by the **BACKUP CFSTRUCT** command plus all logged changes to persistent messages since the backup time.

The **RECOVER CFSTRUCT** command always recovers all persistent messages in the coupling facility structure together with offloaded message data stored in Db2. For offloaded data stored in shared message data sets, each data set is only selected for recovery processing if it is already marked as **STATUS(FAILED)** or if it is found to be unexpectedly empty or otherwise invalid when opened by recovery processing. Any shared message data set which is marked as active and which passes the validation checks does not need to be recovered, as the existing message data is already correct, but the header is updated to indicate that any saved space map will need to be rebuilt after recovery.

Recovery processing is only possible when the structure has been marked as failed, as the complete contents of the structure need to be reconstructed by recovery processing. However, if at least one shared message data set has been marked as failed the **RECOVER CFSTRUCT** command will automatically mark the structure as failed if necessary to allow recovery processing to proceed.

Recovery may be performed from any queue manager in the queue-sharing group, provided that it has been given write access to the relevant data sets.

Only persistent messages are backed up and logged, so normal recovery processing will restore all persistent messages, but will cause any non-persistent messages in the structure to be lost.

When recovery has completed, any data set which was selected for recovery is automatically changed to **STATUS(RECOVERED)**, and if the availability was **ACCESS(SUSPENDED)** it is changed to **ACCESS(ENABLED)**. The queue manager rebuilds the space map for each data set by scanning the messages in the coupling facility, then marks the data set as **STATUS(ACTIVE)** so that it can be used again.

Data set recovery with TYPE(PURGE)

For a recoverable structure, if the data set contents have been lost, but recovery is not possible for some reason, for example because recovery logs are not available or recovery would take too long, the **RECOVER CFSTRUCT** command can be used with **TYPE(PURGE)** to get the structure back to a usable state. This resets the structure to the empty state and marks all of the associated data sets as **STATUS(EMPTY)**.

Deleting the application structure

If a non-recoverable application structure is deleted using the MVS **SETXCF FORCE** command, or as a result of structure failure, then the next time the structure is connected, message CSQE028I is issued to say that the structure has been reset and all existing messages have been discarded, and any existing data sets are automatically reset to **STATUS(EMPTY)** as well. This action makes a non-recoverable structure usable again after loss of data either in the structure or in any of the associated data sets.

If a recoverable application structure is deleted, it will be treated in the same way as if the structure had failed.

Data set recovery fails

If **RECOVER CFSTRUCT** cannot complete for some reason, for example because a log data set is no longer available, or because the queue manager terminated while recovery was in progress, then any data set for which recovery was at least started will be marked in the header to show that partial recovery has been attempted, and the data set will be left in the **STATUS(FAILED)** state.

In this case, the options are to repeat the original recovery request or to recover with **TYPE(PURGE)** instead, discarding the existing data.

If an attempt is made to mark the data set as **STATUS(RECOVERED)** without actually recovering it, then the next time it is opened the queue manager will see that the header indicates incomplete recovery and mark it as **STATUS(FAILED)** again.

Off site disaster recovery

For off site disaster recovery, persistent shared messages can be re-created using only the logs and the Db2 shared objects containing the CFSTRUCT definitions and associated SMDS status information.

After setting up the Db2 tables containing the definitions, the application structure and the shared message data sets can be set up as empty. When a queue manager connects to them and finds that they are unexpectedly empty, it will mark them as failed, after which a single **RECOVER CFSTRUCT** command can be used to recover all persistent messages for all affected structures.

SMDS related commands:

This topic describes and provides access to the commands relating to shared message data sets.

Display and alter the **CFSTRUCT** options relating to large message offload (**OFFLOAD** and offload rules) and shared message data sets (**DSGROUP**, **DSBLOCK**, **DSBUFS**, **DSEXPAND**):

- DISPLAY CFSTRUCT
- DEFINE CFSTRUCT
- ALTER CFSTRUCT
- DELETE CFSTRUCT

Display **CFSTRUCT** status relating to large message offload (**OFFLDUSE**):

- DISPLAY CFSTATUS

Display and alter override data set options (**DSEXPAND** and **DSBUFS**) for individual queue managers:

- DISPLAY SMDS
- ALTER SMDS

Display or modify the status and availability of the data sets within the queue-sharing group:

- DISPLAY CFSTATUS TYPE(SMDS)
- RESET SMDS

Display SMDS data set space usage and buffer usage information for a queue manager:

- DISPLAY USAGE TYPE(SMDS)

Display or modify the status and availability of the connections (**SMDSCONN**) to the data sets from an individual queue manager:

- DISPLAY SMDSCONN
- START SMDSCONN
- STOP SMDSCONN

Backup and recover shared messages, including large message data in SMDS when necessary:

- BACKUP CFSTRUCT
- RECOVER CFSTRUCT

Advantages of using shared queues:

Shared queue allows for IBM MQ applications to be scalable, highly available, and allows workload balancing to be implemented.

The advantages of shared queues

The shared queue architecture, where cloned servers pull work from a single shared queue, has some useful properties:

- It is scalable, by adding new instances of the server application, or even adding a new z/OS image with a queue manager (in the queue-sharing group) and a copy of the application.
- It is highly available.
- It naturally performs *pull* workload balancing, based on the available processing capacity of each queue manager in the queue-sharing group.

Using shared queues for high availability

The following examples illustrate how you can use a shared queue to increase application availability.

Consider an IBM MQ scenario where client applications running in the network want to make requests of server applications running on z/OS. The client application constructs a request message and places it on a request queue. The client then waits for a reply from the server, sent to the reply-to queue named in the message descriptor of the request message.

IBM MQ manages the transportation of the request message from the client machine to the server's input queue on z/OS and of the response from the server back to the client. By defining the server's input queue as a shared queue, any messages put to the queue can be retrieved on any queue manager in the queue-sharing group. This means that you can configure a queue manager on each z/OS image in the sysplex and, by connecting them all to the same queue-sharing group, any one of them can access messages on the server's input queue.

Messages on the input queue of the server are still available, even if one of the queue managers terminates abnormally or you have to stop it for administrative reasons. You can take an entire z/OS image offline and the messages will still be available.

To take advantage of this availability of messages on a shared queue, run an instance of the server application on each z/OS image in the sysplex to provide higher server application capacity and availability, as shown in Figure 60 on page 344.

One instance of the server application retrieves a request message from the shared queue and, based on the content, performs its processing, producing a result that is sent back to the client as an IBM MQ message. The response message is destined for the reply-to queue and reply-to queue manager named in the message descriptor of the request message.

There are a number of options that you can use to configure the return path. For more information about these options, see “Distributed queuing and queue-sharing groups” on page 363.

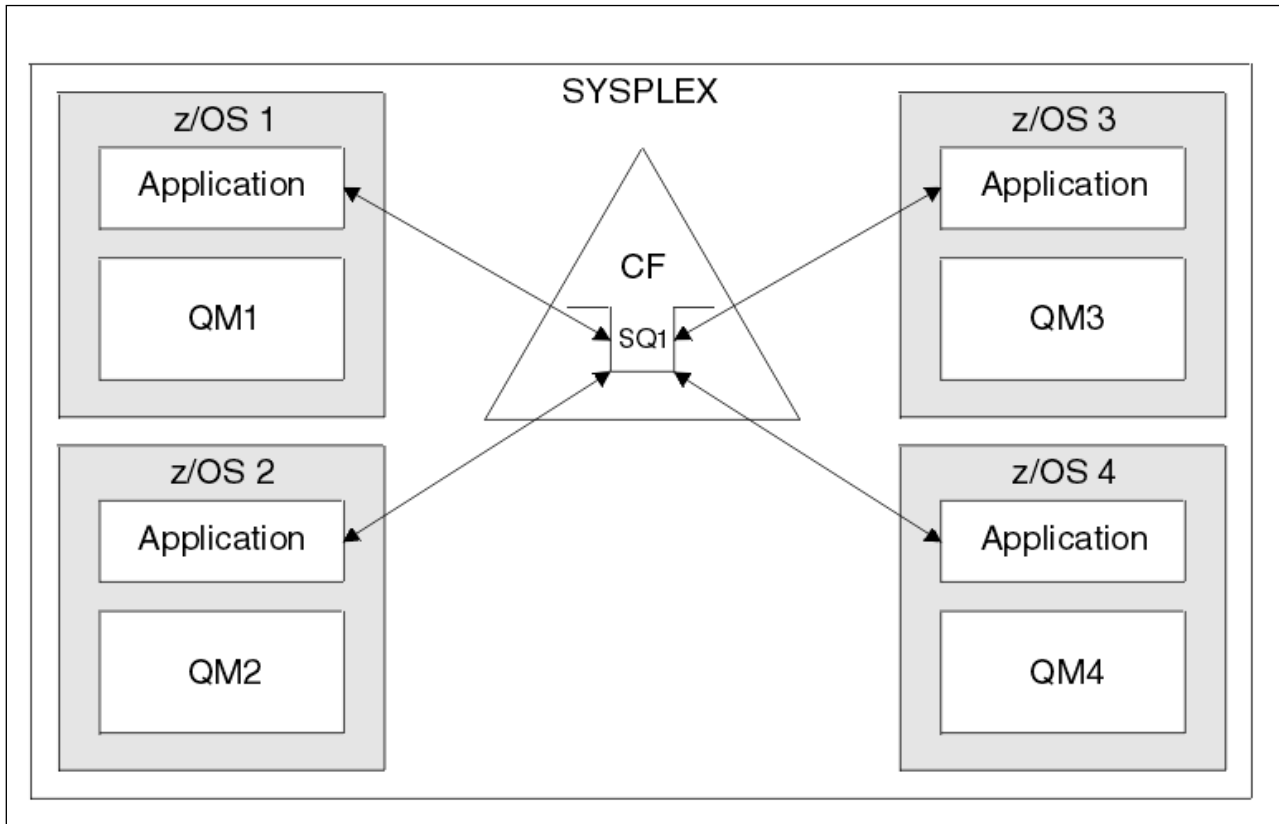


Figure 60. Multiple instances of an application servicing a shared queue

Peer recovery

To further enhance the availability of messages in a queue-sharing group, IBM MQ detects if another queue manager in the group disconnects from the coupling facility abnormally and completes units of work for that queue manager that are still pending, where possible. This feature is known as *peer recovery*.

Suppose a queue manager terminates abnormally at a point where an application has retrieved a request message from a queue in sync point, but has not yet put the response message or committed the unit of work. Another queue manager in the queue-sharing group detects the failure, and backs out the in-flight units of work being performed on the failed queue manager. This means that the request message is put back on to the request queue and is available for one of the other server instances to process, without waiting for the failed queue manager to restart.

If IBM MQ cannot resolve a unit of work automatically, you can resolve the shared portion manually to enable another queue manager in the queue-sharing group to continue processing that work.

Use of storage class memory with shared queues:

The use of storage class memory (SCM) can be advantageous when used with IBM MQ for z/OS shared queues.

The z13, zEC12, and zBC12 machines allow the installation of Flash Express cards. These cards contain flash solid-state drives (SSD). After installation, flash storage from the cards can be allocated to one or more LPARs where it is typically known as SCM.

SCM sits between real storage and direct access storage device (DASD) in terms of both I/O latency and cost. Because SCM has no moving parts, it exhibits much lower I/O latencies than DASD.

SCM is also much cheaper than real storage. As a result, a large amount of storage can be installed for a relatively low cost; for example, a pair of Flash Express cards contains 1424 GB of usable storage.

These characteristics mean that SCM is useful when a large amount of data must be taken from real storage in a short period of time, because that data can be written to SCM much quicker than it can be written to DASD. This specific point can be very useful when using coupling facility (CF) list structures containing IBM MQ shared queues.

Why list structures fill up

When a CF structure is defined, it is configured with a `SIZE` attribute that describes the maximum size of the structure. Because CF structures are always permanently resident in real storage, the sum of the `SIZE` attributes of the structures that are defined on a CF should be less than the amount of the real storage allocated to the CF.

As a result, there is a constant pressure to keep the `SIZE` value of any given structure to the minimum value possible, so that more structures can fit into the CF. However, ensuring structures are large enough to achieve their purpose can result in a conflicting pressure, because making a structure too small means that it might fill up, disrupting the applications or subsystems making use of it.

There is a strong need to accurately size a structure based on its expected usage. However, this task is difficult to do because workloads can change over time and accounting for their fluctuations is not easy.

IBM MQ shared queues use CF list structures to store messages. IBM MQ calls CF structures, which contain messages and application structures.

Application structures are referenced using the information stored in IBM MQ `CFSTRUCT` objects. When a message that is smaller than 63 KB is put on a shared queue, the message is stored entirely in an application structure as a single list entry, and zero or more list elements.

Because IBM MQ shared queues use list structures, the pressures described also affect shared queues. In this case, the maximum number of messages that can be stored on a shared queue is a function of the:

- Size of the messages on the queue
- Maximum size of the structure
- Number of entries and elements available in the structure

Because up to 512 shared queues can use the same structure, and effectively compete for entries and elements, this complicates matters even further

IBM MQ queues are used for the transfer of data between applications, so a common situation is an application putting messages to a queue, when the partner application, which should be getting those messages, is not running.

When this happens the number of messages on the queue increase over time until one or more of the following situations occur:

- The putting application stops putting messages.
- The getting application starts getting messages.
- Existing messages on the queue start expiring, and are removed from the queue.
- The queue reaches its maximum depth in which case an MQRC_Q_FULL reason code is returned to the putting application.
- The structure containing the shared queue reaches its maximum size, or the CF containing the structure runs out of available storage. In either case, an MQRC_STORAGE_MEDIUM_FULL reason code is returned to the putting application.

In the last three cases the queue is full. At this point the putting application has a problem because there is nowhere for its messages to go. The putting application typically solves this problem by using one or more of the following solutions:

- Repeatedly retry putting the message, optionally with a delay between retries.
- Put the messages somewhere else, such as a database or a file. The messages can be accessed later and put to the queue as normal.
- Discard the message if it is nonpersistent.

However, for some classes of applications, for example those with a large volume of incoming messages, or no access to a file system, these solutions are not practical. There is a real need to ensure that queues never, or are extremely unlikely to, fill up in the first place and this is especially pertinent to shared queues.

SMDS and offload rules

The offload rules introduced in IBM WebSphere MQ Version 7.1 provide a way of reducing the likelihood of an application structure filling up.

Each application structure has three rules associated with it, specified using three pairs of keywords:

- OFFLD1SZ and OFFLD1TH
- OFFLD2SZ and OFFLD2TH
- OFFLD3SZ and OFFLD3TH

Each rule specifies the conditions that must be met for message data to be offloaded to the storage mechanism that is associated with the application structure. Two types of storage mechanisms are currently available:

- Db2
- Group of Virtual Storage Access Method (VSAM) linear data sets, which IBM MQ calls a shared message data set (SMDS).

The following example shows the MQSC command to create an application structure named LIST1, using the DEFINE CFSTRUCT command.

This structure has the default offload rules in place, and uses SMDS as the offload mechanism. This means that when the structure is 70% full (OFFLD1TH), all messages that are 32 KB or larger (OFFLD1SZ) are offloaded to SMDS.


Similarly, when the structure is 80% full (OFFLD2TH) all messages that are 4 KB or larger (OFFLD2SZ) are offloaded. When the structure is 90% full (OFFLD3TH) all messages (OFFLD3SZ) are offloaded.

```
DEFINE CFSTRUCT(LIST1)
CFLEVEL(5)
OFFLOAD(SMDS)
OFFFLD1SZ(32K) OFFFLD1TH(70)
OFFFLD2SZ(4K) OFFFLD2TH(80)
OFFFLD3SZ(0K) OFFFLD3TH(90)
```

An offloaded message is stored in the offload medium, and a pointer to the message is stored in the structure. While the offload rules reduce the chance of the structure filling up, by putting less message data in the structure as it runs out of storage, some data is still written to the structure for each message. That is, the pointer to the offloaded message.

Additionally, the offload rules come with a performance cost. Writing a message to a structure is relatively quick and is largely dominated by the time spent to send the request for the write to the CF. The actual writing to the structure is fast, happening at real storage speeds.

Writing a message to SMDS is much slower because it includes writing to the structure for the message pointer, and writing the message data to SMDS. This second write operation is done at DASD speed and has the potential to add latency. If Db2 is used as the offload mechanism the performance cost is much greater.

How storage class memory works with IBM MQ for z/OS: 

An overview of the use of storage class memory (SCM) with IBM MQ for z/OS shared queues.

A coupling facility (CF) that is at CFLEVEL 19, or greater, can have SCM allocated to it. The structures defined in that CF can then be configured to make use of SCM to reduce the chances of the structures filling up (known as a structure full condition). When a structure configured to make use of SCM fills up past a system-determined point, the CF starts moving data from the structure into SCM, which frees space in the structure for new data.

Note: Because SCM itself can fill up, allocating SCM to a structure reduces only the likelihood of a structure full condition, but does not entirely remove the chance of one occurring.

A structure is configured to use SCM by specifying both the **SCMALGORITHM** and **SCMMAXSIZE** keywords in the coupling facility resource manager (CFRM) policy, containing the definition of that structure.

Note that after these keywords are specified, and the CFRM policy is applied, the structure must be rebuilt, or deallocated so that they can take effect.

SCMALGORITHM

Because the input/output speed of SCM is slower than that of real storage, the CF uses an algorithm that is tailored to the expected use of the structure in order to reduce the impact of writing to, or reading from, SCM.

The algorithm is configured by the **SCMALGORITHM** keyword in the CFRM policy for the structure, using the *KEYPRIORITY1* value. Note that you should use the *KEYPRIORITY1* value only with list structures used by IBM MQ shared queues.

The *KEYPRIORITY1* algorithm works by assuming that most applications will get messages from a shared queue in priority order; that is, when an application gets a message, it gets the oldest message with the highest priority.

When a structure starts to fill past the system-defined threshold of 90%, the CF starts asynchronously migrating messages that are least likely to be got next. These are messages with lower priorities that were more recently put on the queue.

This asynchronous migration of messages from the structure into SCM is known as "pre-staging".

Pre-staging reduces the performance cost of using SCM because it reduces the chance of an application being blocked during the occurrence of synchronous input/output to SCM.

In addition to pre-staging, the *KEYPRIORITY1* algorithm also asynchronously brings back messages from SCM and into the structure when sufficient free space is available. For the *KEYPRIORITY1* algorithm, this means that when the structure is less than or equal to 70% full.

The act of bringing messages from SCM into the structure is known as "pre-fetching".

Pre-fetching reduces the likelihood of an application trying to get a message that has been pre-staged to SCM and having to wait while the CF synchronously brings back the message into the structure.

SCMMAXSIZE

The **SCMMAXSIZE** keyword defines the maximum amount of SCM that can be used by a structure. Because SCM is allocated to the structure by the CF when it is required, it is possible to specify a **SCMMAXSIZE** that is greater than the total amount of free SCM available. This is known as "over-committing".

Important: Never over-commit SCM. If you do, the applications that are relying on it will not obtain the behavior that they expect. For example, IBM MQ applications using shared queues might get unexpected MQRC_STORAGE_MEDIUM_FULL reason codes.

The CF uses various data structures to track its use of SCM. These data structures reside in the real storage that is allocated to the CF and, as a result, reduce the amount of real storage that can be used by structures. The storage used by these data structures is known as "augmented space".

When a structure is configured with SCM, a small amount of real storage is allocated from the CF to the structure known as fixed augmented space. This is allocated even if the structure never actually uses any SCM. As data from the structure is stored into SCM, extra dynamic augmented space will be allocated from the spare real storage in the CF.

When the data is removed from SCM, the dynamic augmented space is returned to the CF. Augmented space, either fixed or dynamic, is never taken from the real storage that is allocated to a structure.

In addition to augmented storage, when a structure is configured to use SCM, the amount of control storage used by that structure increases. This means that a list structure configured with SCM can contain fewer entries and elements than a structure of the same size without SCM being configured.

To understand the impact of SCM on new or existing structures, use the CFSizer tool.

A final important point to note is that after data is moved from the structure into SCM, and dynamic augmented space has been used, the structure cannot be altered either manually or automatically.

That is, the amount of storage allocated to the structure cannot be increased or decreased, the entry-to-element ratio that is used by the structure cannot be changed, and so on. To make the structure alterable again, the structure must not have any data stored in SCM and must not be making use of dynamic augmented storage.

Emergency storage and improved performance are two use cases for using SCM with IBM MQ for z/OS.

This section introduces the theory behind the two possible scenarios. For further details on how you set up the scenarios, see:

- “Emergency storage - basic configuration” on page 352
- “Improved performance - basic configuration” on page 358

Important: The use of SCM with CF structures is not dependent on any specific version of IBM MQ. However the emergency storage scenario works only with IBM WebSphere MQ Version 7.1 and later, because it requires SMDS and the offload rules.

Emergency storage

SMDS and message offloading can be used in conjunction with SCM to reduce the likelihood of an MQRC_STORAGE_MEDIUM_FULL reason code being returned to an IBM MQ application during an extended outage.

Overview

A single shared queue is configured on an application structure. The putting application puts messages onto the shared queue; the getting application gets messages from the shared queue.

During normal running, the queue depth is expected to be close to zero, but a business requirement indicates that the system must be able to tolerate a two-hour outage of the getting application. This means that the shared queue must be able to contain two hours of messages from the putting application.

Currently, this process is achieved by using the default offload rules, and SMDS, so that the size of the structure is minimized, while reducing the performance cost that is associated with offloading.

The rate of messages being sent to the shared queue is expected to double in the short to medium term. Although the requirement that the system be able to tolerate a two-hour outage still exists, not enough real storage is available in the CF to double the size of the structure.

Because the CF, which contains the application structure, resides on a zEC12 machine, the possibility exists to associate sufficient SCM with the structure to store enough messages so that a two-hour outage can be tolerated.

Consider what happens over a period of time:

1. Initially, the system is in its steady state. Both the putting and getting application are running normally and the queue depth is close to, or at, zero. The result is that the application structure is largely empty.
2. At a certain time, the getting application suffers an unexpected failure and stops. The putting application continues to put messages to the queue and the application structure starts to fill up.
3. After the structure reaches 70% full, the conditions of the first offload rule are met and all messages with a size greater than or equal to 32 KB are offloaded to SMDS.

See “SMDS and offload rules” on page 346 for an overview of the offload rules.

4. As messages continue to be put to the shared queue, the structure continues to fill (either because of the message data being stored in the structure, or as a result of the pointers to the offloaded messages being stored in the structure).

When the structure reaches 80% full, the second offload rule starts to apply and messages that are 4 KB or greater are offloaded to SMDS.

5. When the structure is past 90% full, all messages are offloaded to SMDS and only the message pointers are being placed in the structure.

About this time, the pre-staging algorithm starts to run, and begins moving data from the structure into SCM. Assuming all messages on the queue are the same priority, the newest messages are pre-staged.

Because all messages are now being offloaded to SMDS, the data being moved into SCM is not actual message data, but instead the pointers to the messages on SMDS.

As a result, the number of messages that can be stored on the combination of the structure, and the SCM and SMDS associated with the structure, is very large.

Performance: During this stage of the outage, the putting application can suffer a degree of performance degradation because of having to write to SMDS. In this case, the use of SCM should not be a limiting factor on the putting application in terms of performance. SCM provides extra space to prevent the structure filling up.

6. Eventually the getting application is available again and the outage is over.

However SCM is still being used by the structure. The getting application starts reading messages off the queue, getting the oldest, highest priority messages first.

Because these messages were written before the structure started to fill up, they come out entirely from the real storage portion of the structure.

7. As the structure starts to empty, it goes below the threshold at which pre-staging is active, and so pre-staging stops.

8. The structure usage reduces below the point at which the offload rules take effect, so messages are no longer offloaded to SMDS unless they are more than 63 KB.

At about this time, the pre-fetch algorithm starts moving data from SCM in to the structure. Because the getting application gets messages from the queue in the order expected by the SCM algorithms, messages are brought in before the getting application needs them.

The result is that the getting application never needs to wait for messages to be brought in synchronously from SCM.

9. As the getting application continues to move down the queue, it starts retrieving messages that were offloaded to SMDS.
10. Finally, the system is in a steady state again. No messages are stored in SCM or SMDS, and the queue depth is close to zero.

Improved performance

This scenario describes using SCM to increase the number of messages that can be stored on a shared queue without incurring the performance cost of using SMDS.

Description

For this scenario, a putting and getting application communicate through a shared queue which is stored in application structure.

The putting application tends to run in bursts, when it puts a large number of messages in a short amount of time. Then, in an extended period of time, it produces no messages at all.

The getting application sequentially processes each message, and performs complex processing on each one. As a result, most of the time the queue depth is zero, except for when the putting application starts to run, where the queue depth starts increasing as messages are being put faster than they are being got.

The queue depth increases until the putting application stops, and the getting application has enough time to process all the messages on the queue.

Notes:

1. In this scenario, the key factor is performance. The messages being sent to the queue are always less than 63 KB and so never need to be offloaded to SMDS.
2. The application structure has been sized so that it is large enough to contain all of the messages that will be placed on it by the putting application in a single "burst."
3. The offload rules must all be disabled so that, even when the structure starts to fill, the messages are not offloaded to SMDS. This is because the performance costs that are associated with writing messages to, and reading messages from, SMDS are deemed unacceptable.

Over time, the number of messages that the putting application send in a burst must increase by several orders of magnitude. Because the getting application must process each message sequentially, the number of messages on the queue increases to the point where the structure fills up.

At this point, the putting application receives a reason code (MQRC_STORAGE_MEDIUM_FULL) when putting a message, and the put operation fails. The putting application can only briefly tolerate periods when it is unable to put messages to the queue. If the period is too long, the application ends.

Assuming that you do not have the time, or skills available, to rewrite either the putting application or getting application, this problem has three possible solutions:

1. Increase the size of the application structure.
2. Add offload rules to the application structure so that messages are offloaded to SMDS as the queue starts to fill up.
3. Associate SCM with the structure.

The first solution is quick to implement, but not enough real storage is available on the CF.

The second solution might also be quick to implement, but the performance impact of offloading to SMDS is considered too significant to use this option.

The third solution, associating SCM with the structure, provides an acceptable balance of cost and performance.

Associating SCM with a structure results in a higher use of real storage in the CF because of the augmented storage that get operations used. However, the actual amount of real storage will be less than the amount used in the first option.

Another consideration is the cost of SCM. However this cost is much cheaper than real storage. These factors combine to make the third option cheaper than the first option.


Although the third option, potentially, might not perform as well as the first option, the pre-fetch and pre-staging algorithms used by the CF can combine to make the differences in performance acceptable, or in some cases negligible.

Certainly the performance can be much better than using SMDS to offload messages.

Consider what happens over a period of time:

1. Initially, the getting application is active and waiting for messages to be delivered to the shared queue. The putting application is not active and the shared queue is empty.
2. At a certain time, the putting application becomes active, and starts putting a large number of messages to the shared queue. The getting application starts getting the messages, but the queue depth rapidly starts to increase because the getting application is slower than the putting application. As a result, the application structure starts to fill up.

3. As the time increases, the putting application is still active. The application structure fills up to approximately 90%.
This is when the SCM pre-staging algorithm starts to move messages from the structure into SCM, freeing space in the structure.
Because the getting application gets the oldest, highest priority messages from the queue first, it is always getting messages from the structure and does not need to wait for messages to be brought synchronously from SCM in to the structure.
4. The putting application is still active and putting messages to the shared queue. However, the application never receives an MQRC_STORAGE_MEDIUM_FULL reason code, because enough space exists in SCM to store all the messages that do not fit in the structure.
5. Eventually, the putting application stops because it has no more messages to put.
The pre-staging algorithm stops because the structure falls below 90% in use, and the getting application continues processing the messages in the queue.
6. As the getting application starts to free space in the structure, the pre-fetch algorithm starts to bring messages back from SCM in to the structure.
Because the getting application processes messages in the order expected by the pre-fetch algorithm, the getting application never becomes blocked waiting for message data to be brought synchronously from SCM in to the structure.
7. Finally, the getting application processes all the messages on the shared queue, and waits until the next message is available. The structure and SCM are empty of messages.

Emergency storage - basic configuration: 

How you set up a basic scenario for emergency storage on IBM MQ.

About this task

SMDS and message offloading can be used in conjunction with SCM to reduce the likelihood of an MQRC_STORAGE_MEDIUM_FULL reason code being returned to an IBM MQ application during an extended outage.

For example, your enterprise has an application that puts messages onto the queue and an application that gets messages from the queue. During normal running, you expect the queue depth to be close to zero, but a business requirement indicates that the system be able to tolerate a two-hour outage of the application that gets the messages.

This means that the shared queue being used must be able to contain two hours of messages from the putting application. Currently you achieve this by using the default offload rules, and SMDS.

You expect the rate of messages being sent to the shared queue to double in the short to medium term. Although your requirement that the system be able to tolerate a two-hour outage still exists, not enough real storage is available in the CF to double the size of the structure. Because the CF containing the application structure resides on a zEC12 machine, you have the capability of associating sufficient SCM with the structure to store enough messages, so that a two-hour outage can be tolerated

This initial scenario uses a:

- Queue-sharing group, IBM1, that contains a single queue manager, CSQ3. In addition to the administration structure, the queue-sharing group defined a single application structure, SCEN1.
- Coupling facility (CF) CF01, in which the SCEN1 application structure is stored as the IBM1SCEN1 structure. This structure has a maximum size of 1 GB.
- Single shared queue, SCEN1.Q that the application structure uses.

This configuration is illustrated in Figure 61 on page 353.

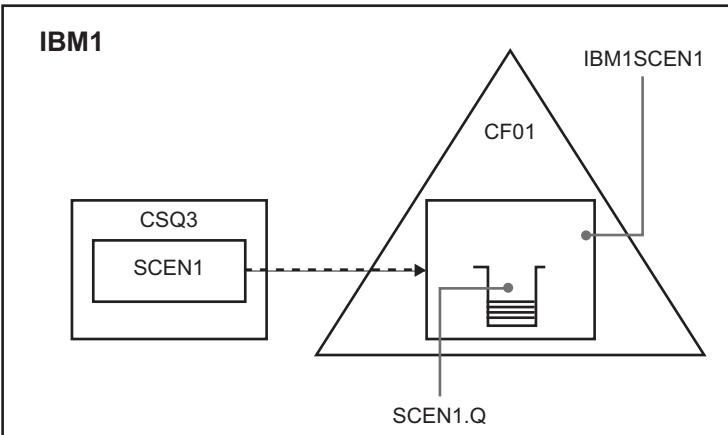


Figure 61. Basic configuration

Furthermore, assume that queue manager CSQ3 is already the only member of queue-sharing group IBM1.

You must add the definition for structure IBM1SCEN1 to the coupling facility resource manager (CFRM) policy. For simplicity, the structure is defined so that it can be created in only a single coupling facility, CF01, by specifying PREFLIST(CF01).

Attention: To allow for high availability in your production system, you should include at least two CFs in the PREFLIST for any structures that are used by IBM MQ.

Procedure

1. Refresh the CFRM policy by using the following command:

```
SETXCF START,POLICY,TYPE=CFRM,POLNAME=IBM1SCEN1
```

Sample CFRM policy for structure IBM1SCEN1:

```
STRUCTURE
NAME(IBM1SCEN1)
SIZE(1024M)
INITSIZE(512M)
ALLOWAUTOALT(YES)
FULLTHRESHOLD(85)
PREFLIST(CF01)
ALLOWREALLOCATE(YES)
DUPLEX(DISABLED)
ENFORCEORDER(NO)
```

2. Verify that the structure has been created correctly, by using the following command:

```
D XCF,STR,STRNAME=IBM1SCEN1
```

At this point, your structure has not been allocated, shown by the STATUS line, to the queue-sharing group.

3. Configure IBM MQ to make use of the structure defined in the CFRM policy.

- a. Use the DEFINE CFSTRUCT command, with the structure name of SCEN1 to create a IBM MQ CFSTRUCT object:

```
DEFINE CFSTRUCT(SCEN1)
CFCONLOS(TOLERATE)
CFLEVEL(5)
DESCR('Structure for SCM scenario 1')
RECOVER(NO)
RECAUTO(YES)
```

```
OFFLOAD(DB2)
OFFLD1SZ(64K) OFFLD1TH(70)
OFFLD2SZ(64K) OFFLD2TH(80)
OFFLD3SZ(64K) OFFLD3TH(90)
```

- b. Validate the structure, using the DISPLAY CFSTRUCT command.
- c. Define the SCEN1.Q shared queue, to use the SCEN1 structure, using the following MQSC command:
DEFINE QLOCAL(SCEN1.Q) QSGDISP(SHARED) CFSTRUCT(SCEN1) MAXDEPTH(999999999)
4. Use IBM MQ Explorer to put a single message to the queue SCEN1.Q and take the message off again.
5. Issue the following command to check that the structure is now allocated:
D XCF,STR,STRNAME=IBM1SCEN1

Check in the output from the command, that the STATUS line shows ALLOCATED.

Results

You have created the basic configuration. You can now obtain an idea of the baseline performance of your configuration using whatever method you select.


What to do next

Add SMDS and SCM to the initial structure

Related concepts:

“Use of storage class memory with shared queues” on page 345

The use of storage class memory (SCM) can be advantageous when used with IBM MQ for z/OS shared queues.

Adding SMDS and SCM to the initial structure: 

How you add SMDS and SCM for emergency storage on IBM MQ.

About this task

This part of the task uses the basic configuration described in “Emergency storage - basic configuration” on page 352. The scenario describes the addition of shared message data sets (SMDS), and then of SCM to the initial structure.

This final configuration is illustrated in Figure 62 on page 355.

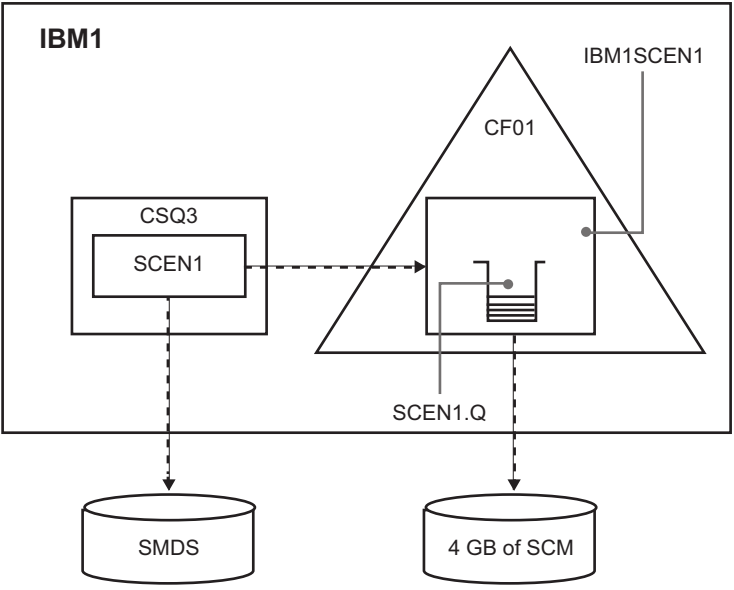


Figure 62. Configuration adding SMDS and SCM for emergency storage

Procedure

1. Create the SMDS data set that the SCEN1 application structure uses, by editing the **CSQ4SMDS** sample JCL, as shown:

```
//CSQ4SMDS JOB NOTIFY=&SYSUID
/**
/** Allocate SMDS
/**
//DEFINE EXEC PGM=IDCAMS,REGION=4M
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
DEFINE CLUSTER          -
(NAME(CSQSMDS.SCEN1.CSQ3.SMDS) -
MEGABYTES(5000 3000)   -
LINEAR                 -
SHAREOPTIONS(2 3)     -
DATA                   -
(NAME(CSQSMDS.SCEN1.CSQ3.SMDS.DATA) )
/**
/**
/** Format the SMDS
/**
//FORM EXEC PGM=CSQJUFMT,COND=(0,NE),REGION=0M
//STEPLIB DD DSN=MQ800.SCSQANLE,DISP=SHR
// DD DSN=MQ800.SCSQAUTH,DISP=SHR
//SYSUT1 DD DISP=OLD,DSN=CSQSMDS.SCEN1.CSQ3.SMDS
//SYSPRINT DD SYSOUT=*
```

2. Issue the ALTER CFSTRUCT command to change the SCEN1 application structure, to use SMDS for offloading, implementing the default offload rules:

```
ALTER CFSTRUCT(SCEN1) OFFLOAD(SMDS) OFFLD1SZ(32K) OFFLD2SZ(4K) OFFLD3SZ(0K)
DSGROUP('CSQSMDS.SCEN1.*.SMDS') DSBLOCK(1M)
```

Note the following:

- Because SCEN1.Q is the only shared queue on the SCEN1 application structure, the **DSBLOCK** value has been set to 1M, the largest value possible. This should be the most efficient setting for our scenario.
- Because the messages sent by the putting application are 30 KB, offloading to SMDS does not start until the second offload rule is met, when the structure is 80% full.

3. Run your test application again. Note the increased storage of messages on the queue.

4. Add 4 GB of SCM to structure IBM1SCEN1 by carrying out the following procedure:
 - a. Check how much SCM is installed, and allocated to CF01, by issuing the following command:
D CF,CFNAME=CF01
 - b. Check the STORAGE-CLASS MEMORY figures in the STORAGE CONFIGURATION section of the displayed output to see the available storage.
 - c. Update the CFRM policy with the SCMMAXSIZE and SCMALGORITHM keywords as shown:



```
STRUCTURE
NAME(IBM1SCEN1)
SIZE(1024M)
INITSIZE(512M)
ALLOWAUTOALT(YES)
FULLTHRESHOLD(85)
PREFLIST(CF01)
ALLOWREALLOCATE(YES)
DUPLEX(DISABLED)
ENFORCEORDER(NO)
SCMMAXSIZE(4G)
SCMALGORITHM(KEYPRIORITY1)
```
5. Activate the CFRM policy by issuing the following command:
SETXCF START,POLICY,TYPE=CFRM,POLNAME=polname
6. Rebuild the IBM1SCEN1 structure. You must carry out this procedure because the structure was allocated when you made the previous changes.
Issue the following command to rebuild the structure:
SETXCF START,REBUILD,STRNM=IBM1SCEN1

Results

You have successfully added SCM to your configuration.

What to do next

Optimize the performance of your system. See “Optimizing storage class memory usage” for more information.

Optimizing storage class memory usage: 

How you improve your use of storage class memory (SCM).

Run the following command:

```
D XCF,STR,STRNAME=IBM1SCEN1
```

As the structure was already full with message data, because of the previous tests, part of the rebuild involved pre-staging some of the messages from the structure into SCM. This process was initiated by using the previous command.

The output from this command produces, for example:

```
ACTIVE STRUCTURE
-----
ALLOCATION TIME: 06/17/2014 09:28:50
CFNAME : CF01
COUPLING FACILITY: 002827.IBM.02.00000000B8D7
PARTITION: 3B CPCID: 00
STORAGE CONFIGURATION ALLOCATED MAXIMUM %
ACTUAL SIZE: 1024 M 1024 M 100
AUGMENTED SPACE: 3 M 142 M 2
STORAGE-CLASS MEMORY: 88 M 4096 M 2
ENTRIES: 120120 1089536 11
ELEMENTS: 240240 15664556 1
```

```

SPACE USAGE IN-USE TOTAL %
ENTRIES: 84921 219439 38
ELEMENTS: 2707678 3149050 85
EMCS: 2 282044 0
LOCKS: 1024
SCMHIGHTHRESHOLD : 90
SCMLOWTHRESHOLD : 70
ACTUAL SUBNOTIFYDELAY: 5000
PHYSICAL VERSION: CD5186A0 2BD8B85C
LOGICAL VERSION: CD515C50 CE2ED258
SYSTEM-MANAGED PROCESS LEVEL: 9
XCF GRPNAME : IXCL0053
DISPOSITION : KEEP
ACCESS TIME : NOLIMIT
MAX CONNECTIONS: 32
# CONNECTIONS : 1
CONNECTION NAME ID VERSION SYSNAME JOBNAME ASID STATE
-----
CSQEIBM1CSQ301 01 00010059 SC61 CSQ3MSTR 0091 ACTIVE

```

Note the following from the output of the command:

- That `STORAGE_CLASS MEMORY` provides confirmation that a **MAXIMUM** of 4096 MB of SCM has been added to the structure.
- The `ALLOCATED` figure for the amount of `STORAGE-CLASS MEMORY` used for pre-staging. There is now free space in the structure where there was none before SCM was added.
- The amount of `AUGMENTED SPACE` used to track SCM usage.
- The point at which the pre-staging algorithm starts to move data from the structure into SCM is when the structure is 90% full. This is indicated by the non-configurable **SCMHIGHTHRESHOLD** property.
- The point below which the prefetching algorithm starts to move data from SCM into the structure is when the structure is 70% full. This is indicated by the non-configurable **SCMLOWTHRESHOLD** property.

You can now test various ways to optimize the use of SCM. Note the following:

- After SCM is used to store messages, you cannot alter the structure until you have removed all the data from SCM.

In this case, that means that the entry-to-element ratio is frozen at the value that was in place when SCM was first used. You must carefully ensure that the structure is in the state you want, before the pre-staging algorithm starts moving data into SCM.

- Is the current structure size correct before using SCM?

For example, have you increased **INITSIZE** from 512 MB to a `SIZE` of 1 GB?

If you do not do this, it is possible that although you enabled your structure for auto-alteration, the pre-staging algorithm will start to move data into SCM before the alteration has a chance to start. As a result, the structure is frozen using 512 MB of real storage.

- Is the entry-to-element ratio correct before using SCM?

The goal of this scenario is to increase the number of offloaded message pointers that can be stored in the structure and SCM as a whole, as well as keeping as many messages entirely in structure storage as possible. Accessing these messages is faster than accessing messages on SMDS.

Therefore, you need to have a structure that starts with an entry-to-element ratio that is good for storing messages, and then transitions to a ratio that is good for storing message pointers before the prestage algorithm first starts. This transition can be achieved, in part, by making use of the IBM MQ offload rules.

Change the offload rules by issuing the following command:

```
ALTER CFSTRUCT(SCEN1) OFFLD1SZ(0K)
```

You might have to carry out several runs to optimize the entry-to-element ratios.


The following table shows possible improvements in the number of messages put on the queue during the different phases of the emergency storage scenario.

Table 22. Comparison of results for the emergency storage scenario

Test description	Number of messages	Time to fill queue (seconds)
Basic configuration	27,850	3.2
SMDS with default offload rules	205,000	158
SCM with default offload rules	828,610	469
SCM with adjusted offload rules	1,135,775	679

The last row in the table shows that adjusting the offload rules had the required effect.

You need to examine your system to see if you can improve on these figures in any way. For example, you might run out of available SMDS storage. If you can allocate more SMDS storage you should be able to increase the number of messages on the queue quite significantly.

Improved performance - basic configuration: 

How you set up a basic scenario for improved performance using shared queues on IBM MQ.

About this task

This scenario describes the use of SCM to increase the number of messages that can be stored on a shared queue without incurring the performance cost of using SMDS.

This initial scenario is very similar to that used for emergency storage and uses a:

- Queue-sharing group, IBM1, that contains a single queue manager, CSQ3. In addition to the administration structure, the queue-sharing group defined a single application structure, SCEN2.
- Coupling facility (CF) CF01, in which the SCEN2 application structure is stored as the IBM1SCEN2 structure. This structure has a maximum size of 2 GB.
- Single shared queue, SCEN2.Q, which is configured to use the application structure.

This configuration is illustrated in Figure 63.

Figure 63. Basic configuration

Furthermore, assume that queue manager CSQ3 is already the only member of queue-sharing group IBM1.

You must add the definition for structure IBM1SCEN2 to the coupling facility resource manager (CFRM) policy. For simplicity, the structure is defined so that it can be created in only a single coupling facility, CF01, by specifying PREFLIST(CF01).

Sample CFRM policy for structure IBM1SCEN2:

```
STRUCTURE
NAME(IBM1SCEN2)
SIZE(2048M)
INITSIZE(2048M)
ALLOWAUTOALT(YES)
FULLTHRESHOLD(85)
PREFLIST(CF01)
ALLOWREALLOCATE(YES)
DUPLEX(DISABLED)
ENFORCEORDER(NO)
```

Both the **INITSIZE** and **SIZE** keywords have the value 2048M so that the structure cannot resize.

Procedure

1. Refresh the CFRM policy by using the following command:
SETXCF START,POLICY,TYPE=CFRM,POLNAME=IBM1SCEN2
2. Verify that the structure has been created correctly, by using the following command:
D XCF,STR,STRNAME=IBM1SCEN2

Issuing the preceding command gives the following output:

```
RESPONSE=SC61
IXC360I 07.58.51 DISPLAY XCF 581
STRNAME: IBM1SCEN2
STATUS: NOT ALLOCATED
POLICY INFORMATION:
POLICY SIZE : 2048 M
POLICY INITSIZE: 2048 M
POLICY MINSIZE : 1536 M
FULLTHRESHOLD : 85
ALLOWAUTOALT : YES
REBUILD PERCENT: N/A
DUPLEX : DISABLED
ALLOWREALLOCATE: YES
PREFERENCE LIST: CF01
ENFORCEORDER : NO
EXCLUSION LIST IS EMPTY
```

```
EVENT MANAGEMENT: MESSAGE-BASED   MANAGER SYSTEM NAME: SC53
MANAGEMENT LEVEL : 01050107
```

At this point, your structure has not been allocated, shown by the STATUS line, to the queue-sharing group.

3. Configure IBM MQ to make use of the structure defined in the CFRM policy.
 - a. Use the DEFINE CFSTRUCT command, with the structure name of SCEN2 to create a IBM MQ CFSTRUCT object.

```
DEFINE CFSTRUCT(SCEN2)
CFCONLOS(TOLERATE)
CFLEVEL(5)
DESCR('Structure for SCM scenario 2')
RECOVER(NO)
RECAUTO(YES)
OFFLOAD(DB2)
OFFLD1SZ(64K) OFFLD1TH(70)
OFFLD2SZ(64K) OFFLD2TH(80)
OFFLD3SZ(64K) OFFLD3TH(90)
```
 - b. Check the structure, using the DISPLAY CFSTRUCT command.
 - c. Define the SCEN2.Q shared queue, to use the SCEN2 structure, using the following MQSC command:

```
DEFINE QLOCAL(SCEN2.Q) QSGDISP(SHARED) CFSTRUCT(SCEN2) MAXDEPTH(999999999)
```
4. Use IBM MQ Explorer to put a single message to the queue SCEN2.Q and take the message off again.
5. Issue the following command to check that the structure is now allocated:

```
D XCF,STR,STRNAME=IBM1SCEN2
```

Review the output from the command, a portion of which is shown, and ensure that the STATUS line shows ALLOCATED.

```
RESPONSE=SC61
IXC360I 08.31.27 DISPLAY XCF 703
STRNAME: IBM1SCEN2
STATUS: ALLOCATED
EVENT MANAGEMENT: MESSAGE-BASED
TYPE: SERIALIZED LIST
```

POLICY INFORMATION:
POLICY SIZE : 2048 M
POLICY INITSIZE: 2048 M
POLICY MINSIZE : 1536 M
FULLTHRESHOLD : 85
ALLOWAUTOALT : YES
REBUILD PERCENT: N/A
DUPLEX : DISABLED
ALLOWREALLOCATE: YES
PREFERENCE LIST: CF01
ENFORCEORDER : NO
EXCLUSION LIST IS EMPTY

Additionally, note the values of the fields in the SPACE USAGE section:

- ENTRIES
- ELEMENTS
- EMCS
- LOCKS

An example of the values follows:

SPACE USAGE	IN-USE	TOTAL	%
ENTRIES:	344686	345242	99
ELEMENTS:	6548455	6548467	99
EMCS:	2	780318	0
LOCKS:	1024		

Results

You have created the basic configuration. You can now obtain an idea of the baseline performance of your configuration using whatever method you select.

What to do next

You should test the basic scenario. As an example, you can use the following three applications, starting the applications in the order shown and, running them concurrently.

1. Use a PCF application to request the current depth (**CURDEPTH**) value for SCEN2.Q every five seconds. The output can be used to plot the depth of the queue over time.
2. A single threaded getting application repeatedly gets messages from SCEN2.Q, using a get with an infinite wait. To simulate processing of the messages that were removed, the getting application pauses for four milliseconds for every ten messages that it removed.
3. A single threaded putting application puts a total of one million 4 KB non-persistent messages to SCEN2.Q. This application does not pause between putting each message so messages are put on SCEN2.Q faster than the getting application can get them.

As a result, when the putting application is running, the depth of SCEN2.Q increases.

When structure IBM1SCEN2 is filled, and the putting application receives a MQRC_STORAGE_MEDIUM_FULL reason code, the putting application sleeps for five seconds before attempting to put the next message to the queue.


You can plot the results of the CURDEPTH application over a period of time. You obtain some form of saw-tooth wave output as the putting application pauses to allow the queue to partially empty.

Go to “Adding SCM to the initial structure” on page 361.

Related concepts:

“Use of storage class memory with shared queues” on page 345

The use of storage class memory (SCM) can be advantageous when used with IBM MQ for z/OS shared queues.

Adding SCM to the initial structure: 

How you add SCM for improved performance on IBM MQ.

About this task

This part of the task uses the basic configuration described in “Improved performance - basic configuration” on page 358. The scenario describes the addition of SCM to the initial structure.

This final configuration is illustrated in Figure 64.

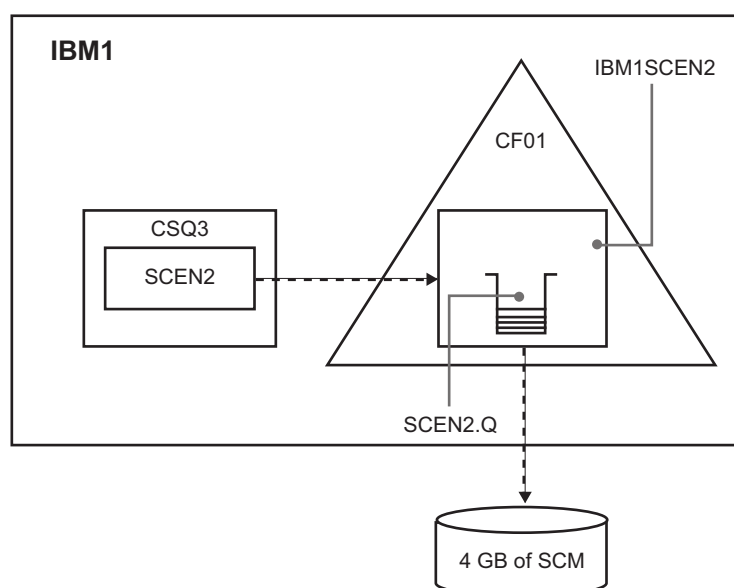


Figure 64. Configuration adding SCM for improved performance

Procedure

1. Add 4 GB of SCM to structure IBM1SCEN2 by carrying out the following procedure:
 - a. Check how much SCM is installed, and allocated to CF01, by issuing the following command:
 - b. Check the STORAGE-CLASS MEMORY figures in the STORAGE CONFIGURATION section of the displayed output to see the available storage.
 - c. Update the CFRM policy with the SCMMAXSIZE and SCMALGORITHM keywords as shown:

```
STRUCTURE
NAME(IBM1SCEN2)
SIZE(2048M)
INITSIZE(2048M)
ALLOWAUTOALT(YES)
FULLTHRESHOLD(85)
PREFLIST(CF01)
ALLOWREALLOCATE(YES)
```

```
DUPLEX(DISABLED)
ENFORCEORDER(NO)
SCMMAXSIZE(4G)
SCMALGORITHM(KEYPRIORITY1)
```

2. Activate the CFRM policy by issuing the following command:
SETXCF START,POLICY,TYPE=CFRM,POLNAME=IBM1SCEN2
3. Rebuild the IBM1SCEN2 structure. You must carry out this procedure because the structure was allocated when you made the previous changes.
Issue the following command to rebuild the structure:
SETXCF START,REBUILD,STRNM=IBM1SCEN2
4. Issue the following command to confirm the new configuration of the structure:
D XCF,STR,STRNAME=IBM1SCEN2

Review the output of the command, a portion of which follows:

SPACE USAGE	IN-USE	TOTAL	%
ENTRIES:	33	342684	0
ELEMENTS:	48	6503697	0
EMCS:	2	575600	0
LOCKS:		1024	

Results

Calculate the change in the use of real storage by the increase in control storage required to use SCM.

- Before SCM is added to the structure, the structure has these totals as shown in “Improved performance - basic configuration” on page 358:
 - 345,242 entries
 - 6,548,467 elements
 - 780,318 EMCS
- After SCM is added to the structure, the structure has these totals:
 - 342,684 entries
 - 6,503,697 elements
 - 575,600 EMCS

Using these figures, after the SCM was added, the structure is reduced in size by:

- 2558 entries
- 44,770 elements
- 204,718 EMCS

The amount of structure storage that is used to manage SCM, is as follows for a 2 GB structure with 4 GB of SCM allocated:

$$(2558 + 44,770 + 204,718) * 256 = 61.5 \text{ MB}$$

Note that adding more SCM is likely to achieve only a marginal reduction of the size of the structure, because the amount of control storage used to track SCM increases, both as the structure size, and the amount of allocated SCM increases.

What to do next

Repeat the tests described in the final section of “Improved performance - basic configuration” on page 358.

You can plot the results of the revised application over a period of time. Comparing the plot to the one obtained previously, you now obtain an output without a saw-tooth wave, as the putting application no longer has to wait for the queue to partially empty.

For more information, refer to MP16: WebSphere MQ for z/OS - Capacity planning & tuning.

Distributed queuing and queue-sharing groups

z/OS

Distributed queuing and queue-sharing groups are two techniques that you can use to increase the availability of your application systems. Use this topic to find further information about these techniques.

To complement the high availability of messages on shared queues, the distributed queuing component of IBM MQ has additional functions to provide the following:

- Higher availability to the network.
- Increased capacity for inbound network connections to the queue-sharing group.

Figure 65 illustrates distributed queuing and queue-sharing groups. It shows two queue managers within a sysplex, both of which belong to the same queue-sharing group. They can both access shared queue SQ1. Queue managers in the network (on AIX and Windows for example) can put messages onto this queue through the channel initiator of either queue manager. Cloned applications on both queue managers service the queue.

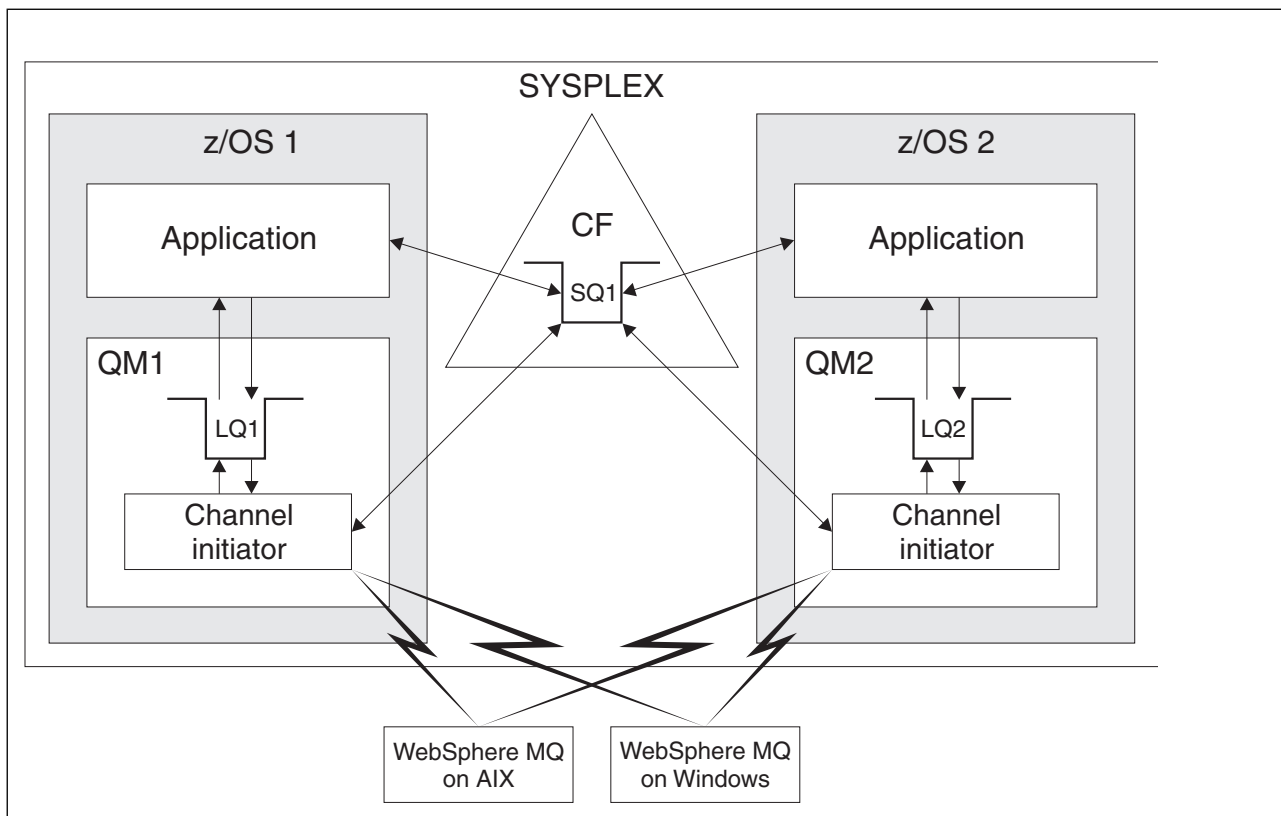


Figure 65. Distributed queuing and queue-sharing groups

Related concepts:

“Shared channels”

Use this topic to understand the concepts of shared channels and their use with IBM MQ for z/OS.

“Intra-group queuing” on page 366

Intra-group queuing allows transfer of messages between queue managers in a queue-sharing group.

“Clusters and queue-sharing groups” on page 367

Use this topic to understand how you can use queue-sharing groups with clusters.

Shared channels:

Use this topic to understand the concepts of shared channels and their use with IBM MQ for z/OS.

A number of networking products provide a mechanism to hide server failures from the network, or to balance inbound network requests across a set of eligible servers. The network products make a *generic port* available for inbound network connection requests, and the inbound request can be satisfied by connecting to one of the eligible servers.

These networking products include:

- VTAM generic resources
- SYSPLEX Distributor

The channel initiator takes advantage of these products to use the capabilities of shared queues

There are two types of shared channels, *shared inbound channel*, and the *shared outbound channel*.

- Shared inbound channels
- Shared outbound channels

For further information about channels see

- Shared channel summary
- Shared channel status

Shared inbound channels

Each channel initiator in the queue-sharing group starts an additional listener task to listen on a *generic port*. This generic port is made available to the network by one of the supporting technologies (VTAM, TCP/IP). Inbound network attach requests to the generic port are dispatched by the network technology, to any one of the listeners in the queue-sharing group (QSG) that are listening on the generic port.

You can start a channel on the channel initiator to which the inbound attach is directed if the channel initiator has access to a channel definition for a channel with that name. You can define a channel definition to be private to a queue manager or stored on the shared repository and so available anywhere (a global definition). This means that you can make a channel definition available on any channel initiator in the queue-sharing group by defining it as a global definition.

There is an additional difference when starting a channel through the generic port; channel synchronization is with the queue-sharing group and not with an individual queue manager. For example, consider a remote queue manager starting a channel through the generic port. When the channel first starts, it might start on queue manager QM1 and messages flow. If the channel stops and is restarted on queue manager QM2, information about the number of messages that have flowed is still correct because the synchronization is with the queue-sharing group.

You can use an inbound channel started through the generic port to put messages to any queue. The remote queue manager does not know whether the target queue is shared or not. If the target queue is a

shared queue, the remote queue manager connects through any available channel initiator in a load-balanced fashion and the messages are put to the shared queue.

If the target queue is a private queue, the messages are put to the private queue owned by which ever queue manager the current instance of the channel is connected to. In this environment, known as *replicated local queues*, each queue manager must have the same set of private queues defined.

Configuring SVRCONN channels for a queue-sharing group

The optimal configuration for SVRCONN channels in a queue-sharing group is to set up private listeners in each CHINIT which use a different port number from the point to point channels. These listener ports are then used as the 'back-end' resources for a new workload distribution mechanism such as Sysplex Distributor using Virtual IP addresses (VIPA). The external VIPA address is then used as the target address for the CLNTCONN definitions in the network. The SVRCONN channel can be defined with QSGDISP(GROUP) so the same definition is available to all queue managers in the QSG. This configuration avoids using a shared listener, and therefore the reduces performance effect of the queue-sharing group maintaining shared channel state which is not needed for client/server channels.

Shared outbound channels

An outbound channel is considered to be a shared channel if it is taking messages from a shared transmission queue. If it is shared, it holds synchronization information at queue-sharing group level. This means that the channel can be restarted on a different queue manager and channel initiator instance within the queue-sharing group if the communications subsystem, channel initiator, or queue manager fails. Restarting failed channels in this way is a feature of shared channels called *peer channel recovery*.

Workload balancing for shared outbound channels

An outbound shared channel is eligible for starting on any channel initiator within the queue-sharing group, if you have not specified that you want it to be started on a particular channel initiator. The channel initiator selected by IBM MQ is determined using the following criteria:

- Is the communications subsystem required currently available to the channel initiator?
- Is a Db2 connection available to the channel initiator?
- Which channel initiator has the lowest current workload? The workload includes channels that are active and retrying.

Shared channel summary

Shared channels differ from private channels in the following ways:

Private channel

Tied to a single channel initiator.

- Outbound channel uses a local transmission queue.
- Inbound channel started through a local port.
- Synchronization information held in SYSTEM.CHANNEL.SYNCQ queue.

Shared Channel

Workload balanced with high availability.

- Outbound channel uses a shared transmission queue.
- Inbound channel started through a generic port.
- Synchronization information held in SYSTEM.QSG.CHANNEL.SYNCQ queue.

You specify whether a channel is private or shared when you start the channel by using CHLDISP options with the START CHANNEL command. A shared channel can be started by triggering in the same way as a private channel. However, when a shared channel is started, IBM MQ performs workload balancing

and starts the channel on the most appropriate channel initiator within the queue-sharing group. (If required, you can specify that a shared channel is to be started on a particular channel initiator.)

Shared channel status

The channel initiators in a queue-sharing group maintain a shared channel-status table in Db2. This records which channels are active on which channel initiators. The shared channel-status table is used if there is a channel initiator or communications system failure. It indicates which channels need to be restarted on a different channel initiator in the queue-sharing group.

Intra-group queuing:

Intra-group queuing allows transfer of messages between queue managers in a queue-sharing group.

You can perform fast message transfer between queue managers in a queue-sharing group without defining channels. This uses a system queue called the `SYSTEM.QSG.TRANSMIT.QUEUE`, which is a shared transmission queue. Each queue manager in the queue-sharing group starts a task called the intra-group queuing agent, which waits for messages to arrive on this queue that are destined for their queue manager. When such a message is detected, it is removed from the queue and placed on the correct destination queue.

Standard name resolution rules are used but, if intra-group queuing (IGQ) is enabled and the target queue manager is within the queue-sharing group, the `SYSTEM.QSG.TRANSMIT.QUEUE` is used to transfer the message to the correct destination queue manager instead of using a transmission queue and channel.

You enable intra-group queuing through a queue manager attribute. Intra-group queuing moves nonpersistent messages outside sync point, and persistent messages within sync point. If it finds a problem delivering messages to the target queue, intra-group queuing tries to put them to the dead-letter queue. If the dead-letter queue is full or undefined, nonpersistent messages are discarded, but persistent messages are backed out and returned to the `SYSTEM.QSG.TRANSMIT.QUEUE`, and the IGQ agent tries to deliver the messages until it is successful.

An inbound shared channel that receives a message destined for a queue on a different queue manager in the queue-sharing group can use intra-group queuing to *hop* the message to the correct destination.

There might be times when you want the local queue manager to put a message directly to the target queue if the target queue is a shared queue, rather than the message first being transferred to the target queue manager. You can use the queue manager attribute `SQQMNAME` to control this. If you set the value of `SQQMNAME` to `USE`, the `MQOPEN` command is performed on the queue manager specified by the `ObjectQMgrName`. However, if the target queue is a shared queue and you set the value of `SQQMNAME` to `IGNORE`, and the `ObjectQMgrName` is that of another queue manager in the queue-sharing group, the shared queue is opened on the local queue manager. If the local queue manager cannot open the target queue, or put a message to the queue, the message is transferred to the specified `ObjectQMgrName` through either IGQ or an MQ channel.

Intra-group queuing (IGQ) supports large messages, the largest being 100 MB *minus* the length of the transmission queue header.

If you use this feature, users must have the same access to the queues on each queue manager in the queue-sharing group.

Clusters and queue-sharing groups: z/OS

Use this topic to understand how you can use queue-sharing groups with clusters.

You can make your shared queues available to a cluster in a single definition. To do this you specify the name of the cluster when you define the shared queue.

Users in the network see the shared queue as being hosted by each queue manager within the queue-sharing group (the shared queue is not advertised as being hosted by the queue-sharing group). Clients can start sessions with any members of the queue-sharing group to put messages to the same shared queue.

Figure 66 shows how members of a cluster can access a shared queue through any member of the queue-sharing group.

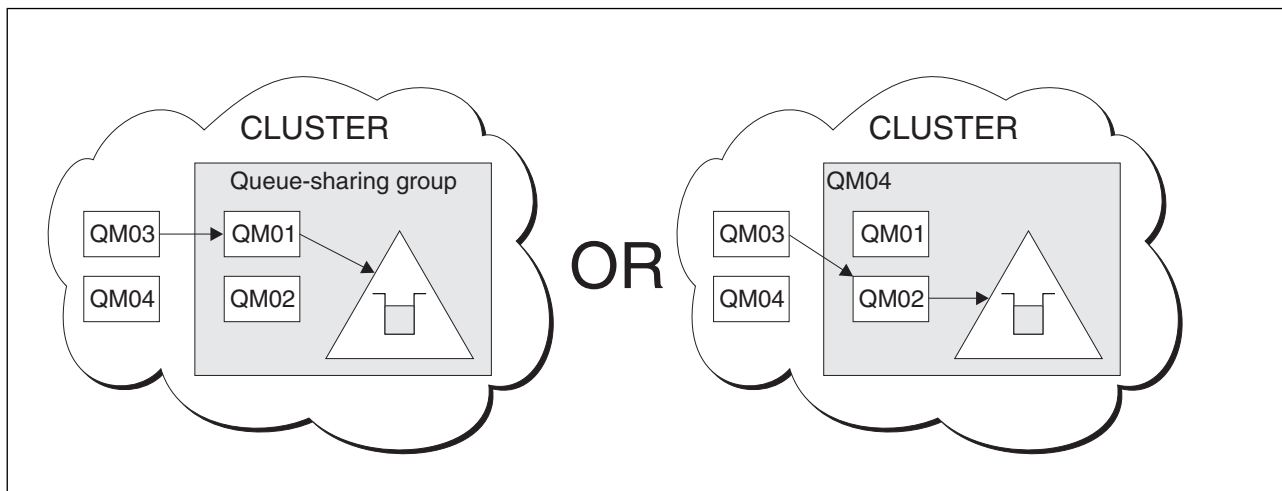


Figure 66. A queue-sharing group as part of a cluster

Influencing workload distribution with shared queues

z/OS

Use this topic to understand the factors that affect workload distribution with shared queues in a queue-sharing group.

IBM MQ does not provide workload balancing for shared queues. However, workload distribution in a queue-sharing group (QSG) can be influenced in a *pull based fashion*. The choice of which queue manager services a queue (receives a message written to a shared queue) is affected by the available processing capacity of each queue manager in the queue-sharing group and the workload management goals defined across the sysplex.

However, it is important to appreciate, the queue manager that performs the MQPUT of a message can also have a large influence in deciding which queue manager gets the message.

A local queue manager is more likely to perform the MQGET

For an application performing an MQPUT, the local queue manager is said to be the queue manager to which the application is connected.

Exactly which queue manager services an MQPUT of a message by performing an MQGET on behalf of a getting application is influenced by the following considerations.

When a message is put to an empty shared queue, the local queue manager is typically posted before any of the other queue manager in the queue-sharing group is notified. If the local queue manager is in a position to process the message, it receives a list transition notification from the coupling facility (CF) before any other queue manager in the QSG. (A list transition notification is a notification that the shared queue has changed state from empty to non-empty.)

The possible scenarios, in this case, are as follows:

1. MQPUT of nonpersistent message out of sync point and *fast put to waiting getter*.

If there is an application with an *MQGET with wait* on the local queue manager for the queue, then the MQPUT of the message is passed directly to the getting application's buffer and not written to the queue. This is true for shared and non-shared queues. This feature is often called *fast put to a waiting getter* mechanism. In the case of shared queues, no other queue manager in the QSG is notified as there is no transition from empty to non-empty of the queue. This means, for example, that provided this queue manager can service all the puts from this application and assuming that no other applications are putting messages to the queue, then no other queue manager in the queue-sharing group assists in draining this queue. If however there is no *MQGET with wait* on the local queue manager, and a message is put to the shared queue then the CF will notify other queue managers in the queue-sharing group according to its rules for notifications of list transitions.

2. MQPUT of a persistent or in-syncpoint message.

In this case, if there is an application with an *MQGET with wait* on the local queue manager, then the message is put to the shared queue and the CF notifies other queue managers in the queue-sharing group according to its rules for notifications of list transitions. However, the local queue manager does not wait for a transition notification from the CF but honors any local *MQGET with wait* first and usually performs the get of this message on behalf of the application before any other queue manager in the queue-sharing group can respond to a CF notification. This is dependent on how busy the local queue manager is. Otherwise, any queue manager notified by the CF due to the arrival of the message on the empty queue will try to service the get first. The first queue manager to respond processes the new message.

3. Finally, if the queue is not drained of messages, where the CF has sent a notification of a state change from empty to non-empty for the queue, all connected queue managers will have an opportunity to assist in the processing of the queue. In this event, the workload is said to be *pull based*.

This design allows for the improved performance over a purely pull based workload distribution. The aim is to take advantage of the high availability offered by queues held in the CF while allowing the queue manager, where possible, to perform the *MQGET* without needing to reference the CF and so to process the message workload as efficiently as possible.

Alternative approaches can be adopted where emphasis on balance of the workload is more important than the previously described performance enhancements. For example, ensuring that none of the getting applications are connected to the same queue manager that the putting application is connected to. Using this design all messages are put to the queue and all queue managers in the QSG are notified when the queue moves from empty to non-empty, in accordance with the CF algorithm for handling such transitions. In addition, the *fast put to waiting getter* mechanism is not applicable.

Where to find more information about shared queues and queue-sharing groups

z/OS

Use the table in this topic to find more information about how IBM MQ for z/OS uses shared queues and queue-sharing groups.

Table 23. Where to find more information about shared queues and queue-sharing groups

Topic	Where to look
Queue-sharing group recovery	"Recovery and restart on z/OS" on page 412
Queue-sharing group security	"Security concepts in IBM MQ for z/OS" on page 429
Private and global object definitions Directing commands to different queue managers	Issuing commands
Planning your coupling facility environment	Defining coupling facility resources
Planning your SMDS environment	Planning your shared message data set (SMDS) environment
Planning your Db2 environment	Planning your Db2 environment
Setting up your shared queues System parameters	"Shared queues and queue-sharing groups" on page 322
Utility programs Migrating queues	Using the IBM MQ Utilities
Console messages	Messages for IBM MQ for z/OS
MQSC commands	MQSC commands
IBM MQ clusters	Configuring a queue manager cluster
IBM MQ distributed queuing Channel names	Introduction to distributed queue management
Writing applications	Overview of application design
MQCONN call	MQCONN

Intra-group queuing

z/OS

This section describes intra-group queuing, an IBM MQ for z/OS function unique to the z/OS platform. This function is only available to queue managers defined to a queue-sharing group.

For information about queue-sharing groups, see "Shared queues and queue-sharing groups" on page 322.

Related concepts:

[“Intra-group queuing concepts”](#)

Intra-group queuing can be used to deliver, more efficiently, small messages to queues residing on remote queue managers within a queue-sharing group.

[“Benefits of intra-group queuing”](#) on page 372

The benefits of intra-group queuing are: reduced system definitions, reduced system administration, improved performance, supports migration, and delivery of messages when multi-hopping between queue managers in a queue-sharing group.

[“Limitations of intra-group queuing”](#) on page 374

The limitations of intra-group queuing are: messages eligible for transfer using intra-group queuing, number of intra-group queuing agents per queue manager, and starting and stopping the intra-group queuing agent.

[“Getting started with intra-group queuing”](#) on page 374

You can enable, disable, and use intra-group queuing as described in this topic.

[“Intra-group queuing configurations”](#) on page 375

In addition to the typical intra-group queuing configuration, other configurations are possible.

[“Intra-group queuing messages”](#) on page 380

This section describes the messages put to the `SYSTEM.QSG.TRANSMIT.QUEUE`.

[“Security for intra-group queuing”](#) on page 382

This topic describes the security arrangements for intra-group queuing.

[“Specific properties of intra-group queuing”](#) on page 383

This section describes the specific properties of intra-group queuing including Invalidation of object handles, self recovery and retry capability of the intra-group queuing agent, and the intra-group queuing agent and serialization.

Related information:

[“Intra-group queuing terminology”](#) on page 372

Explanations of the terminology: intra-group queuing, shared transmission queue for use by intra-group queuing, and intra-group queuing agent.

Intra-group queuing concepts

z/OS

Intra-group queuing can be used to deliver, more efficiently, small messages to queues residing on remote queue managers within a queue-sharing group.

Intra-group queuing (IGQ) can effect potentially fast and less-expensive small message transfer between queue managers within a queue-sharing group (QSG), without the need to define channels between the queue managers.

The following diagram shows a typical example of intra-group queuing.

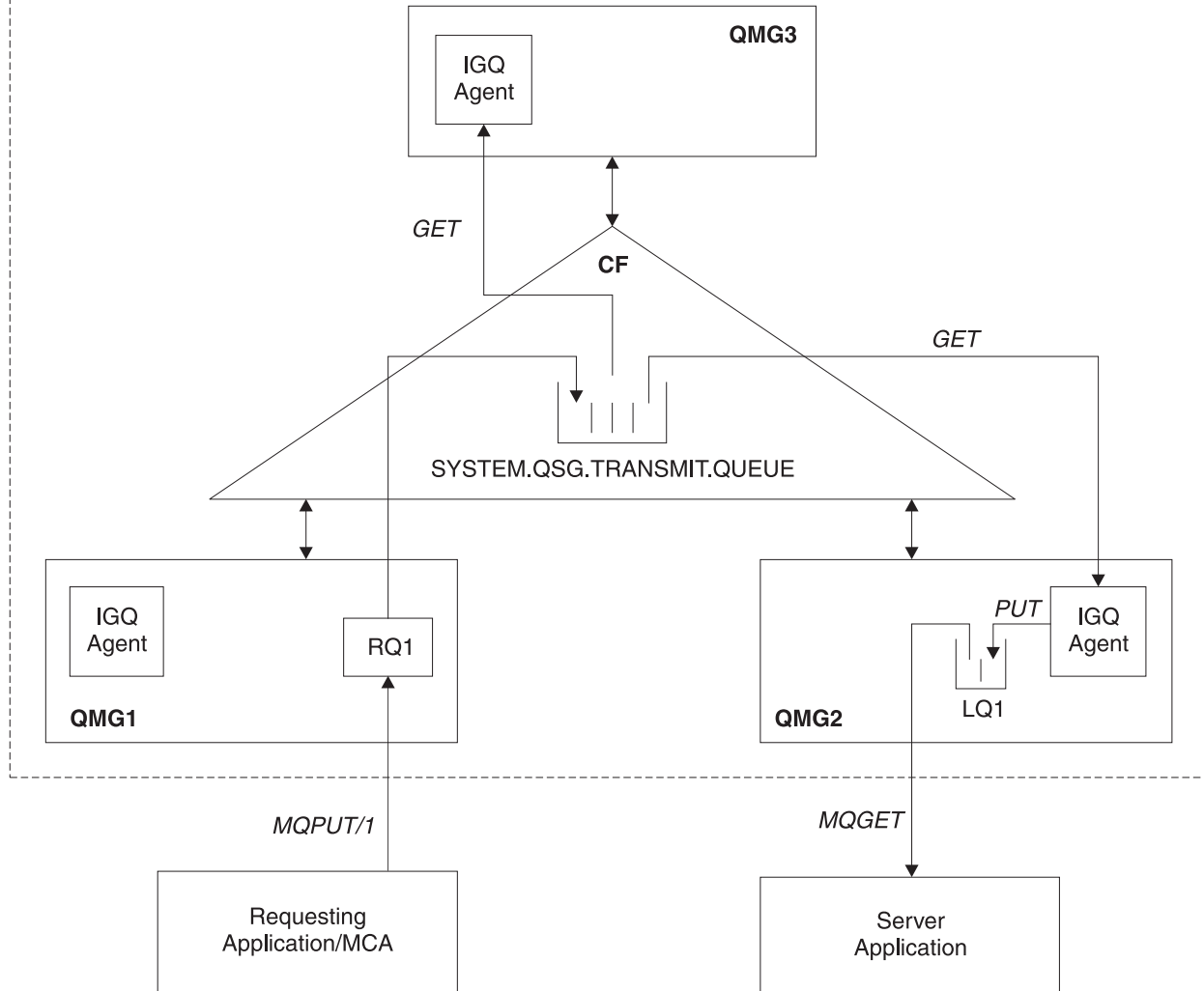


Figure 67. An example of intra-group queuing

The diagram shows:

- IGQ agents running on three queue managers (QMG1, QMG2, and QMG3) that are defined to a queue-sharing group called SQ26.
- Shared transmission queue SYSTEM.QSG.TRANSMIT.QUEUE that is defined in the coupling facility (CF).
- A remote queue definition that is defined in queue manager QMG1.
- A local queue that is defined in queue manager QMG2.
- A requesting application (this application could be a Message Channel Agent (MCA)) that is connected to queue manager QMG1.
- A server application that is connected to queue manager QMG2.
- A request message being placed on to the SYSTEM.QSG.TRANSMIT.QUEUE.

Intra-group queuing and the intra-group queuing agent

An IGQ agent is started during queue manager initialization. When applications open and put messages to remote queues, the local queue manager determines whether intra-group queuing is used for message transfer. If intra-group queuing is to be used, the local queue manager places the message on to the

SYSTEM.QSG.TRANSMIT.QUEUE. The IGQ agent on the target remote queue manager retrieves the message and places it on to the destination queue.

Intra-group queuing terminology

z/OS

Explanations of the terminology: intra-group queuing, shared transmission queue for use by intra-group queuing, and intra-group queuing agent.

Intra-group queuing

Intra-group queuing can effect potentially fast and less expensive message transfer between queue managers in a queue-sharing group, without the need to define channels.

Shared transmission queue for use by intra-group queuing

Each queue-sharing group has a shared transmission queue called SYSTEM.QSG.TRANSMIT.QUEUE for use by intra-group queuing. If intra-group queuing is enabled, SYSTEM.QSG.TRANSMIT.QUEUE appears in the name resolution path when opening remote queues. When applications (including Message Channel Agents (MCAs)) put messages to a remote queue, the local queue manager determines the eligibility of messages for fast transfer and places them on SYSTEM.QSG.TRANSMIT.QUEUE.

Intra-group queuing agent

The IGQ agent is the task, started at queue manager initialization, that waits for suitable messages to arrive on the SYSTEM.QSG.TRANSMIT.QUEUE. The IGQ agent retrieves suitable messages from this queue and delivers them to the destination queues.

The IGQ agent for each queue manager is always started because intra-group queuing is used by the queue manager itself for its own internal processing.

Benefits of intra-group queuing

z/OS

The benefits of intra-group queuing are: reduced system definitions, reduced system administration, improved performance, supports migration, and delivery of messages when multi-hopping between queue managers in a queue-sharing group.

The benefits of intra-group queuing are:

Reduced system definitions

Intra-group queuing removes the need to define channels between queue managers in a queue-sharing group.

Reduced system administration

Because there are no channels defined between queue managers in a queue-sharing group, there is no requirement for channel administration.

Improved performance

Because there is only one IGQ agent needed for the delivery of a message to a target queue (instead of two intermediate sender and receiver agents), the delivery of messages using intra-group queuing can be less expensive than the delivery of messages using channels. In intra-group queuing there is only a receiving component, because the need for the sending component has been removed. This saving is because the message is available to the IGQ agent at the destination queue manager for delivery to the destination queue once the put operation at the local queue manager has completed and, in the case of messages put in sync point scope, committed.

Supports migration

Applications external to a queue-sharing group can deliver messages to a queue residing on any queue manager in the queue-sharing group, while being connected only to a particular queue manager in the queue-sharing group. This is because messages arriving on a receiver channel, destined for a queue on a remote queue manager, can be transparently sent to the destination queue using intra-group queuing. This facility allows applications to be deployed among the queue-sharing group without the need to change any systems that are external to the queue-sharing group.

A typical configuration is illustrated by the following diagram, in which:

- A requesting application connected to queue manager QMG1 needs to send a message to a local queue on queue manager QMG3.
- Queue manager QMG1 is connected only to queue manager QMG2.
- Queue managers QMG2 and QMG3, which were previously connected using channels, are now members of queue-sharing group SQ26.

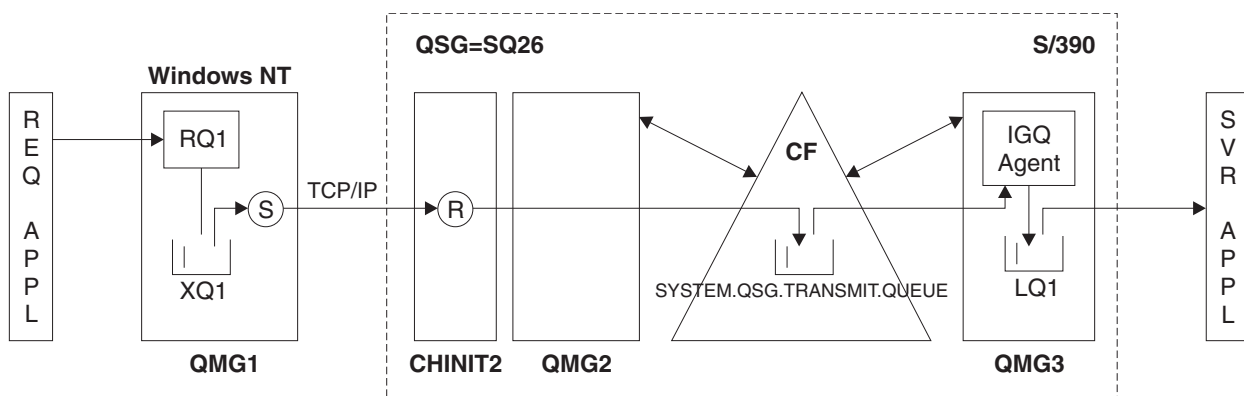


Figure 68. An example of migration support

The flow of operations is as follows:

1. The requesting application puts a message, destined for local queue LQ1 at remote queue manager QMG3, on to remote queue definition RQ1.
2. Queue manager QMG1, running on a Windows NT workstation, places the message on to the transmission queue XQ1.
3. Sender MCA (S) on QM1 transmits the message, using TCP/IP, to the receiver MCA (R) on channel initiator CHINIT2.
4. Receiver MCA (R) on channel initiator CHINIT2 places the message on to the shared transmission queue SYSTEM.QSG.TRANSMIT.QUEUE.
5. IGQ agent on queue manager QMG3 retrieves the message from the SYSTEM.QSG.TRANSMIT.QUEUE and places it on to the target local queue LQ1.
6. The server application retrieves the message from the target local queue and processes it.

Delivery of messages when multi-hopping between queue managers in a queue-sharing group

The previous diagram in Supports migration also illustrates the delivery of messages when multi-hopping between queue managers in a queue-sharing group. Messages arriving on a queue manager within the queue-sharing group, but destined for a queue on another queue manager in the queue-sharing group, can be easily transmitted to the destination queue on the destination queue manager, using intra-group queuing.

Limitations of intra-group queuing

z/OS

The limitations of intra-group queuing are: messages eligible for transfer using intra-group queuing, number of intra-group queuing agents per queue manager, and starting and stopping the intra-group queuing agent.

This topic describes the limitations of intra-group queuing.

Messages eligible for transfer using intra-group queuing

Because intra-group queuing uses a shared transmission queue that is defined in the coupling facility (CF), intra-group queuing is limited to delivering messages of the maximum supported message length for shared queues minus the length of a transmission queue header (MQXQH).

Number of intra-group queuing agents per queue manager

Only one IGQ agent is started per queue manager in a queue-sharing group.

Starting and stopping the intra-group queuing agent

The IGQ agent is started during queue manager initialization and terminated during queue manager shut-down. It is designed to be a long running, self recovering (in the event of abnormal termination), task. If there is an error with the definition of the `SYSTEM.QSG.TRANSMIT.QUEUE` (for example, if this queue is Get inhibited) the IGQ agent keeps retrying. If the IGQ agent encounters an error that results in normal termination of the agent while the queue manager is still active, it can be restarted by issuing an `ALTER QMGR IGQ(ENABLED)` command. This command avoids the need to recycle the queue manager.

Setting the queue manager attribute IGQ to ENABLED or DISABLED

If the queue manager attribute IGQ is set to `ENABLED` or `DISABLED`, existing object handles may be invalidated with reason code `MQRC_OBJECT_CHANGED`. See *Getting started with intra-group queuing* for more information.

Getting started with intra-group queuing

z/OS

You can enable, disable, and use intra-group queuing as described in this topic.

Enabling intra-group queuing

To enable intra-group queuing on your queue managers, you need to do the following:

- Define a shared transmission queue called `SYSTEM.QSG.TRANSMIT.QUEUE`. The definition of this queue can be found in `thlqual.SCSQPROC(CSQ4INSS)`, the `CSQINP2` sample for `SYSTEM` objects for queue-sharing groups. This queue must be defined with the correct attributes, as stated in `thlqual.SCSQPROC(CSQ4INSS)`, for intra-group queuing to work properly.
- Because the IGQ agent is always started at queue manager initialization, intra-group queuing is always available for inbound message processing. The IGQ agent processes any messages that are placed on the `SYSTEM.QSG.TRANSMIT.QUEUE`. However, to enable intra-group queuing for outbound processing, the queue manager attribute IGQ must be set to `ENABLED`.

Important: If the queue manager attribute IGQ is set to `ENABLED`, existing object handles may be invalidated with reason code `MQRC_OBJECT_CHANGED`. See “Invalidation of object handles (`MQRC_OBJECT_CHANGED`)” on page 384 for more information. As described in the 'Programmer response' section for this reason code, applications need to be coded to handle this situation (see 2041 (07F9) (RC2041): `MQRC_OBJECT_CHANGED` for more details).

Additionally, as IGQ is designed as a long running and self-recovering task, which starts during initialization and terminates with shutdown, see “Limitations of intra-group queuing” for further information.

Disabling intra-group queuing

To disable intra-group queuing for outbound message transfer, set the queue manager attribute IGQ to DISABLED. If intra-group queuing is disabled for a particular queue manager, the IGQ agent on that queue manager can still process inbound messages that have been placed on the SYSTEM.QSG.TRANSMIT.QUEUE by a queue manager that does have intra-group queuing enabled for outbound transfer.

Important: If the queue manager attribute IGQ is set to ENABLED, existing object handles may be invalidated with reason code MQRC_OBJECT_CHANGED. See “Invalidation of object handles (MQRC_OBJECT_CHANGED)” on page 384 for more information. As described in the ‘Programmer response’ section for this reason code, applications need to be coded to handle this situation (see 2041 (07F9) (RC2041): MQRC_OBJECT_CHANGED for more details).

Additionally, as IGQ is designed as a long running and self-recovering task, which starts during initialization and terminates with shutdown, see “Limitations of intra-group queuing” on page 374 for further information.

Using intra-group queuing

Once intra-group queuing is enabled, it is available for use and a queue manager uses it whenever possible. That is, when an application puts a message to a remote queue definition, to a fully qualified remote queue, or to a cluster queue, the queue manager determines if the message is eligible to be delivered using intra-group queuing and if it is, places the message on to SYSTEM.QSG.TRANSMIT.QUEUE. There is no need to change user applications, or to application queues, because for eligible messages the queue manager uses the SYSTEM.QSG.TRANSMIT.QUEUE, in preference to any other transmission queue.

Intra-group queuing configurations

z/OS

In addition to the typical intra-group queuing configuration, other configurations are possible.

Figure 67 on page 371 describes the typical configuration.

Related concepts:

“Distributed queuing with intra-group queuing (multiple delivery paths)”

For applications that process short messages it might be feasible to configure intra-group queuing only for delivering messages between queue managers in a queue-sharing group.

“Clustering with intra-group queuing (multiple delivery paths)” on page 377

It is possible to configure queue managers so that they are in a cluster as well as in a queue-sharing group.

“Clustering, intra-group queuing and distributed queuing” on page 380

It is possible to configure a queue manager that is a member of a cluster as well as a queue-sharing group and is connected to a distributed queue manager using a sender/receiver channel pair.

Distributed queuing with intra-group queuing (multiple delivery paths): z/OS

For applications that process short messages it might be feasible to configure intra-group queuing only for delivering messages between queue managers in a queue-sharing group.

The choice of intra-group queuing over channel communications can be controlled by the CFSTRUCT type level. (3 instead of 4 or 5). The maximum message length as set on the SYSTEM.QSQ.TRANSMIT.QUEUE.

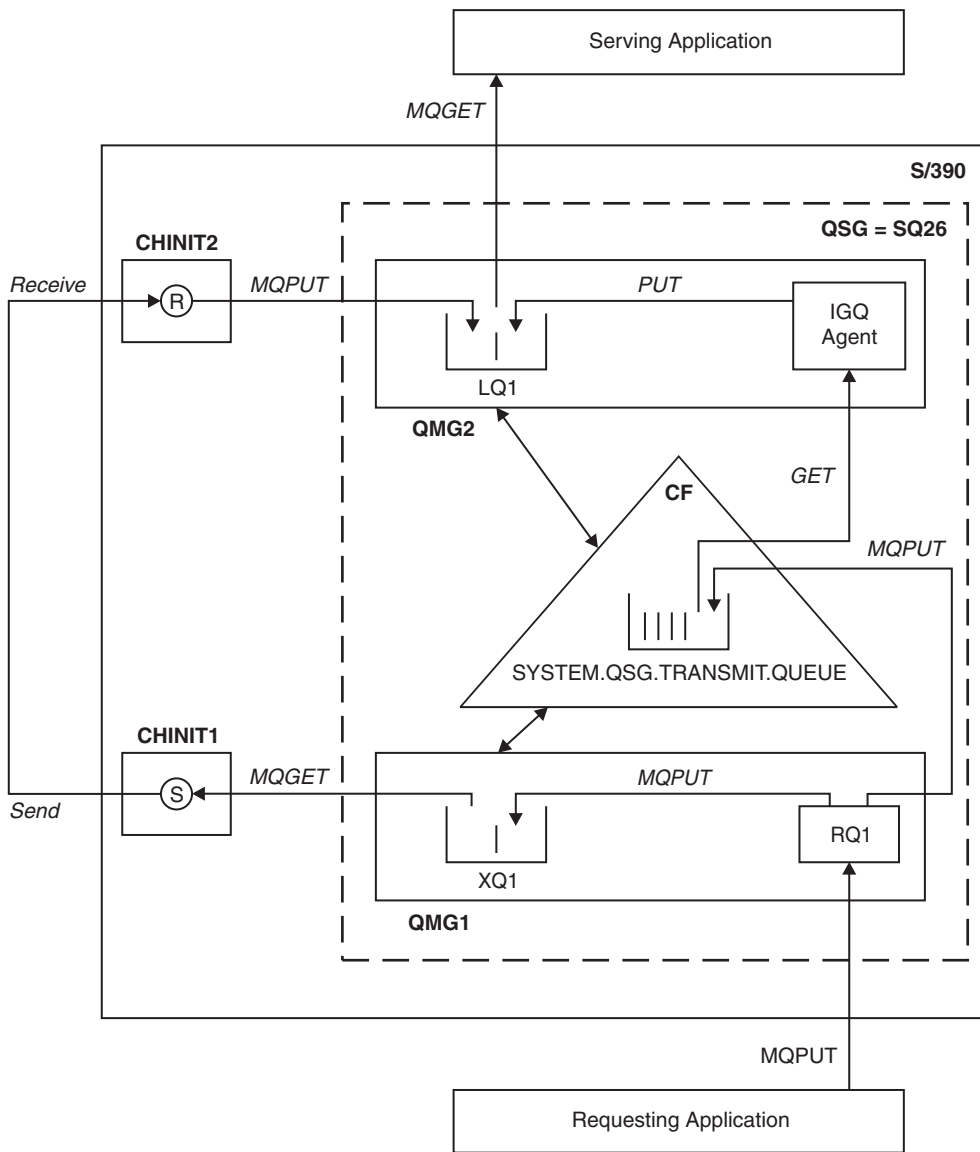


Figure 69. An example configuration

Open/Put processing

1. It is important to note that when the requesting application opens remote queue RQ1, name resolution occurs for both the non-shared transmission queue XQ1 and the shared transmission queue SYSTEM.QSG.TRANSMIT.QUEUE.
2. When the requesting application puts a message on to the remote queue, based on whether intra-group queuing is enabled for outbound transfer on the queue manager and on the message characteristics, the message is put to transmission queue XQ1, or to transmission queue SYSTEM.QSG.TRANSMIT.QUEUE. The queue manager places all large messages on to transmission queue XQ1, and all small messages on to transmission queue SYSTEM.QSG.TRANSMIT.QUEUE.
3. If transmission queue XQ1 is full, or is not available, put requests for large messages fail synchronously with a suitable return and reason code. However, put requests for small messages continue to succeed and are placed on transmission queue SYSTEM.QSG.TRANSMIT.QUEUE.
4. If transmission queue SYSTEM.QSG.TRANSMIT.QUEUE is full, or cannot be put to, put requests for small messages fail synchronously with a suitable return and reason code. However, put requests for

large messages continue to succeed and are placed on transmission queue XQ1. In this case, no attempt is made to put the small messages on to a transmission queue.

Flow for large messages

1. The requesting application puts large messages to remote queue RQ1.
2. Queue manager QMG1 puts the messages on to transmission queue XQ1.
3. Sender MCA (S) on queue manager QMG1 retrieves the messages from transmission queue XQ1 and sends them to queue manager QMG2.
4. Receiver MCA (R) on queue manager QMG2 receives the messages and places them on to destination queue LQ1.
5. The serving application retrieves and then processes the messages from queue LQ1.

Flow for small messages

1. The requesting application puts small messages on to remote queue RQ1.
2. Queue manager QMG1 puts the messages on to transmission queue SYSTEM.QSG.TRANSMIT.QUEUE.
3. IGQ on queue manager QMG2 retrieves the messages and places them on to the destination queue LQ1.
4. The serving application retrieves the messages from queue LQ1.

Points to note

1. The requesting application does not need to be aware of the underlying mechanism used for the delivery of messages.
2. A potentially faster message delivery mechanism can be achieved for small messages.
3. Multiple paths are available for message delivery (that is, the normal channel route and the intra-group queuing route).
4. The intra-group queuing route, being potentially faster, is selected in preference to the normal channel route. Depending on the message characteristics, message delivery might be divided across the two paths. Hence, messages might be delivered out of sequence (though this delivery is also possible if messages are delivered using only the normal channel route).
5. When a route has been selected, and messages have been placed on to the transmission queues, only the selected route is used for message delivery. Any unprocessed messages on the SYSTEM.QSG.TRANSMIT.QUEUE are not diverted to transmission queue XQ1.

Clustering with intra-group queuing (multiple delivery paths):

It is possible to configure queue managers so that they are in a cluster as well as in a queue-sharing group.

When messages are sent to a cluster queue and the local and remote destination queue managers are in the same queue-sharing group, intra-group queuing is used for the delivery of small messages (using the SYSTEM.QSG.TRANSMIT.QUEUE), and the delivery of large messages if intra-group queuing supports the size of the message. Also, the SYSTEM.CLUSTER.TRANSMIT.QUEUE is used for the delivery of messages to any queue manager that is in the cluster, but outside the queue-sharing group. The following diagram illustrates this configuration (the channel initiators are not shown).

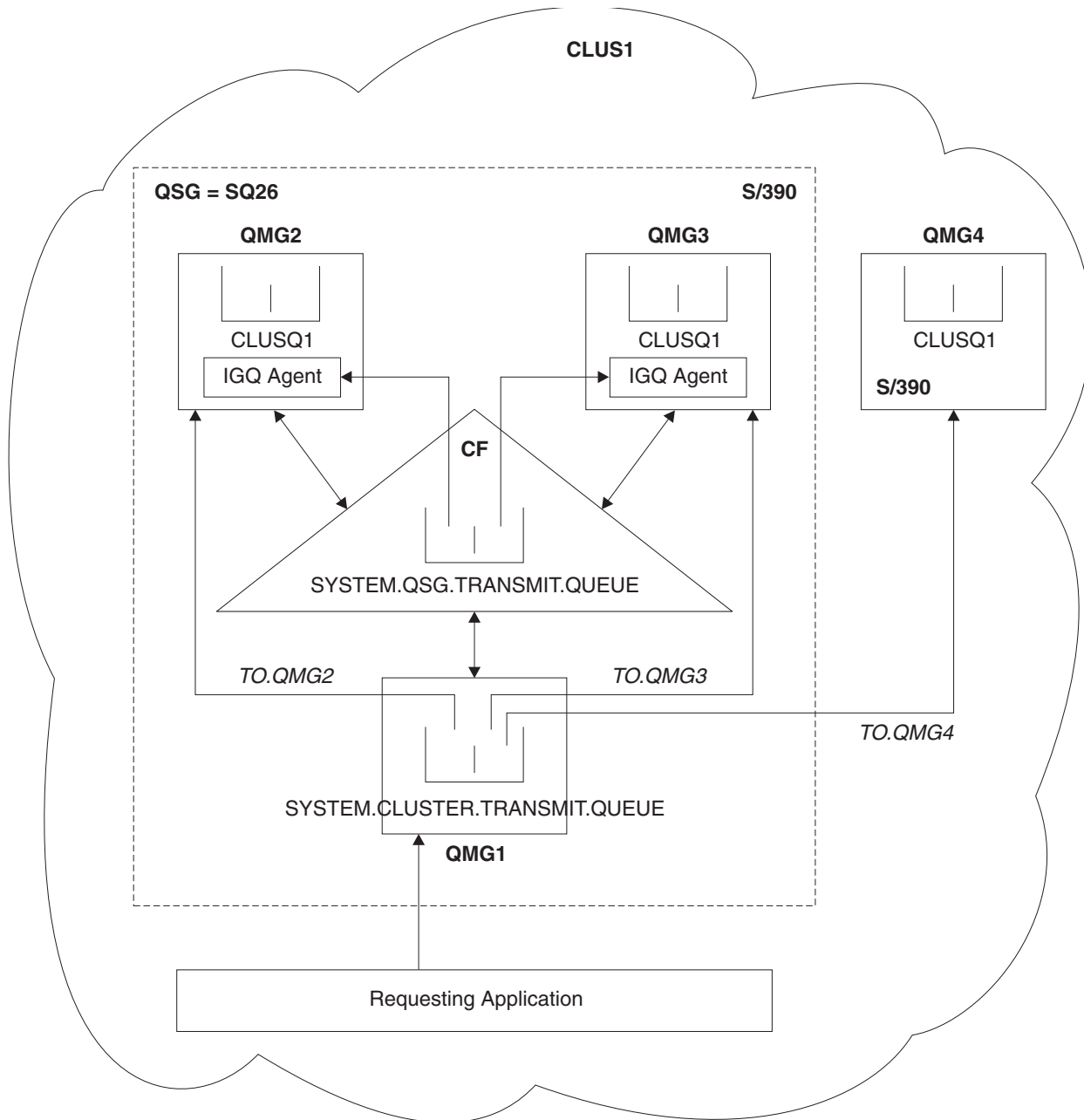


Figure 70. An example of clustering with intra-group queuing

The diagram shows:

- Four z/OS queue managers QMG1, QMG2, QMG3, and QMG4 configured in a cluster CLUS1.
- Queue managers QMG1, QMG2, and QMG3 configured in a queue-sharing group SQ26.
- IGQ agents running on queue managers QMG2 and QMG3.
- The local SYSTEM.CLUSTER.TRANSMIT.QUEUE defined in QMG1.

Note: For clarity, the SYSTEM.CLUSTER.TRANSMIT.QUEUE on the other queue managers not shown.

- The shared SYSTEM.QSG.TRANSMIT.QUEUE defined in the CF, which is in an IBM MQ structure configured with the CFLEVEL(3) RECOVER(YES) attribute.

- Cluster channels TO.QMG2 (connecting QMG1 to QMG2), TO.QMG3 (connecting QMG1 to QMG3), and TO.QMG4 (connecting QMG1 to QMG4).
- Cluster queue CLUSQ1 being hosted on queue managers QMG2, QMG3, and QMG4.

Assume that the requesting application opens the cluster queue with the MQOO_BIND_NOT_FIXED option, so that the target queue manager for the cluster queue is selected at put time.

If the selected target queue manager is QMG2:

- All large messages put by the requesting application are:
 - Put to the SYSTEM.CLUSTER.TRANSMIT.QUEUE on QMG1, because SYSTEM.QSG.TRANSMIT.QUEUE is in a CFLEVEL(3) structure; therefore supports messages only up to 63 KB in size.
 - Transferred to cluster queue CLUSQ1 on QMG2 using cluster channel TO.QMG2
- All small messages put by the requesting application are
 - Put to the shared transmission queue SYSTEM.QSG.TRANSMIT.QUEUE. This queue is in a structure configured with the RECOVER(YES) attribute, so is used for both persistent, and non-persistent, small messages.
 - Retrieved by the IGQ agent on QMG2
 - Put to the cluster queue CLUSQ1 on QMG2

If the selected target queue manager is QMG4:

- Because QMG4 is not a member of queue-sharing group SQ26, all messages put by the requesting application are
 - Put to the SYSTEM.CLUSTER.TRANSMIT.QUEUE on QMG1
 - Transferred to cluster queue CLUSQ1 on QMG4 using cluster channel TO.QMG4

Points to note

- The requesting application does not need to be aware of the underlying mechanism used for the delivery of messages.
- A potentially faster delivery mechanism is achieved for the transfer of small non-persistent messages between queue managers in a queue-sharing group (even if the same queue managers are in a cluster).
- Multiple paths are available for message delivery (that is, both the cluster route and the intra-group queuing route).
- The intra-group queuing route, being potentially faster, is selected in preference to the cluster route. Depending on the message characteristics, message delivery might be divided across the two paths. Hence, messages might be delivered out of sequence. It is important to note that this delivery is possible without regard to the MQOO_BIND_* option specified by the application. Intra-group queuing distributes messages in the same way as clustering does, depending on whether the MQOO_BIND_NOT_FIXED, MQOO_BIND_ON_OPEN, MQOO_BIND_ON_GROUP, or MQOO_BIND_AS_Q_DEF is specified on open.
- When a route has been selected, and messages have been placed on to the transmission queues, only the selected route is used for message delivery. Any unprocessed messages on the SYSTEM.QSG.TRANSMIT.QUEUE are not diverted to the SYSTEM.CLUSTER.TRANSMIT.QUEUE.

Clustering, intra-group queuing and distributed queuing:

It is possible to configure a queue manager that is a member of a cluster as well as a queue-sharing group and is connected to a distributed queue manager using a sender/receiver channel pair.

This configuration is a combination of distributed queuing with intra-group queuing and clustering with intra-group queuing.

Intra-group queuing is described in “Distributed queuing with intra-group queuing (multiple delivery paths)” on page 375.

Clustering with intra-group queuing is described in “Clustering with intra-group queuing (multiple delivery paths)” on page 377.

Intra-group queuing messages



This section describes the messages put to the SYSTEM.QSG.TRANSMIT.QUEUE.

Message structure

Like all other messages that are put to transmission queues, messages that are put to the SYSTEM.QSG.TRANSMIT.QUEUE are prefixed with the transmission queue header (MQXQH).

Message persistence

In IBM WebSphere MQ Version 5.3 and above, shared queues support both persistent and non-persistent messages.

If the queue manager terminates while the IGQ agent is processing non-persistent messages, or if the IGQ agent terminates abnormally while in the middle of processing messages, non-persistent messages being processed might be lost. Applications must make arrangements for the recovery of non-persistent messages if their recovery is required.

If a put request for a non-persistent message, issued by the IGQ agent, fails unexpectedly, the message being processed is lost.

Delivery of messages

The IGQ agent retrieves and delivers all nonpersistent messages outside of sync point scope, and all persistent messages within sync point scope. In this case, the IGQ agent acts as the sync point coordinator. The IGQ agent therefore processes nonpersistent messages like the way fast, nonpersistent messages are processed on a message channel. See Fast, nonpersistent messages.

Batching of messages

The IGQ agent uses a fixed batch size of 50 messages. Any persistent messages retrieved within a batch are committed at intervals of 50 messages. The agent commits a batch consisting of persistent messages when there are no more messages available for retrieval on the SYSTEM.QSG.TRANSMIT.QUEUE.

Message size

The maximum size of message that can be put to the SYSTEM.QSG.TRANSMIT.QUEUE is the maximum supported message length for shared queues minus the length of a transmission queue header (MQXQH).

Default message persistence and default message priority

If the SYSTEM.QSG.TRANSMIT.QUEUE is in the queue name resolution path established at open time, then for messages that are put with default persistence and default priority (or with default persistence or default priority), the normal rules are applied in the selection of the queue that has default priority and persistence values that are used. (See the IBM MQ messages section for more information about the rules of queue selection).

Related concepts:

“Undelivered/unprocessed messages”

This topic describes what happens to undelivered and unprocessed messages on the SYSTEM.QSG.TRANSMIT.QUEUE.

“Report messages - Intra Group Queuing” on page 382

This topic describes the report messages: Confirmation of arrival, confirmation of delivery, expiry report, and exception report.

Undelivered/unprocessed messages:

This topic describes what happens to undelivered and unprocessed messages on the SYSTEM.QSG.TRANSMIT.QUEUE.

If an IGQ agent cannot deliver a message to the destination queue, the IGQ agent:

- Honors the MQRO_DISCARD_MSG report option (if the Report options field of the MQMD for the undelivered message indicates that it must) and discards the undelivered message.
- Attempts to place the undelivered message on to the dead letter queue for the destination queue manager, if the message has not already been discarded. The IGQ agent prefixes the message with a dead letter queue header (MQDLH).

If a dead letter queue is not defined, or if an undelivered message cannot be put to the dead letter queue, and if the undelivered message is:

- persistent, the IGQ agent backs out the current batch of persistent messages that it is processing, and enters a state of retry. For more information, see “Retry capability of the intra-group queuing agent” on page 384.
- non-persistent, the IGQ agent discards the message and continues to process the next message.

If a queue manager in a queue-sharing group is terminated before its associated IGQ agent has had time to process all its messages, the unprocessed messages remain on the SYSTEM.QSG.TRANSMIT.QUEUE until the queue manager is next started. The IGQ agent then retrieves and delivers the messages to the destination queues.

If the coupling facility fails before all the messages on the SYSTEM.QSG.TRANSMIT.QUEUE have been processed, any unprocessed non-persistent messages are lost.

IBM recommends that applications do not put messages directly to transmission queues. If an application does put messages directly to the SYSTEM.QSG.TRANSMIT.QUEUE, the IGQ agent might not be able to process these messages and they remain on the SYSTEM.QSG.TRANSMIT.QUEUE. Users then have to use their own methods to deal with these unprocessed messages.

Report messages - Intra Group Queuing:

This topic describes the report messages: Confirmation of arrival, confirmation of delivery, expiry report, and exception report.

Confirmation of arrival (COA)/confirmation of delivery (COD) report messages

COA and COD messages are generated by the queue manager, when intra-group queuing is used.

Expiry report messages

Expiry report messages are generated by the queue manager.

Exception report messages

Depending on the MQRO_EXCEPTION_* report option specified in the *Report options* field of the message descriptor for the undelivered message, the IGQ agent generates the required exception report and places it on the specified reply-to queue. Intra-group queuing can be used to deliver the exception report to the destination reply-to queue.

The persistence of the report message is the same as the persistence of the undelivered message. If the IGQ agent fails to resolve the name of the destination reply-to queue, or if it fails to put the reply message to a transmission queue (for subsequent transfer to the destination reply-to queue) it attempts to put the exception report to the dead letter queue of the queue manager on which the report message is generated. If it is not possible, then if the undelivered message is:

- persistent, the IGQ agent discards the exception report, backs out the current batch of messages, and enters a state of retry. For more information, see “Retry capability of the intra-group queuing agent” on page 384.
- non-persistent, the IGQ agent discards the exception report and continues processing the next message on the SYSTEM.QSG.TRANSMIT.QUEUE.

Security for intra-group queuing



This topic describes the security arrangements for intra-group queuing.

Queue manager attributes IGQAUT (IGQ authority) and IGQUSER (IGQ agent user ID) can be set to control the level of security checking that is performed when the IGQ agent opens destination queues.

Intra-group queuing authority (IGQAUT)

The IGQAUT attribute can be set to indicate the type of security checks to be performed, and hence to determine the userids to be used by the IGQ agent when it establishes the authority to put messages on to the destination queue.

The IGQAUT attribute is analogous to the PUTAUT attribute that is available on channel definitions.

Intra-group queuing user identifier (IGQUSER)

The IGQUSER attribute can be used to nominate a user ID to be used by the IGQ agent when it establishes the authority to put messages on to a destination queue.

The IGQUSER attribute is analogous to the MCAUSER attribute that is available on channel definitions.

Specific properties of intra-group queuing

z/OS

This section describes the specific properties of intra-group queuing including Invalidation of object handles, self recovery and retry capability of the intra-group queuing agent, and the intra-group queuing agent and serialization.

Invalidation of object handles (MQRC_OBJECT_CHANGED)

If the attributes of an object are found to have changed after the object is opened, the queue manager invalidates the object handle with MQRC_OBJECT_CHANGED on its next use.

Intra-group queuing introduces the following rules for object handle invalidation:

- If the SYSTEM.QSG.TRANSMIT.QUEUE was included in the name resolution path during open processing because intra-group queuing was ENABLED at open time, but intra-group queuing is found to be DISABLED at put time, then the queue manager invalidates the object handle and fails the put request with MQRC_OBJECT_CHANGED.
- If the SYSTEM.QSG.TRANSMIT.QUEUE was not included in the name resolution path during open processing because intra-group queuing was DISABLED at open time, but intra-group queuing is found to be ENABLED at put time, then the queue manager invalidates the object handle and fails the put request with MQRC_OBJECT_CHANGED.
- If the SYSTEM.QSG.TRANSMIT.QUEUE was included in the name resolution path during open processing because intra-group queuing was enabled at open time, but the SYSTEM.QSG.TRANSMIT.QUEUE definition is found to have changed by put time, then the queue manager invalidates the object handle and fails the put request with MQRC_OBJECT_CHANGED.

Self recovery of the intra-group queuing agent

If the IGQ agent terminates abnormally, message CSQM067E is issued and the IGQ agent starts again.

Retry capability of the intra-group queuing agent

If the IGQ agent encounters a problem accessing the SYSTEM.QSG.TRANSMIT.QUEUE (because it is not defined, for example, or is defined with incorrect attributes, or is inhibited for Gets, or for some other reason), the IGQ agent goes into the state of retry.

The IGQ agent observes short and long retry counts and intervals. The values for these counts and intervals, which cannot be changed, are as follows:

Table 24. Short and long retry counts and intervals values

Constant	Value
Short retry count	10
Short retry interval	60 seconds = 1 min
Long retry count	999,999,999
Long retry interval	1200 seconds = 20 min

The intra-group queuing agent and serialization

An attempt, by the IGQ agent to serialize access to shared queues while peer recovery is still in progress might fail.

If there is a failure of a queue manager in a queue-sharing group while the IGQ agent is dealing with uncommitted messages on a shared queue or queues, the IGQ agent ends, and shared queue peer

recovery takes place for the failing queue manager. Because shared queue peer recovery is an asynchronous activity, it leaves the possibility for the failing queue manager, and also the IGQ agent for that queue manager, to restart before shared queue peer recovery is complete. Which in turn leaves the possibility for any committed messages to be processed ahead of and out of sequence with the messages still being recovered. To ensure that messages are not processed out of sequence, the IGQ agent serializes access to shared queues by issuing the MQCONN API call.

An attempt, by the IGQ agent to serialize access to shared queues while peer recovery is still in progress might fail. An error message is issued and the IGQ agent is put into retry state. When queue manager peer recovery is complete, for example at the time of the next retry, the IGQ agent can start.

Invalidation of object handles (MQRC_OBJECT_CHANGED):

If the attributes of an object are found to have changed after the object is opened, the queue manager invalidates the object handle with MQRC_OBJECT_CHANGED on its next use.

This information has moved. See “Specific properties of intra-group queuing” on page 383.

Self recovery of the intra-group queuing agent:

If the IGQ agent terminates abnormally, message CSQM067E is issued and the IGQ agent starts again.

This information has moved. See “Specific properties of intra-group queuing” on page 383.

Retry capability of the intra-group queuing agent:

If the IGQ agent encounters a problem accessing the SYSTEM.QSG.TRANSMIT.QUEUE (because it is not defined, for example, or is defined with incorrect attributes, or is inhibited for Gets, or for some other reason), the IGQ agent goes into the state of retry.

This information has moved. See “Specific properties of intra-group queuing” on page 383.

The intra-group queuing agent and Serialization:

An attempt, by the IGQ agent to serialize access to shared queues while peer recovery is still in progress might fail.

This information has moved. See “Specific properties of intra-group queuing” on page 383.

Storage management on z/OS

z/OS

IBM MQ for z/OS requires permanent and temporary data structures and uses page sets and memory buffers to store this data. These topics give more details on how IBM MQ uses these page sets and buffers.

Related concepts:

“Page sets for IBM MQ for z/OS”

Use this topic to understand how IBM MQ for z/OS uses pages sets to store messages.

“Storage classes for IBM MQ for z/OS” on page 386

A storage class is an IBM MQ for z/OS concept that allows the queue manager to map queues to page sets. You can use storage classes to control which data sets are used by which queues.

“Buffers and buffer pools for IBM MQ for z/OS” on page 388

IBM MQ for z/OS uses buffers and buffer pools to temporarily cache data. Use this topic to further understand how buffers are organized, and used.

Related reference:

“Where to find more information about storage management for IBM MQ for z/OS” on page 390

Use this topic as a reference to find further information about storage management for IBM MQ for z/OS.

Page sets for IBM MQ for z/OS



Use this topic to understand how IBM MQ for z/OS uses pages sets to store messages.

A *page set* is a VSAM linear data set that has been specially formatted to be used by IBM MQ. Page sets are used to store most messages and object definitions.

The exceptions to this are global definitions, which are stored in a shared repository on Db2, and the messages on shared queues. These are not stored on the queue manager page sets. For more information about shared queues, see “Shared queues and queue-sharing groups” on page 322, and for more information about global definitions, see Private and global definitions.

IBM MQ page sets can be up to 64 GB in size. Each page set is identified by a page set identifier (PSID), an integer in the range 00 through 99. Each queue manager must have its own page sets.

IBM MQ uses page set zero (PSID=00) to store object definitions and other important information relevant to the queue manager. For normal operation of IBM MQ it is essential that page set zero does not become full, so do not use it to store messages.

To improve the performance of your system, you should also separate short-lived messages from long-lived messages by placing them on different page sets.

You must format page sets, and IBM MQ provides a FORMAT utility for this; see Formatting page sets (FORMAT). Page sets must also be defined to the IBM MQ subsystem.

IBM MQ for z/OS can be configured to expand a page set dynamically if it becomes full. IBM MQ continues to expand the page set if required until 119 logical extents exist, if there is sufficient disk storage space available. The extents can span volumes if the linear data set is defined in this way, however, IBM MQ cannot expand the page sets beyond 64 GB.

You cannot use page sets from one IBM MQ queue manager on a different IBM MQ queue manager, or change the queue manager name. If you want to transfer the data from one queue manager to another, you must unload all the objects and messages from the first queue manager and reload them onto another.

It is not possible to use page sets greater than 4 GB in a queue manager running a release earlier than V6. During the migration period, when it is likely that you might need to fall back to a previous release of code:

- Do not change page set 0 to be greater than 4 GB.

- Other page sets greater than 4 GB will be left offline when restarting a queue manager with a previous release.

For further information about migrating existing page sets capable of expanding beyond 4 GB, see [Defining a page set to be larger than 4 GB](#).

It is possible for an administrator to dynamically add page sets to a running queue manager, or remove page sets from a running queue manager (except for page set zero). The DEFINE PSID command can run after the queue manager restart has completed, only if the command contains the DSN keyword.

Storage classes for IBM MQ for z/OS

z/OS

A storage class is an IBM MQ for z/OS concept that allows the queue manager to map queues to page sets. You can use storage classes to control which data sets are used by which queues.

Introducing storage classes

A *storage class* maps one or more queues to a page set. This means that messages for that queue are stored on that page set.

Storage classes allow you to control where non-shared message data is stored for administrative, data set space and load management, or application isolation purposes. You can also use storage classes to define the XCF group and member name of an IMS region if you are using the IMS bridge (described in “IBM MQ and IMS” on page 445).

Shared queues do not use storage classes to obtain a page set mapping because the messages on them are not stored on page sets.

How storage classes work

- You define a storage class, using the DEFINE STGCLASS command, specifying a page set identifier (PSID).
- When you define a queue, you specify the storage class in the STGCLASS attribute.

In the following example, the local queue QE5 is mapped to page set 21 through storage class ARC2.

```
DEFINE STGCLASS(ARC2) PSID(21)
DEFINE QLOCAL(QE5) STGCLASS(ARC2)
```

This means that messages that are put on the queue QE5 are stored on page set 21 (if they stay on the queue long enough to be written to DASD).

More than one queue can use the same storage class, and you can define as many storage classes as you like. For example, you can extend the previous example to include more storage class and queue definitions, as follows:

```

DEFINE STGCLASS(ARC1) PSID(05)
DEFINE STGCLASS(ARC2) PSID(21)
DEFINE STGCLASS(MAX1) PSID(05)
DEFINE QLOCAL(QE1) STGCLASS(ARC1) ...
DEFINE QLOCAL(QE2) STGCLASS(ARC1) ...
DEFINE QLOCAL(QE3) STGCLASS(MAX1) ...
DEFINE QLOCAL(QE4) STGCLASS(ARC2) ...
DEFINE QLOCAL(QE5) STGCLASS(ARC2) ...

```

In Figure 71, both storage classes ARC1 and MAX1 are associated with page set 05. Therefore, the queues QE1, QE2, and QE3 are mapped to page set 05. Similarly, storage class ARC2 associates queues QE4 and QE5 with page set 21.

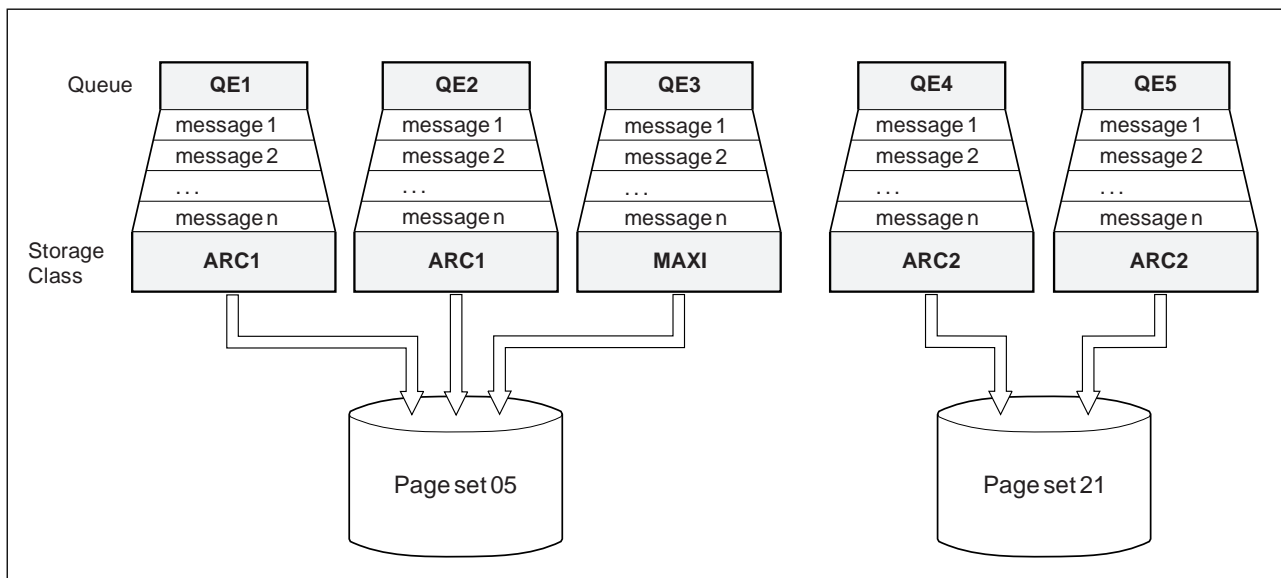


Figure 71. Mapping queues to page sets through storage classes

If you define a queue without specifying a storage class, IBM MQ uses a default storage class.

If a message is put on a queue that names a nonexistent storage class, the application receives an error. You must alter the queue definition to give it an existing storage class name, or create the storage class named by the queue.

You can change a storage class only when:

- All queues that use this storage class are empty, and have no uncommitted activity.
- All queues that use this storage class are closed.

Buffers and buffer pools for IBM MQ for z/OS

z/OS

IBM MQ for z/OS uses buffers and buffer pools to temporarily cache data. Use this topic to further understand how buffers are organized, and used.

For efficiency, IBM MQ uses a form of caching whereby messages (and object definitions) are stored temporarily in buffers before being stored in page sets on DASD. Short-lived messages, that is, messages that are retrieved from a queue shortly after they are received, might only ever be stored in the buffers. This caching activity is controlled by a buffer manager, which is a component of IBM MQ.

The buffers are organized into *buffer pools*. You can define up to 16 buffer pools (0 through 15) for each queue manager.

If OPMODE is set to OPMODE=(NEWFUNC, 800), you can define up to 100 buffer pools (0 through 99) for each queue manager.

You are recommended to use the minimal number of buffer pools consistent with the object and message type segregation outlined in Figure 72 on page 389, and any data isolation requirements your application might have. Each buffer is 4 KB long. Buffer pools use 31 bit storage by default, in this mode, the maximum number of buffers is determined by the amount of 31 bit storage available in the queue manager address space; do not use more than about 70% for buffers. Alternatively, buffer pool storage allocation can be made from 64 bit storage (use the LOCATION attribute of the **DEFINE BUFFPOOL** command). Using LOCATION(ABOVE) so that 64 bit storage is used has two benefits. Firstly, there is much more 64 bit storage available so buffer pools can be much bigger, and secondly, 31 bit storage is made available for use by other functions. Typically, the more buffers you have, the more efficient the buffering and the better the performance of IBM MQ.

Figure 72 on page 389 shows the relationship between messages, buffers, buffer pools, and page sets. A buffer pool is associated with one or more page sets; each page set is associated with a single buffer pool.

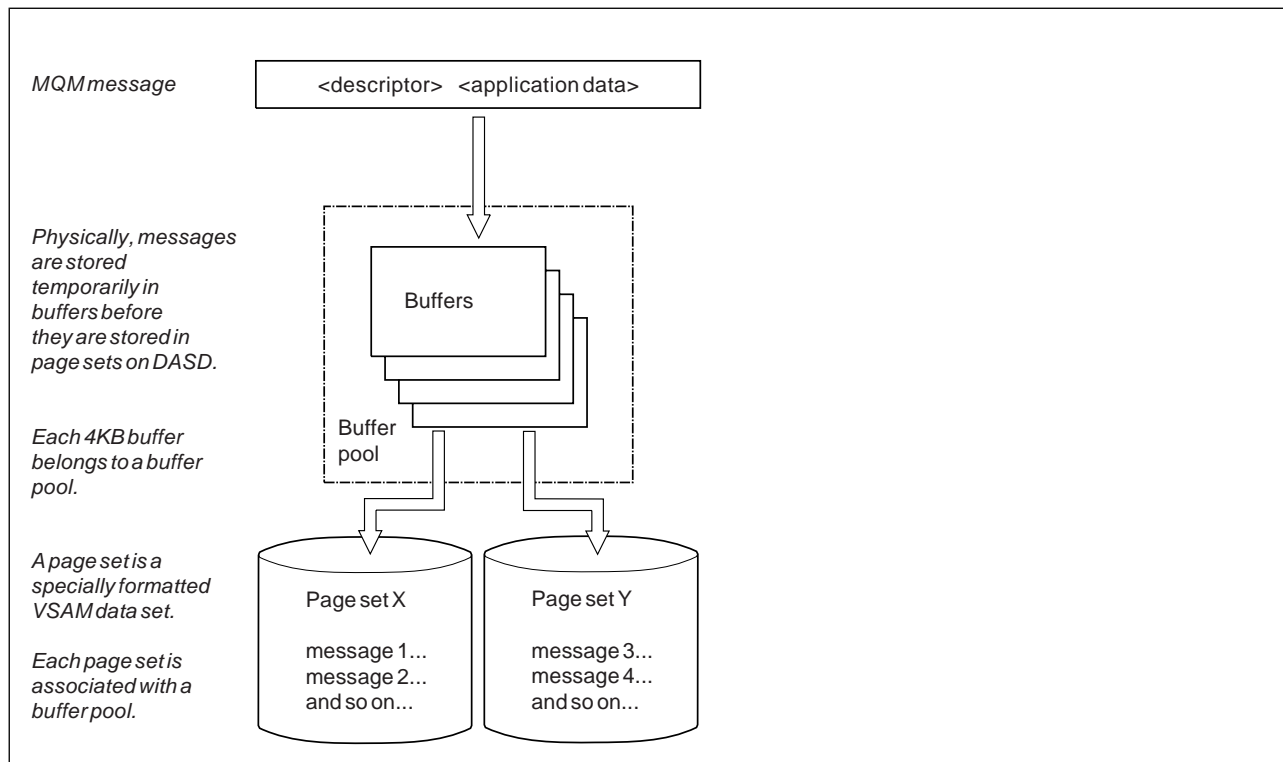


Figure 72. Buffers, buffer pools, and page sets

You can dynamically issue commands to modify buffer pool size, and location, using the **ALTER BUFFPOOL** command. Pagesets can be dynamically added by using the **DEFINE PSID** command, or deleted by using the **DELETE PSID** command.

If a buffer pool is too small, IBM MQ issues message CSQP020E. You can then dynamically add more buffers to the affected buffer pool (note that you may have to remove buffers from other buffer pools to do this).

You specify the number of buffers in a pool with the **DEFINE BUFFPOOL** command, and you can dynamically resize buffer pools with the **ALTER BUFFPOOL** command. You determine the current number of buffers in a pool dynamically by displaying a page set that uses the buffer pool, using the **DISPLAY USAGE** command.

For performance reasons, do not put messages and object definitions in the same buffer pool. Use one buffer pool (say number zero) exclusively for page set zero, where the object definitions are kept. Similarly, keep short-lived messages and long-lived messages in different buffer pools and therefore on different page sets, and in different queues.

The **DEFINE BUFFPOOL** command cannot be used after restart to create a new buffer pool. Instead, if a **DEFINE PSID** command uses the DSN keyword, it can explicitly identify a buffer pool that is not currently defined. That new buffer pool will then be created.

Where to find more information about storage management for IBM MQ for z/OS

z/OS

Use this topic as a reference to find further information about storage management for IBM MQ for z/OS.

You can find more information about the topics in this section from the following sources:

Table 25. Where to find more information about storage management

Topic	Where to look
How much storage you need	Planning your storage and performance requirements on z/OS
How large to make your page sets and buffer pools	Plan your page sets and buffer pools
Managing page sets	Managing page sets
MQSC commands	The MQSC commands

Logging in IBM MQ for z/OS

z/OS

IBM MQ maintains *logs* of data changes and significant events as they occur. These logs can be used to recover data to a previous state if required.

The *bootstrap data set* (BSDS) stores information about the data sets that contain the logs.

The log does not contain information for statistics, traces, or performance evaluation. For further details about the statistical and monitoring information that IBM MQ collects, see *Monitoring and statistics*.

For more information about logging, see the following topics:

- “Log files in IBM MQ for z/OS” on page 391
- “How the log is structured” on page 395
- “How the IBM MQ for z/OS logs are written” on page 395
- “Larger log Relative Byte Address” on page 398
- “The bootstrap data set” on page 399

Related information:

Planning your logging environment

Setting logs using the system parameter module

▶ z/OS Administering z/OS

▶ z/OS Messages for IBM MQ for z/OS

MQSC reference

Log files in IBM MQ for z/OS

▶ z/OS

Log files contain information needed for transaction recovery. Active log files can be archived so that you can keep log data for a long period.

What is a log file

IBM MQ records all significant events as they occur in an *active log*. The log contains the information needed to recover:

- Persistent messages
- IBM MQ objects, such as queues
- The IBM MQ queue manager

The active log comprises a collection of data sets (up to 310) which are used cyclically.

You can enable log archiving so that when an active log fills a copy is made in an archive data set. Using archiving allows you to keep log data for an extended period. If you do not use archiving, the logs wrap and earlier data is overwritten. To recover a page set, or recover data in a CF structure, you need log data from when the backup of the page set or structure was taken. An archive log can be created on disk or on tape.

Archiving

Because the active log has a fixed size, IBM MQ copies the contents of each log data set periodically to an *archive log*, which is normally a data set on a direct-access storage device (DASD) or a magnetic tape. If there is a subsystem or transaction failure, IBM MQ uses the active log and, if necessary, the archive log for recovery.

The archive log can contain up to 1000 sequential data sets. You can catalog each data set using the z/OS integrated catalog facility (ICF).

Archiving is an essential component of IBM MQ recovery. If a unit of recovery is a long-running one, log records within that unit of recovery might be found in the archive log. In this case, recovery requires data from the archive log. However, if archiving is disabled, the active log with new log records wraps, overwriting earlier log records. This means that IBM MQ might not be able to back out the unit of recovery and messages might be lost. The queue manager then terminates abnormally.

Therefore, in a production environment, **never switch archiving off**. If you do, you run the risk of losing data after a system or transaction failure. Only if you are running in a test environment can you consider switching archiving off. If you need to do this, use the CSQ6LOGP macro, which is described in Using CSQ6LOGP.

To help prevent problems with unplanned long-running units of work, IBM MQ issues a message (CSQJ160I or CSQJ161I) if a long-running unit of work is detected during active log offload processing.

Dual logging

In dual logging, each log record is written to two different active log data sets to minimize the likelihood of data loss problems during restart.

You can configure IBM MQ to run with either *single logging* or *dual logging*. With single logging, log records are written once to an active log data set. Each active log data set is a single-extent VSAM linear data set (LDS). With dual logging, each log record is written to two different active log data sets. Dual logging minimizes the likelihood of data loss problems during restart.

Log shunting

Log shunting causes the log records for some units of work to be written further down the log. This reduces the amount of log data that must be read at queue manager restart, or backout, for long running or long term in-doubt units of work.

When a unit of work is considered to be long, a representation of each log record is written further down the log. This technique is known as *shunting*. When the whole of the unit of work has been processed, the unit of work is in a *shunted* state. Any backout or restart activity relating to the shunted unit of work can use the shunted log records instead of using the original unit of work log records.

Detecting a long-running unit of work is a function of the checkpoint process. At checkpoint time, each active unit of work is checked to establish whether it needs to be shunted. If the unit of work has been through two prior checkpoints since it was created, or since it was last shunted, the unit of work is suitable to be shunted. This means that a single unit of work might be shunted more than once. This is known as a *multi-shunted* unit of work.

A unit of work is shunted every three checkpoints. However the checkpoint is performed asynchronously to the log-switch (or the writing of the log record which caused LOGLOAD to be exceeded).

There is only a single checkpoint taking place at a time, so there might be multiple log-switches before a checkpoint completes.

This means that if there are not enough active logs, or if they are too small, then shunting of a large unit of work might not complete before all the logs are filled.

Message CSQR027I results if shunting is unable to complete.

If log archiving is turned off, ABEND 5C6 with reason 00D1032A occurs if there is an attempt to back out the unit of work for which shunting failed. To avoid this problem you should use OFFLOAD=YES.

Log shunting is always active, and runs whether log archiving is enabled or not.

Note: Although all log records for a unit of work are shunted, the entire content of each record is not shunted, only the part that is necessary for backout. This means that the amount of log data written is kept to a minimum, and that shunted records cannot be used if a page set failure occurs. A long running unit of work is one that has been running for more than three queue manager checkpoints.

For more information about log shunting, see *Managing the logs*.

Log compression

You can configure IBM MQ for z/OS to compress and decompress log records as they are written and read from the log data set.

Log compression can be used to reduce the amount of data written to the log for persistent messages on private queues. The amount of compression that is achieved depends on the type of data contained within messages. For example, Run Length Encoding (RLE) works by compacting repeated instances of bytes which can give good results efficiently for structured or record oriented data.

You can use fields within the Log manager section of the System Management Facility 115 (SMF) records to monitor how much data compression is achieved. For more information about SMF, see *Using the System Management Facility and Accounting and statistics messages*.

Log compression increases the processor utilization of the system. You should only consider using compression if throughput of your queue manager is constrained by the IO bandwidth writing to the log data sets or you are constrained by the disk storage needed to hold log data sets. If you are using shared queues then IO bandwidth constraints can be relieved by adding additional queue managers to the queue-sharing group and distributing the workload across more queue managers.

The log compression option can be enabled and disabled as required without the need to stop and restart the queue manager. The queue manager can read any compressed log records regardless of the current log compression setting.

The queue manager supports 3 settings for log compression.

NONE

No log data compression is used. This is the default value.

RLE Log data compression is performed using run-length encoding (RLE).

ANY Enable the queue manager to select the compression algorithm that gives the greatest degree of log record compression. This option results in RLE compression.

You can control the compression of log records using one of the following:

- The SET and DISPLAY LOG commands in MQSC; see SET LOG and DISPLAY LOG
- The Set Log and Inquire Log functions in the PCF interface; see Set log and Inquire log
- The CSQ6LOGP macro in the system parameter module; see Using CSQ6LOGP

In addition the Log Print utility CSQ1LOGP has support for expanding any compressed log records.

Log data

The log can contain up to 18 million million million (1.8×10^{19}) bytes. Each byte can be addressed by its offset from the beginning of the log, and that offset is known as its *relative byte address* (RBA).

The RBA is referenced by a 6-byte or 8-byte field giving a total addressable range of 2^{48} bytes, or 2^{64} bytes, depending on whether 6-byte or 8-byte log RBAs are in use.

However, when IBM MQ detects that the used range is beyond F00000000000 (if 6-byte RBAs are in use) or FFFF800000000000 (if 8-byte log RBAs are in use), messages CSQI045, CSQI046, CSQI047, and CSQJ032 are issued, warning you to reset the log RBA.

If the RBA value reaches FFF800000000 (if 6-byte log RBAs are in use) or FFFFFFFC00000000 (if 8-byte log RBAs are in use) the queue manager terminates with reason code 00D10257.

Once the warning messages about the used log range are being issued, you should plan a queue manager outage during which the queue manager can be converted to use 8-byte log RBAs, or the log can be reset. The procedure to reset the log is documented in *Resetting the queue manager's log*.

If your queue manager is using 6-byte log RBAs, consider converting the queue manager to use 8-byte log RBAs rather than resetting the queue manager's log, following the procedure documented in *Implementing the larger log Relative Byte Address*.

The log consists of *log records*, each of which is a set of log data treated as a single unit. A log record is identified either by the RBA of the first byte of its header, or by its log record sequence number (LRSN). The RBA or LRSN uniquely identifies a record that starts at a particular point in the log.

Whether you use the RBA or LRSN to identify log points depends on whether you are using queue-sharing groups. In a queue-sharing environment, you cannot use the relative byte address to uniquely identify a log point, because multiple queue managers can update the same queue at the same time, and each has its own log. To solve this, the log record sequence number is derived from a timestamp value, and does not necessarily represent the physical displacement of the log record within the log.

Each log record has a header that gives its type, the IBM MQ subcomponent that made the record, and, for unit of recovery records, a unit of recovery identifier.

There are four types of log record, described under the following headings:

- Unit of recovery log records
- Checkpoint records
- Page set control records
- CF structure backup records

Unit of recovery log records

Most of the log records describe changes to IBM MQ queues. All such changes are made within units of recovery.

IBM MQ uses special logging techniques involving *undo/redo* and *compensating log records* to reduce restart times and improve system availability.

One effect of this is that the restart time is bounded. If a failure occurs during a restart so that the queue manager has to be restarted a second time, all the recovery activity that completed to the point of failure in the first restart does not need to be reapplied during a second restart. This means that successive restarts do not take progressively longer times to complete.

Checkpoint records

To reduce restart time, IBM MQ takes periodic checkpoints during normal operation. These occur as follows:

- When a predefined number of log records has been written. This number is defined by the checkpoint frequency operand called LOGLOAD of the system parameter macro CSQ6SYSP, described in *Using CSQ6SYSP*.
- At the end of a successful restart.
- At normal termination.
- Whenever IBM MQ switches to the next active log data set in the cycle.

At the time a checkpoint is taken, IBM MQ issues the DISPLAY CONN command (described in DISPLAY CONN) internally so that a list of connections currently in doubt is written to the z/OS console log.

Page set control records

These records register the page sets and buffer pools known to the IBM MQ queue manager at each checkpoint, and record information about the log ranges required to perform media recovery of the page set at the time of the checkpoint.

Certain dynamic changes to page sets and buffer pools are also written as page set control records, so that the changes can be recovered and automatically reinstated at the next queue manager restart.

CF structure backup records

These records hold data read from a coupling facility list structure in response to a BACKUP CFSTRUCT command. In the unlikely event of a coupling facility structure failure, these records are used, together with unit of recovery records, by the RECOVER CFSTRUCT command to perform media recovery of the coupling facility structure to the point of failure.

Related information:

Implementing the larger log Relative Byte Address

How the log is structured

z/OS

Use this topic to understand the terminology used to describe log records.

Each active log data set must be a VSAM linear data set (LDS). The physical output unit written to the active log data set is a 4 KB control interval (CI). Each CI contains one VSAM record.

Physical and logical log records

One VSAM CI is a *physical* record. The information logged at a particular time forms a *logical* record, with a length that varies independently of the space available in the CI. So one physical record might contain:

- Several logical records
- One or more logical records and part of another logical record
- Part of one logical record only

The term *log record* refers to the *logical* record, regardless of how many *physical* records are needed to store it.

How the IBM MQ for z/OS logs are written

z/OS

Use this topic to understand how IBM MQ processes log file records.

IBM MQ writes each log record to a DASD data set called the *active log*. When the active log is full, IBM MQ copies its contents to a DASD or tape data set called the *archive log*. This process is called *offloading*.

Figure 73 on page 396 illustrates the process of logging. Log records typically go through the following cycle:

1. IBM MQ notes changes to data and significant events in recovery log records.
2. IBM MQ processes recovery log records and breaks them into segments, if necessary.
3. Log records are placed sequentially in *output log buffers*, which are formatted as VSAM Control Intervals (CI). Each log record is identified by a relative byte address in the range zero through $2^{64}-1$.
4. The CIs are written to a set of predefined DASD active log data sets, which are used sequentially and recycled.

- If archiving is active, as each active log data set becomes full, its contents are automatically offloaded to a new archive log data set.

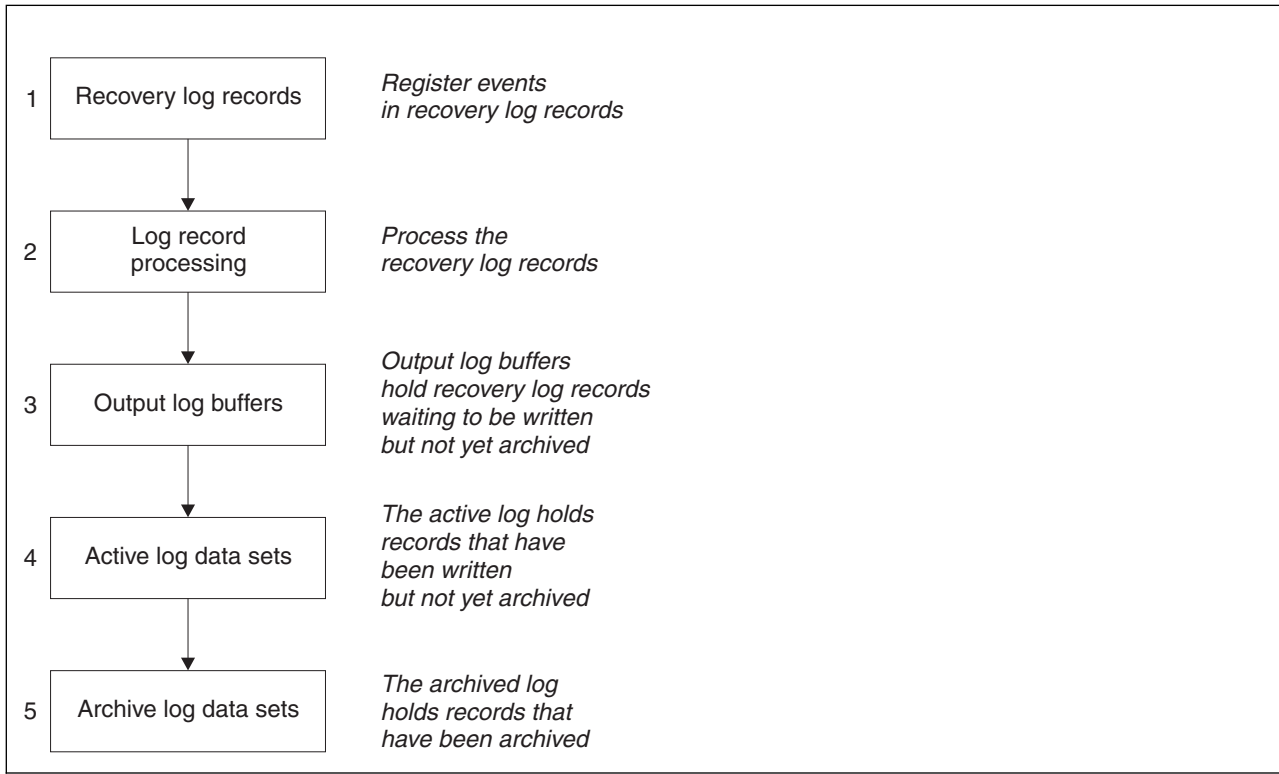


Figure 73. The logging process

When the active log is written

The in-storage log buffers are written to an active log data set whenever any of the following occur:

- The log buffers become full.
- The write threshold is reached (as specified in the CSQ6LOGP macro).
- Certain significant events occur, such as a commit point, or when an IBM MQ BACKUP CFSTRUCT command is issued.

When the queue manager is initialized, the active log data sets named in the BSDS are dynamically allocated for exclusive use by the queue manager and remain allocated exclusively to IBM MQ until the queue manager terminates.

Dynamically adding log data sets

It is possible to dynamically define new active log data sets while the queue manager is running. This feature alleviates the problem of a queue manager hang when archiving is not able to offload active logs due to a transient problem. See the DEFINE LOG command for more information.

Note: To redefine or remove active logs you must terminate and restart the queue manager.

IBM MQ and Storage Management Subsystem

IBM MQ parameters enable you to specify Storage Management Subsystem (MVS/DFP SMS) storage classes when allocating IBM MQ archive log data sets dynamically. IBM MQ initiates the archiving of log data sets, but you can use SMS to perform allocation of the archive data set.

Related reference:

“When the IBM MQ for z/OS archive log is written”

Use this topic to understand the process of copying active logs to archive logs, and when the process occurs.

When the IBM MQ for z/OS archive log is written: z/OS

Use this topic to understand the process of copying active logs to archive logs, and when the process occurs.

The process of copying active logs to archive logs is called *offloading*. The relation of offloading to other logging events is shown schematically in Figure 74.

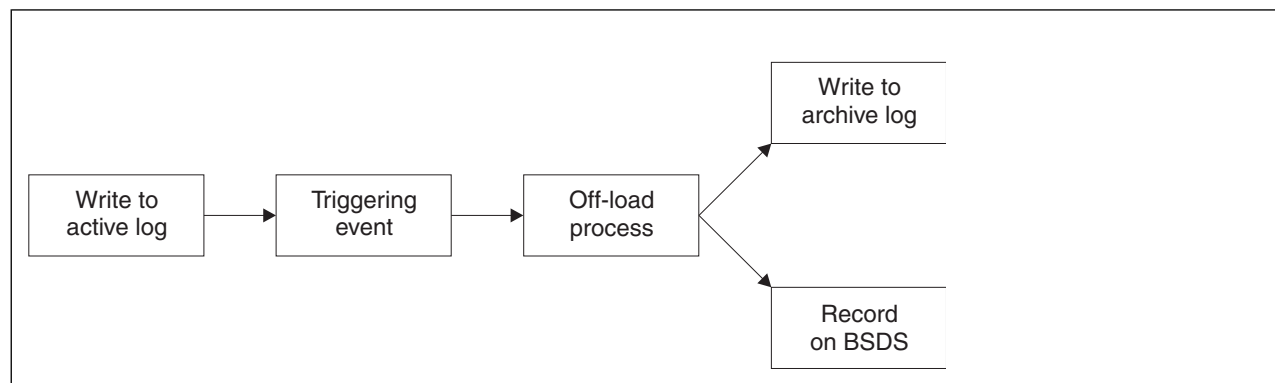


Figure 74. The offloading process

Triggering the offloading process

The offload process of an active log to an archive log can be triggered by several events. For example:

- Filling an active log data set.
- Using the MQSC ARCHIVE LOG command.
- An error occurring while writing to an active log data set.

The data set is truncated before the point of failure, and the record that was not written becomes the first record of the new data set. Offloading is triggered for the truncated data set as for a normal full log data set. If there are dual active logs, both copies are truncated so that the two copies remain synchronized.

Message CSQJ110E is issued when the last available active log is 5% full and at 5% increments thereafter, stating the percentage of the log's capacity that is in use. If all the active logs become full, IBM MQ stops processing, until offloading occurs, and issues this message:

```
CSQJ111A +CSQ1 OUT OF SPACE IN ACTIVE LOG DATA SETS
```

The offload process

When all the active logs become full, IBM MQ runs the offloading process and halts processing until the offloading process has been completed. If the offload processing fails when the active logs are full, IBM MQ abends.

When an active log is ready to be offloaded, a request is sent to the z/OS console operator to mount a tape or prepare a DASD unit. The value of the ARCWTOR logging option (for further information, see Using CSQ6ARVP) determines whether the request is received. If you are using tape for offloading, specify ARCWTOR=YES. If the value is YES, the request is preceded by a WTOR (message number CSQJ008E) telling the operator to prepare an archive log data set to be allocated.

The operator need not respond to this message immediately. However, delaying the response delays the offload process. It does not affect IBM MQ performance unless the operator delays the response for so long that IBM MQ runs out of active logs.

The operator can respond by canceling the offload process. In this case, if the allocation is for the first copy of dual archive data sets, the offload process is merely delayed until the next active log data set becomes full. If the allocation is for the second copy, the archive process switches to single copy mode, but for this data set only.

Interruptions and errors while offloading

A request to stop the queue manager does not take effect until offload processing has finished. If IBM MQ fails while offloading is in progress, offloading begins again when the queue manager is restarted.

Messages during offload processing

Offloaded messages are sent to the z/OS console by IBM MQ and the offloading process. You can use these messages to find the RBA ranges in the various log data sets.

Larger log Relative Byte Address

z/OS

This function improves the availability of the queue manager by increasing the period of time before you have to reset the log.

Recovery data is written to the log so that persistent messages are available when the queue manager is restarted. The term log Relative Byte Address (log RBA) is used to refer to the location of data as an offset from the beginning of the log.

Before IBM MQ Version 8.0, the 6 byte log RBA could address up to 256 terabytes of data. Before this quantity of log records has been written, you have to reset the queue manager's log by following the procedure documented in *Resetting the queue manager's log*.

Resetting the logs of queue managers is not a quick process, and can require an extended outage, due to the need to reset the page sets as part of the process. For a high use queue manager this operation might typically be done once a year.

From IBM MQ Version 8.0, the log RBA can be 8 bytes long and the queue manager can now address over 64,000 times as much data (16 exabytes) before the log RBA needs to be reset. The impact of using the larger log RBA is that the size of the log data written increases by a few bytes.

When is this function enabled?

You can enable this function at any time, although ideally you should plan for the change and explicitly enable it at a time to suit your enterprise. See *Planning to increase the maximum addressable log range* for guidance on planning the migration to 8 byte log RBAs.

Important: Before you can use this capability in a queue-sharing group, all the queue managers in that group need to be running IBM MQ Version 8.0 or later and have Version 8.0 or later new functions enabled.

If the current log RBA is approaching the end of the log RBA range, consider converting the queue manager to use 8 byte log RBAs rather than resetting the queue manager's log. Converting a queue manager to use 8 byte log RBAs requires a shorter outage than resetting the log, and significantly increases the period of time before you have to reset the log.

Message CSQJ034I, issued during queue manager initialization, indicates the end of the log RBA range for the queue manager as configured, and can be used to determine whether 6-byte or 8 byte log RBAs are in use.

How is this function enabled?

8 byte log RBA is enabled by starting the queue manager with a version 2 format BSDS. In summary, this is achieved by:

1. Enabling version 8 new functions
2. Shutting down the queue manager cleanly
3. Running the BSDS conversion utility to create a copy of the BSDS in version 2 format.
4. Restarting the queue manager with the converted BSDS.

Once a queue manager has been converted to use 8 byte log RBAs, it cannot go back to using 6 byte log RBA.

See Implementing the larger log Relative Byte Address for the detailed procedure on how to enable 8 byte log RBAs.

Related information:

The BSDS conversion utility (CSQJUCNV)

Planning to increase the maximum addressable log range

The bootstrap data set



The bootstrap data set is required by IBM MQ as a mechanism to reference log data sets, and log records. This information is required during normal processing, and restart recovery.

What the bootstrap data set is for

The *bootstrap data set* (BSDS) is a VSAM key-sequenced data set (KSDS) that holds information needed by IBM MQ. It contains the following:

- An inventory of all active and archived log data sets known to IBM MQ. IBM MQ uses this inventory to:
 - Track the active and archived log data sets
 - Locate log records so that it can satisfy log read requests during normal processing
 - Locate log records so that it can handle restart processing

IBM MQ stores information in the inventory each time an archive log data set is defined or an active log data set is reused. For active logs, the inventory shows which are full and which are available for reuse. The inventory holds the relative byte address (RBA) of each portion of the log held in that data set.

- A *wrap-around* inventory of all recent IBM MQ activity. This is needed if you have to restart the queue manager.

The BSDS is required if the queue manager has an error and you have to restart it. IBM MQ **must** have a BSDS. To minimize the likelihood of problems during a restart, you can configure IBM MQ with dual BSDSs, each recording the same information. Using dual BSDSs is known as running in *dual mode*. If possible, place the copies on separate volumes. This reduces the risk of them both being lost if the volume is corrupted or destroyed. Use dual BSDSs rather than dual write to DASD.

The BSDS is set up when IBM MQ is customized and you can manage the inventory using the change log inventory utility (CSQJU003). For more information about this utility, see Administering IBM MQ for z/OS. It is referenced by a DD statement in the queue manager startup procedure.

Normally, IBM MQ keeps duplicate copies of the BSDS. If an I/O error occurs, it deallocates the failing copy and continues with a single BSDS. You can restore dual-mode operation, this is described in the Administering IBM MQ for z/OS.

The active logs are first registered in the BSDS when IBM MQ is installed. You cannot replace the active logs without terminating and restarting the queue manager.

Archive log data sets are allocated dynamically. When one is allocated, the data set name is registered in the BSDS. The list of archive log data sets expands as archives are added, and wraps when a user-determined number of entries has been reached. The maximum number of entries is 1000 for single archive logging and 2000 for dual logging.

You can use a tape management system to delete the archive log data sets (IBM MQ does not have an automated method). Therefore, the information about an archive log data set can be in the BSDS long after the archive log data set has been deleted by the system administrator.

Conversely, the maximum number of archive log data sets could have been exceeded, and the data from the BSDS dropped long before the data set has reached its expiry date.

You can use the following MQSC command to determine the extent of the log, and the name of the active or archive log data set holding the earliest log RBA, required for various types of media or queue manager recovery:

```
DISPLAY USAGE TYPE(DATASET)
```

If the system parameter module specifies that archive log data sets are cataloged when allocated, the BSDS points to the integrated catalog facility (ICF) catalog for the information needed for later allocations. Otherwise, the BSDS entries for each volume register the volume serial number and unit information that is needed for later allocations.

The BSDS version

The format of the BSDS varies according to its version. Increasing the version of the BSDS allows new features to be used. The following BSDS versions are supported by IBM MQ:

Version 1

Supported by all releases of IBM MQ. A version 1 BSDS supports 6-byte log RBA values.

Version 2

Supported by IBM MQ Version 8.0 and higher. A version 2 BSDS enables 8-byte log RBA values, and up to 310 data sets in each active log copy.

Version 3

Supported by IBM MQ Version 8.0 and higher. The BSDS is automatically converted to version 3 when more than 31 data sets are added to either active log copy.

You can determine the version of a BSDS by running the print log map utility (CSQJU004). To convert a BSDS from Version 1 to Version 2, run the BSDS conversion utility (CSQJUCNV).

See “Larger log Relative Byte Address” on page 398 for more information on 6-byte and 8-byte log RBAs.

Archive log data sets and BSDS copies

Each time a new archive log data set is created, a copy of the BSDS is also created. If the archive log is on tape, the BSDS is the first data set on the first output volume. If the archive log is on DASD, the BSDS is a separate data set.

The data set names of the archive log and the BSDS copy are the same, except that the lowest-level qualifier of the archive log name begins with A and the BSDS copy begins with B, for example:

Archive log name

CSQ.ARCHLOG1.E00186.T2336229. A 0000001

BSDS copy name

CSQ.ARCHLOG1.E00186.T2336229. B 0000001

If there is a read error while copying the BSDS, the copy is not created, message CSQJ125E is issued, and the offloading to the new archive log data set continues without the BSDS copy.

Defining your system on z/OS



IBM MQ for z/OS uses many default object definitions, and provides sample JCL to create those default objects. Use this topic to understand these default objects, and the sample JCL.

Setting system parameters

In IBM MQ for z/OS, a system parameter module controls the logging, archiving, tracing, and connection environments that IBM MQ uses in its operation. The system parameters are specified by three assembler macros, as follows:

CSQ6SYSP

System parameters, including setting the connection and tracing environment.

CSQ6LOGP

Logging parameters.

CSQ6ARVP

Log archive parameters.

Default parameter modules are supplied with IBM MQ for z/OS. If these do not contain the values that you want to use, you can create your own parameter modules using the sample supplied with IBM MQ. The sample is thlqual.SCSQPROC(CSQ4ZPRM).

You can alter some system parameters while a queue manager is running. See the SET SYSTEM, SET LOG, and SET ARCHIVE commands in The MQSC commands.

For more information about defining , see the following topics:

- “Defining system objects for IBM MQ for z/OS” on page 402
- “Tuning your queue manager on IBM MQ for z/OS” on page 406
- “Sample definitions supplied with for IBM MQ for z/OS” on page 407

Related information:

Customize the sample initialization input data sets

Administering z/OS

MQSC reference

Configuring clusters

Monitoring IBM MQ

Defining system objects for IBM MQ for z/OS

▶ z/OS

IBM MQ for z/OS requires additional predefined objects for publish/subscribe applications, cluster, and channel control and other system administration functions.

The system objects required by IBM MQ for z/OS can be divided into the following categories:

- Publish/subscribe objects
- System default objects
- System command objects
- System administration objects
- Channels queues
- Cluster queues
- Queue-sharing group queues
- Storage classes
- Defining the system object dead-letter queue
- Default transmission queue
- Internal queues
- “Channel authentication queue” on page 406

Publish/subscribe objects

There are several system objects that you need to define before you can use publish/subscribe applications with IBM MQ for z/OS. Sample definitions are supplied with IBM MQ to help you define these objects. These samples are described in CSQ4INSG.

To use publish/subscribe you need to define the following objects:

- A local queue called `SYSTEM.RETAINED.PUB.QUEUE`, which is used to hold a copy of each retained publication in the queue manager. Each full topic name could have up to one retained publication stored on this queue. If your applications will make use of retained publications on many different topics, or if your retained publication messages are large messages, the requirements for storage for this queue should be carefully planned, including assigning it to its own page set if the storage requirements for it are large. To improve performance, you should define this queue with an index type of `MSGID` (as shown in the supplied sample queue definition).
- A local queue called `SYSTEM.DURABLE.SUBSCRIBER.QUEUE`, which is used to hold a persistent copy of the durable subscriptions in the queue manager. To improve performance, you should define this queue with an index type of `CORRELID` (as shown in the supplied sample queue definition).
- A local queue called `SYSTEM.DURABLE.MODEL.QUEUE`, which is used as a model for managed durable subscriptions.
- A local queue called `SYSTEM.NDURABLE.MODEL.QUEUE`, which is used as a model for managed non-durable subscriptions.
- A namelist called `SYSTEM.QPUBSUB.QUEUE.NAMELIST`, which contains a list of queue names monitored by the queued publish/subscribe interface.

- A namelist called `SYSTEM.QPUBSUB.SUBPOINT.NAMELIST`, which contains a list of topic objects used by the queued publish/subscribe interface to match topic objects to subscription points.
- A topic called `SYSTEM.BASE.TOPIC`, which is used as a base topic for resolving attributes.
- A topic called `SYSTEM.BROKER.DEFAULT.STREAM`, which is the default stream used by the queued publish/subscribe interface.
- A topic called `SYSTEM.BROKER.DEFAULT.SUBPOINT`, which is the default RFH2 subscription point used by the queued publish/subscribe interface.
- A topic called `SYSTEM.BROKER.ADMIN.STREAM`, which is the admin stream used by the queued publish/subscribe interface.
- A subscription called `SYSTEM.DEFAULT.SUB`, which is a default subscription object used to provide default values on `DEFINE SUB` commands.

System default objects

System default objects are used to provide default attributes when you define an object and do not specify the name of another object to base the definition on.

The names of the default system object definitions begin with the characters "SYSTEM.DEFAULT" or "SYSTEM.DEF." For example, the system default local queue is named `SYSTEM.DEFAULT.LOCAL.QUEUE`.

These objects define the system defaults for the attributes of these IBM MQ objects:

- Local queues
- Model queues
- Alias queues
- Remote queues
- Processes
- Namelists
- Channels
- Storage classes
- Authentication information

Shared queues are a special type of local queue, so when you define a shared queue, the definition is based on the `SYSTEM.DEFAULT.LOCAL.QUEUE`. You need to remember to supply a value for the Coupling Facility structure name because one is not specified in the default definition. Alternatively, you could define your own default shared queue definition to use as a basis for shared queues so that they all inherit the required attributes. Remember that you need to define a shared queue on one queue manager in the queue-sharing group only.

System command objects

The names of the system command objects begin with the characters `SYSTEM.COMMAND`. You must define these objects before you can use the IBM MQ operations and control panels to issue commands to an IBM MQ subsystem.

There are two system command objects:

1. The system-command input queue is a local queue on which commands are put before they are processed by the IBM MQ command processor. It must be called `SYSTEM.COMMAND.INPUT`, although an alias of `SYSTEM.ADMIN.COMMAND.QUEUE` for it can be defined, for compatibility with non-z/OS IBM MQ.
2. `SYSTEM.COMMAND.REPLY.MODEL` is a model queue that defines the system-command reply-to queue.

There are two extra objects for use by the IBM MQ Explorer:

- SYSTEM.MQEXPLORER.REPLY.MODEL queue
- SYSTEM.ADMIN.SVRCONN channel

Commands are normally sent using nonpersistent messages so both the system command objects should have the DEFPSIST(NO) attribute so that applications using them (including the supplied applications like the utility program and the operations and control panels) get nonpersistent messages by default. If you have an application that uses persistent messages for commands, set the DEFTYPE(PERMDYN) attribute for the reply-to queue, because the reply messages to such commands are persistent.

System administration objects

The names of the system administration objects begin with the characters SYSTEM.ADMIN.

There are seven system administration objects:

- The SYSTEM.ADMIN.CHANNEL.EVENT queue
- The SYSTEM.ADMIN.COMMAND.EVENT queue
- The SYSTEM.ADMIN.CONFIG.EVENT queue
- The SYSTEM.ADMIN.PERFM.EVENT queue
- The SYSTEM.ADMIN.QMGR.EVENT queue
- The SYSTEM.ADMIN.TRACE.ROUTE.QUEUE queue
- The SYSTEM.ADMIN.ACTIVITY.QUEUE queue

Channels queues

To use distributed queuing, you need to define the following objects:

- A local queue with the name SYSTEM.CHANNEL.SYNCQ, which is used to maintain sequence numbers and logical units of work identifiers (LUWID) of channels. To improve channel performance, you should define this queue with an index type of MSGID (as shown in the supplied sample queue definition).
- A local queue with the name SYSTEM.CHANNEL.INITQ, which is used for channel commands.

You cannot define these queues as shared queues.

Cluster queues

To use IBM MQ clusters, you need to define the following objects:

- A local queue called the SYSTEM.CLUSTER.COMMAND.QUEUE, which is used to communicate repository changes between queue managers. Messages written to this queue contain updates to the repository data to be applied to the local copy of the repository, or requests for repository data.
- A local queue called SYSTEM.CLUSTER.REPOSITORY.QUEUE, which is used to hold a persistent copy of the repository.
- A local queue called SYSTEM.CLUSTER.TRANSMIT.QUEUE, which is the transmission queue for all destinations in the cluster. For performance reasons, you should define this queue with an index type of CORRELID (as shown in the sample queue definition).

These queues typically contain large numbers of messages.

You cannot define these queues as shared queues.

Queue-sharing group queues

To use shared channels and intra-group queuing, you need to define the following objects:

- A shared queue with the name `SYSTEM.QSG.CHANNEL.SYNCQ`, which is used to hold synchronization information for shared channels.
- A shared queue with the name `SYSTEM.QSG.TRANSMIT.QUEUE`, which is used as the transmission queue for intra-group queuing. If you are running in a queue-sharing group, you must define this queue, even if you are not using intra-group queuing.

Storage classes

You are recommended to define the following six storage classes. You must define four of them because they are required by IBM MQ. The other storage class definitions are recommended because they are used in the sample queue definitions.

DEFAULT (required)

This storage class is used for all message queues that are not performance critical and that don't fit in to any of the other storage classes. It is also the supplied default storage class if you do not specify one when defining a queue.

NODEFINE (required)

This storage class is used if the storage class specified when you define a queue is not defined.

REMOTE (required)

This storage class is used primarily for transmission queues, that is, system related queues with short-lived performance-critical messages.

SYSLNGLV

This storage class is used for long-lived, performance-critical messages.

SYSTEM (required)

This storage class is used for performance critical, system related message queues, for example the `SYSTEM.CHANNEL.SYNQ` and the `SYSTEM.CLUSTER.*` queues.

SYSVOLAT

This storage class is used for short-lived, performance-critical messages.

You can modify their attributes and add other storage class definitions as required.

Defining the system object dead-letter queue

The dead-letter queue is used if the message destination is not valid. IBM MQ puts such messages on a local queue called the dead-letter queue. Although having a dead-letter queue is not mandatory, you should regard it as essential, especially if you are using distributed queuing or one of the IBM MQ bridges.

Do **not** define the dead-letter queue as a shared queue. A put to a local queue on one queue manager might get put to the dead letter queue. If the dead letter queue was a shared queue, a dead letter queue handler on a different system could process the message and put it on a queue with the same name, but because this is on a different queue manager, it would be the wrong queue, or have a different security profile. If the queue did not exist, it would fail to reprocess it.

If you decide to define a dead-letter queue, you must also tell the queue manager its name. To do this use the `ALTER QMGR DEADQ(queue-name)` command. For more information see [Altering queue manager attributes](#).

Default transmission queue

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

If you decide to define a default transmission queue, you must also tell the queue manager its name. To do this use the ALTER QMGR command.

Internal queues

- **Pending data queue**
 - A queue defined for internal use, SYSTEM.PENDING.DATA.QUEUE, supports the use of durable subscriptions in a JMS publish/subscribe environment.
- **JMS 2.0 delivery delay staging queue**
 - If the delivery delay functionality provided by JMS 2.0 is used then an internal staging queue, SYSTEM.DDELAY.LOCAL.QUEUE, must be defined. This queue is used by the queue manager to temporarily store messages sent with a non-zero delivery delay until the delivery delay is completed, and the message is put to its target destination. A sample definition for this queue is provided, commented out, in CSQ4INSG.
 - When you define the SYSTEM.DDELAY.LOCAL.QUEUE queue, you must set the STGCLASS, MAXMSGL and MAXDEPTH attributes for the anticipated number of messages that will be sent with a delivery delay. Additionally when defining the SYSTEM.DDELAY.LOCAL.QUEUE queue ensure that only the queue manager can put messages to this queue. Care should be taken to ensure that no user identifier has the authority to put messages to this queue.

Channel authentication queue

For internal use of channel authentication the SYSTEM.CHLAUTH.DATA.QUEUE queue is required. Sample definitions are supplied with IBM MQ to help you define these objects. This sample is described in CSQ4INSA, which also defines some default rules.

Tuning your queue manager on IBM MQ for z/OS



There are few simple steps that you can take to ensure that your queue manager is tuned to avoid basic performance problems.

There are a number of ways in which you can improve the performance of your queue manager, which are controlled by queue manager attributes set by the ALTER QMGR command. This section contains information about how you can do this by setting the maximum number of messages allowed on the queue manager, or by performing 'housekeeping' on the queue manager. IBM MQ SupportPac MP16 - WebSphere MQ for z/OS Capacity planning & tuning gives more information on performance and tuning.

Syncpoints

One of the roles of the queue manager is syncpoint control within an application. An application constructs a unit of work containing any number of MQPUT or MQGET calls terminated with an MQCMIT call. However, as the number of MQPUT or MQGET calls within the scope of one MQCMIT increases, the performance cost of the commit increases significantly.

You can limit the number of messages within any single syncpoint by using the MAXUMSGS queue manager attribute. You should not normally exceed 100 MQPUTs within a single MQCMIT.

Set a fairly low MAXUMSGS value, such as 100, to avoid performance problems, and to protect against looping applications. If you have a need for a larger value, change MAXUMSGS temporarily while the application that requires the larger value is run, rather than using a permanently high value. However, be aware that reducing the value of MAXUMSGS can cause problems to existing applications and queue manager processes such as clustering if they are already using a higher value.

When publishing messages out of syncpoint to topics it is possible to receive this reason code. See “Publishers and publications” on page 204.

MAXUMSGS has no effect on MQ Telemetry. MQ Telemetry tries to batch requests to subscribe, unsubscribe, send, and receive messages from multiple clients into batches of work within a transaction.

Expired messages

Messages that have expired are discarded by the next appropriate MQGET call. However, if no such call occurs, the expired messages are not discarded, and, for some queues, particularly those where message retrieval is done by MessageId, CorrelId, or GroupId and the queue is indexed for performance, many expired messages can accumulate. The queue manager can periodically scan any queue for expired messages, which are then deleted. You can choose how often this scanning takes place, if at all. There are two ways of doing this:

Explicit request

You can control which queues are scanned and when. Issue the REFRESH QMGR TYPE(EXPIRY) command, specifying the queue or queues that you want to be scanned.

Periodic scan

You can specify an expiry interval in the queue manager object by using the EXPRYINT attribute. The queue manager maintains information about the expired messages on each queue, and knows at what time a scan for expired messages is worthwhile. Each time that the EXPRYINT interval is reached, the queue manager looks for candidate queues that are worth scanning for expired messages, and scans only those queues that it deems to be worthwhile. It does not scan all queues. This avoids any processor time being wasted on unnecessary scans.

Shared queues are only scanned by one queue manager in the queue-sharing group. Generally, the first queue manager to restart or the first to have the EXPRYINT set performs the scan.

Note: You must set the same EXPRYINT value for all queue managers within a queue-sharing group.


Sample definitions supplied with for IBM MQ for z/OS



Use this topic as a reference for the sample JCL, and code supplied with IBM MQ for z/OS.

The following sample definitions are supplied with IBM MQ in the thlqual.SCSQPROC library. You can use them to define the system objects and to customize your own objects. You can include some of them in the initialization input data sets (described in Initialization commands).

Table 26. IBM MQ sample definitions for system objects

Initialization input data set	Sample name
CSQINP1	CSQ4INP1 CSQ4INPR
CSQINP2	CSQ4INSA CSQ4INYS ¹ CSQ4INSX CSQ4INSG CSQ4INSR CSQ4INSS CSQ4INSJ CSQ4INSM CSQ4INYG CSQ4INYR CSQ4INYC CSQ4INYD  CSQ4INSC
CSQINPT	CSQ4INST CSQ4INYT
Other	CSQ4DISP CSQ4INPX CSQ4IVPQ CSQ4IVPG

Note:

1. The order of these sample definitions is important: an error occurs if INYS, INSX, and INSG are ordered incorrectly.

CSQINP1 samples

Use the sample CSQINP1 data set thlqual.SCSQPROC(CSQ4INP1) when you are using one page set for each class of message, or thlqual.SCSQPROC(CSQ4INPR) when using multiple page sets for the major classes of message. It contains definitions of buffer pools, page set to buffer pool associations, and an ALTER SECURITY command. Include the sample in the CSQINP1 concatenation of your queue manager started task procedure.

CSQINP2 samples

CSQ4INSG system object sample

The sample CSQINP2 data set thlqual.SCSQPROC(CSQ4INSG) contains definitions for the following system objects for general use:

- System default objects
- System command objects
- System administration objects
- Other objects for system use

You must define the objects in this sample, but you need to do it only once when the subsystem is first started. Including the definitions in the CSQINP2 data set is the best way to do this. They are maintained across queue manager shutdown and restart. You must not change the object names, but you can change their attributes if required.

When the following conditions are met, one message is put to the SYSTEM.DURABLE.SUBSCRIBER.QUEUE queue (even if publish subscribe is not active):

- The IBM MQ installation is Version 7.0.0 or later

- The QMGR attribute PSMODE is set to DISABLED
- The sample object CSQ4INST statement DEFINE SUB('SYSTEM.DEFAULT.SUB') is present.

To avoid this, delete or comment out the DEFINE SUB('SYSTEM.DEFAULT.SUB') statement.

The JMS 2.0 delivery delay staging queue, SYSTEM.DDELAY.LOCAL.QUEUE only need be defined if JMS 2.0 delivery delay is used. By default, the queue definition is commented out, which you can uncomment if required.

CSQ4INSA system object and authentication sample

The sample CSQINP2 data set thlqual.SCSQPROC(CSQ4INSA) contains the channel authentication system queue definition. This queue holds the channel authentication records. It also contains the default channel authentication rules.

You must define the objects in this sample if CHLAUTH is ENABLED on the queue manager and you want to run channels, or you want to SET or DISPLAY CHLAUTH record. You only need to define them once when the subsystem is first started. Including the definitions in the CSQINP2 data set is the best way to do this. They are maintained across a queue manager shutdown and restart, you must not change the queue name.

V 9.0.3 CSQ4INSC system object sample

The sample CSQINP2 data set thlqual.SCSQPROC(CSQ4INSC) contains the IBM Cloud (formerly Bluemix) registration system queue definition. This queue holds the registration and usage messages for the IBM Cloud Product Insights service on IBM Cloud (formerly Bluemix). You must define the object in this sample if you want to register, and upload usage data for, your IBM MQ Advanced for z/OS, Value Unit Edition queue manager with an instance of the IBM Cloud Product Insights service. You only need to define the queue once when the subsystem is first started. Including the definitions in the CSQINP2 data set is the best way to do this. The definition is maintained across a queue manager shutdown and restart; you must not change the queue name.

CSQ4INSS system object sample

You can define additional system objects if you are using queue-sharing groups.

Sample data set thlqual.SCSQPROC(CSQ4INSS) contains sample commands for use with CF structures and a set of definitions for the system objects required for shared channels and intra-group queuing.

You cannot use this sample as is; you must customize it before use. Then you can include this member in the CSQINP2 DD concatenation of the queue manager startup procedure, or you can use it as input to the COMMAND function of the CSQUTIL utility to issue the required commands.

When you are defining group or shared objects, you need to include them in the CSQINP2 DD concatenation for only one queue manager in the queue-sharing group.

CSQ4INSX system object sample

You must define additional system objects if you are using distributed queuing and clustering.

Sample data set thlqual.SCSQPROC(CSQ4INSX) contains the queue definitions required. You can include this member in the CSQINP2 DD concatenation of the queue manager startup procedure, or you can use it as input to the COMMAND function in CSQUTIL utility to issue the required DEFINE commands.

There are two types of object definitions:

- SYSTEM.CHANNEL.xx, needed for any distributed queuing
- SYSTEM.CLUSTER.xx, needed for clustering

CSQ4INSJ system JMS object sample

Defines queues used in the JMS publish/subscribe domain.

CSQ4INSM system object sample

If you are using advanced message security you must define additional system objects. Sample data set thlqual.SCSQPROC(CSQ4INSM) contains the queue definitions required.

CSQ4INSR object sample

Defines queues used by WebSphere Application Server and brokers.

CSQ4INYD object sample

If you are using distributed queuing and you need to set up your own queues, processes, and channels.

Sample data set thlqual.SCSQPROC(CSQ4INYD) contains sample definitions that you can use for customizing your distributed queuing objects. It comprises:

- A set of definitions for the sending end
- A set of definitions for the receiving end
- A set of definitions for using clients

You cannot use this sample as is - you must customize it before use. Then you can include this member in the CSQINP2 DD concatenation of the queue manager startup procedure, or you can use it as input to the COMMAND function of the CSQUTIL utility to issue the required DEFINE commands. (This is preferable because it means that you don't have to redefine these objects each time you restart the queue manager).

CSQ4INYC object sample

If you are using clustering, definitions equivalent to the channel definitions and remote queue definitions of distributed queuing are created automatically, when needed. However, some manual channel definitions are needed - a cluster-receiver channel for the cluster and a cluster-sender definition to at least one cluster repository queue manager.

The sample data set: thlqual.SCSQPROC(CSQ4INYC) contains the following sample definitions that you can use for customizing your clustering objects:

- Definitions for the queue manager
- Definitions for the receiving channel
- Definitions for the sending channel
- Definitions for cluster queues
- Definitions for lists of clusters

You cannot use this sample as is - you must customize it before use. Then you can include this member in the CSQINP2 DD concatenation of the queue manager startup procedure, or you can use it as input to the COMMAND function of the CSQUTIL utility to issue the required DEFINE commands. This is preferable because it means that you don't have to redefine these objects each time that you restart IBM MQ.

CSQ4INYG object sample

The sample data set: thlqual.SCSQPROC(CSQ4INYG) contains the following sample definitions that you can use for customizing your own objects for general use:

- Dead-letter queue
- Default transmission queue
- CICS adapter objects

You cannot use this sample as is - you must customize it before use. Then you can include this member in the CSQINP2 DD concatenation of the queue manager startup procedure, or you can

use it as input to the COMMAND function of the CSQUTIL utility to issue the required DEFINE commands. This is preferable because it means that you don't have to redefine these objects each time that you restart IBM MQ.

In addition to the sample definitions here, you can use the system object definitions as the basis for your own resource definitions. For example, you can make a working copy of SYSTEM.DEFAULT.LOCAL.QUEUE and name it MY.DEFAULT.LOCAL.QUEUE. You can then change any of the parameters in this copy as required. You could then issue a DEFINE command by whichever method you choose, provided you have the authority to create resources of that type.

Default transmission queue

Read the Default transmission queue description before you decide whether you want to define a default transmission queue.

- If you decide that you do want to define a default transmission queue, remember that you must also define a channel to serve it.
- If you decide that you do not want to define one, remember to remove the DEFXMITQ statement from the ALTER QMGR command in the sample.

CICS adapter objects

The sample defines an initiation queue named CICS01.INITQ. This queue is used by the IBM MQ -supplied CKTI transaction. You can change the name of this queue; however it must match the name specified in the CICS system initialization table (SIT) or SYSIN override in the INITPARM statement.

CSQ4INYS/CSQ4INYR object samples

Storage class definitions for using:

- one page set for each class of message
- multiple page sets for major classes of message

CSQINPT samples

CSQ4INST

The sample data set: thlqual.SCSQPROC(CSQ4INST) contains the definition for the system default subscription.

The default system subscription, SYSTEM.DEFAULT.SUB, moved from CSQ4INSG to CSQ4INST in IBM WebSphere MQ Version 7.1.

You must define the object in this sample, but you need to do it only once when the publish/subscribe engine is first started. Including the definition in the CSQINPT data set is the best way to do this. It is maintained across queue manager shutdown and restart. You must not change the object name, but you can change their attributes if required.

CSQ4INYT

The sample data set: thlqual.SCSQPROC(CSQ4INYT) contains a set of commands that you might want to run when the publish/subscribe engine is started. This sample displays Topic and Subscription information.

Other

CSQ4DISP display sample

The sample data set: thlqual.SCSQPROC(CSQ4DISP) contains a set of generic DISPLAY commands that display all the defined resources on your queue manager. This includes the definitions for all IBM MQ objects and definitions such as storage classes and trace. These commands can generate a large amount of output. You can use this sample in the CSQINP2 data set or as input to the COMMAND function of the CSQUTIL utility.

CSQ4INPX sample

The sample data set: thlqual.SCSQPROC(CSQ4INPX) contains a set of commands that you might want to execute each time the channel initiator starts. You must customize this sample before use; you can then include it in the CSQINPX data set for the channel initiator.

CSQ4IVPQ and CSQ4IVPG samples

The sample data sets: thlqual.SCSQPROC(CSQ4IVPQ) and thlqual.SCSQPROC(CSQ4IVPG) contain sets of DEFINE commands that are required to run the installation verification programs (IVPs).

You can include these samples in the CSQINP2 data set. When you have run the IVPs successfully, you do not need to run them again each time the queue manager is restarted. Therefore, you do not need to keep these samples in the CSQINP2 concatenation permanently.

V 9.0.4

Other samples

CSQ4QMIN IBM Cloud (formerly Bluemix) registration stanza

The sample CSQMOMIN data set thlqual.SCSQPROC(CSQ4QMIN) contains the IBM Cloud (formerly Bluemix) registration stanza. The stanza defines the API key and host for an instance of the IBM Cloud Product Insights service on IBM Cloud (formerly Bluemix). You must define this stanza if you want to register, and upload usage data for, your IBM MQ Advanced for z/OS, Value Unit Edition queue manager with an instance of the IBM Cloud Product Insights service.

Recovery and restart on z/OS

z/OS

Use the links in this topic to find out about the features of IBM MQ for z/OS for restart and recovery.

IBM MQ for z/OS has robust features for restart and recovery. For information about how a queue manager recovers after it has stopped, and what happens when it is restarted, see the following subtopics:

- “How changes are made to data in IBM MQ for z/OS” on page 413
- “How consistency is maintained in IBM MQ for z/OS” on page 414
- “What happens during termination in IBM MQ for z/OS” on page 417
- “What happens during restart and recovery in IBM MQ for z/OS” on page 418
- “How in-doubt units of recovery are resolved” on page 420
- “Shared queue recovery” on page 422

Related information:

Planning for backup and recovery

z/OS IBM MQ for z/OS recovery actions

z/OS Administering z/OS

z/OS Messages for IBM MQ for z/OS

MQSC reference

How changes are made to data in IBM MQ for z/OS

z/OS

IBM MQ for z/OS must interact with other subsystems to keep all the data consistent. This topic contains information about *units of recovery*, what they are and how they are used in *back outs*.

Units of recovery

A *unit of recovery* is the processing done by a single queue manager for an application program, that changes IBM MQ data from one point of consistency to another. A *point of consistency* - also called a *syncpoint* or *commit point* - is a point in time when all the recoverable data that an application program accesses is consistent.

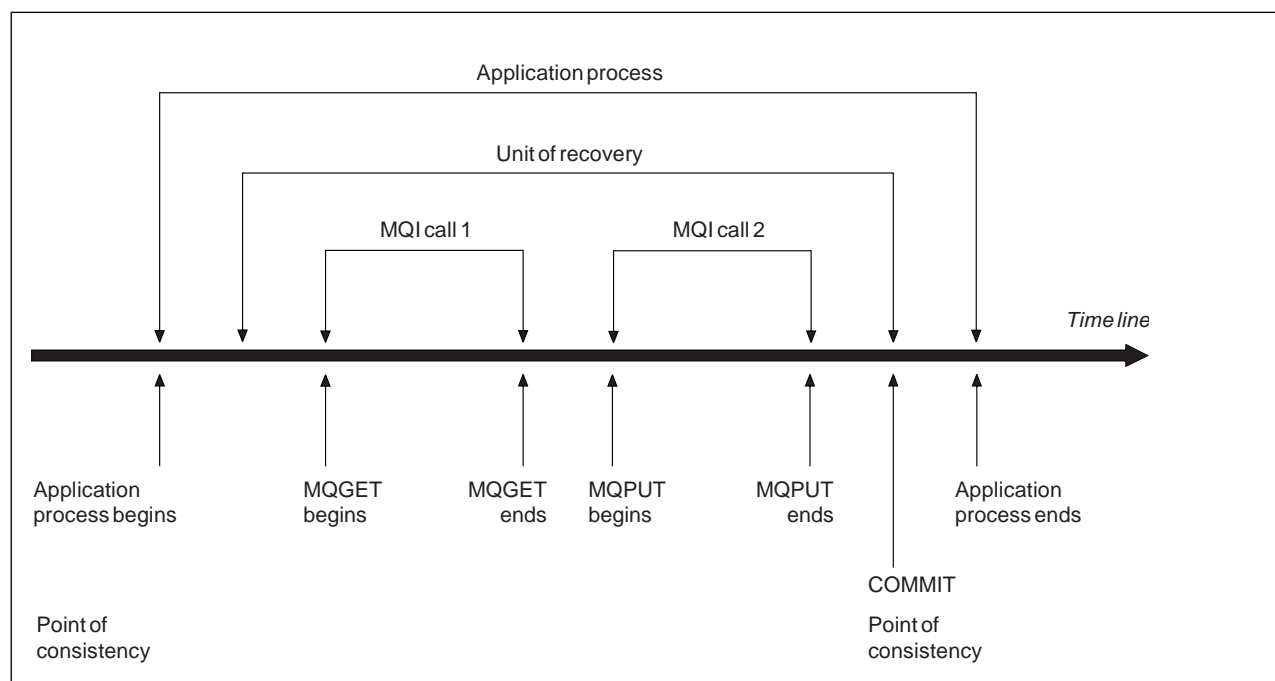


Figure 75. A unit of recovery within an application program. Typically, the unit of recovery consists of more than one MQI call. More than one unit of recovery can occur within an application program.

A unit of recovery begins with the first change to the data after the beginning of the program or following the previous point of consistency; it ends with a later point of consistency. Figure 75 shows the relationship between units of recovery, the point of consistency, and an application program. In this example, the application program makes changes to queues through MQI calls 1 and 2. The application program can include more than one unit of recovery or just one. However, any complete unit of recovery ends in a commit point.

For example, a bank transaction transfers funds from one account to another. First, the program subtracts the amount from the first account, account A. Then, it adds the amount to the second account, B. After subtracting the amount from A, the two accounts are inconsistent and IBM MQ cannot commit. They

become consistent when the amount is added to account B. When both steps are complete, the program can announce a point of consistency through a commit, making the changes visible to other application programs.

Normal termination of an application program automatically causes a point of consistency. Some program requests in CICS and IMS programs also cause a point of consistency, for example, EXEC CICS SYNCPOINT.

Backing out work

If an error occurs within a unit of recovery, IBM MQ removes any changes to data, returning the data to its state at the start of the unit of recovery; that is, IBM MQ backs out the work. The events are shown in Figure 76.

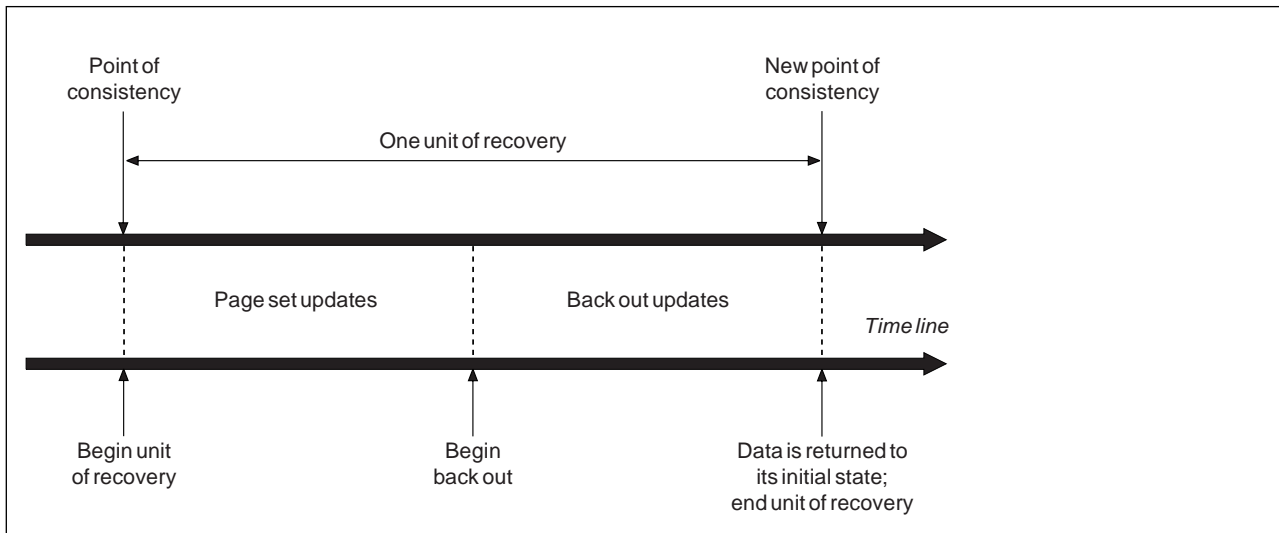


Figure 76. A unit of recovery showing back out

How consistency is maintained in IBM MQ for z/OS

▶ z/OS

Data in IBM MQ for z/OS must be consistent with batch, CICS, IMS, or TSO. Any data changed in one must be matched by a change in the other.

Before one system commits the changed data, it must know that the other system can make the corresponding change. So, the systems must communicate.

During a *two-phase commit* (for example under CICS), one subsystem coordinates the process. That subsystem is called the *coordinator*; the other is the *participant*. CICS or IMS is always the coordinator in interactions with IBM MQ, and IBM MQ is always the participant. In the batch or TSO environment, IBM MQ can participate in two-phase commit protocols coordinated by z/OS RRS.

During a *single-phase commit* (for example under TSO or batch), IBM MQ is always the coordinator in the interactions and completely controls the commit process.

In a WebSphere Application Server environment, the semantics of the JMS session object determine whether single-phase or two-phase commit coordination is used.

Consistency with CICS or IMS

The connection between IBM MQ and CICS or IMS supports the following syncpoint protocols:

- Two-phase commit - for transactions that update resources owned by more than one resource manager. This is the standard distributed syncpoint protocol. It involves more logging and message flows than a single-phase commit.
- Single-phase commit - for transactions that update resources owned by a single resource manager (IBM MQ). This protocol is optimized for logging and message flows.
- Bypass of syncpoint - for transactions that involve IBM MQ but which do nothing in the queue manager that requires a syncpoint (for example, browsing a queue).

In each case, CICS or IMS acts as the syncpoint manager.

The stages of the two-phase commit that IBM MQ uses to communicate with CICS or IMS are as follows:

1. In phase 1, each system determines independently whether it has recorded enough recovery information in its log, and can commit its work.
At the end of the phase, the systems communicate. If they agree, each begins the next phase.
2. In phase 2, the changes are made permanent. If one of the systems abends during phase 2, the operation is completed by the recovery process during restart.

Illustration of the two-phase commit process

Figure 77 illustrates the two-phase commit process. Events in the CICS or IMS coordinator are shown on the upper line, events in IBM MQ on the lower line.

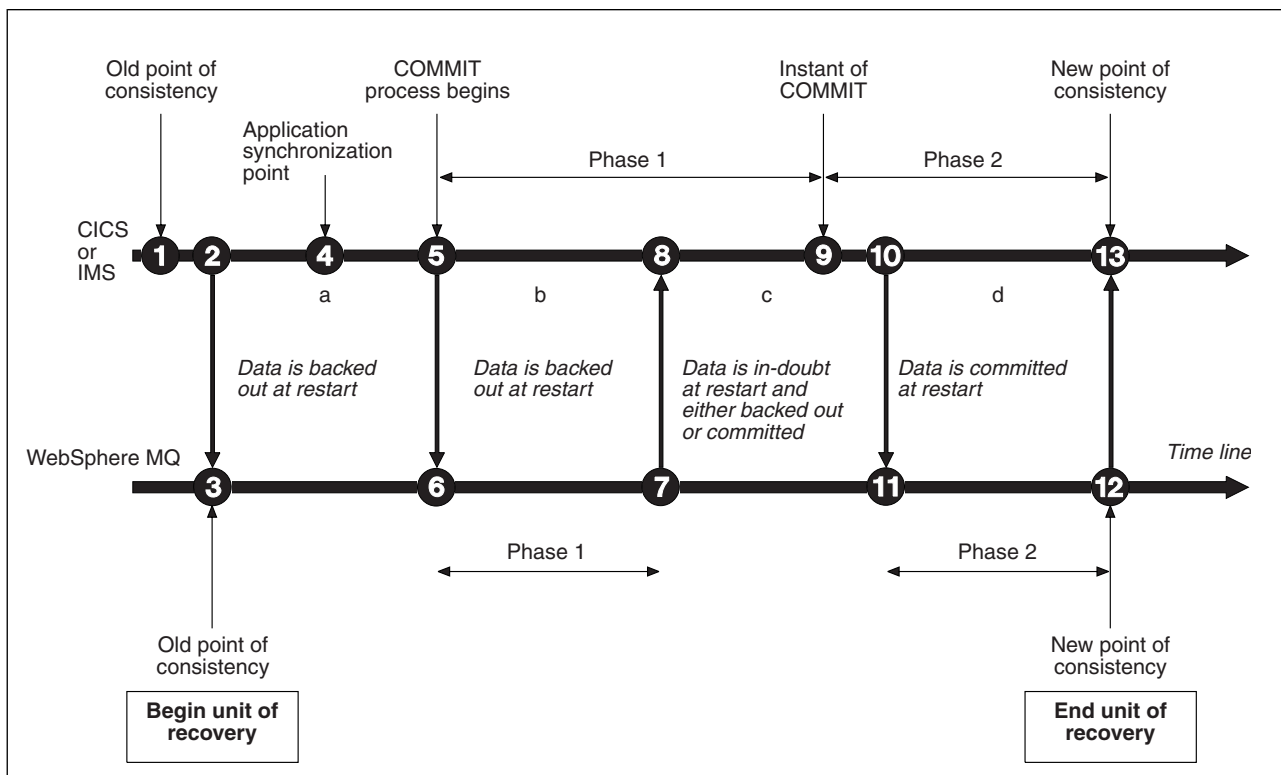


Figure 77. The two-phase commit process

The numbers in the following section are linked to those shown in the figure.

1. The data in the coordinator is at a point of consistency.

2. An application program in the coordinator calls IBM MQ to update a queue by adding a message.
3. This starts a unit of recovery in IBM MQ.
4. Processing continues in the coordinator until an application synchronization point is reached.
5. The coordinator then starts commit processing. CICS programs use a SYNCPOINT command or a normal application termination to start the commit. IMS programs can start the commit by using a CHKP call, a SYNC call, a GET UNIQUE call to the IOPCB, or a normal application termination. Phase 1 of commit processing begins.
6. As the coordinator begins phase 1 processing, so does IBM MQ.
7. IBM MQ successfully completes phase 1, writes this fact in its log, and notifies the coordinator.
8. The coordinator receives the notification.
9. The coordinator successfully completes its phase 1 processing. Now both subsystems agree to commit the data changes, because both have completed phase 1 and could recover from any errors. The coordinator records in its log the instant of commit - the irrevocable decision of the two subsystems to make the changes.
The coordinator now begins phase 2 of the processing - the actual commitment.
10. The coordinator notifies IBM MQ to begin its phase 2.
11. IBM MQ logs the start of phase 2.
12. Phase 2 is successfully completed, and this is now a new point of consistency for IBM MQ. IBM MQ then notifies the coordinator that it has finished its phase 2 processing.
13. The coordinator finishes its phase 2 processing. The data controlled by both subsystems is now consistent and available to other applications.

How consistency is maintained after an abnormal termination

When a queue manager is restarted after an abnormal termination, it must determine whether to commit or to back out units of recovery that were active at the time of termination. For some units of recovery, IBM MQ has enough information to make the decision. For others, it does not, and must get information from the coordinator when the connection is reestablished.

Figure 77 on page 415 shows four periods within the two phases: a, b, c, and d. The status of a unit of recovery depends on the period in which termination happened. The status can be one of the following:

In flight

The queue manager ended before finishing phase 1 (period a or b); during restart, IBM MQ backs out the updates.

In doubt

The queue manager ended after finishing phase 1 and before starting phase 2 (period c); only the coordinator knows whether the error happened before or after the commit (point 9). If it happened before, IBM MQ must back out its changes; if it happened after, IBM MQ must make its changes and commit them. At restart, IBM MQ waits for information from the coordinator before processing this unit of recovery.

In commit

The queue manager ended after it began its own phase 2 processing (period d); it makes committed changes.

In backout

The queue manager ended after a unit of recovery began to be backed out but before the process was complete (not shown in the figure) during restart, IBM MQ continues to back out the changes.

What happens during termination in IBM MQ for z/OS

z/OS

A queue manager terminates normally in response to the STOP QMGR command. If a queue manager stops for any other reason, the termination is abnormal.

Note, that during queue manager termination, IBM MQ internally issues the command `DISPLAY CONN(*) TYPE(CONN) ALL WHERE (APPLTYPE NE SYSTEMAL)`

so that you are aware of what threads might prevent the queue manager from completing shutdown.

SYSTEMAL matches APPLTYPES of either SYSTEM or CHINIT, so the DISPLAY CONN command filtering application types not matching SYSTEMAL, returns to the joblog information about threads that could be preventing normal shutdown.

Normal termination

In a normal termination, IBM MQ stops all activity in an orderly way. You can stop IBM MQ using either quiesce, force, or restart mode. The effects are given in Table 27.

Table 27. Termination using QUIESCE, FORCE, and RESTART

Thread type	QUIESCE	FORCE	RESTART
Active threads	Run to completion	Back out	Back out
New threads	Can start	Not permitted	Not permitted
New connections	Not permitted	Not permitted	Not permitted

Batch applications are notified if a termination occurs while the application is still connected.

With CICS, a current thread runs only to the end of the unit of recovery. With CICS, stopping a queue manager in quiesce mode stops the CICS adapter, and so if an active task contains more than one unit of recovery, the task does not necessarily run to completion.

If you stop a queue manager in force or restart mode, no new threads are allocated, and work on connected threads is rolled back. Using these modes can create in-doubt units of recovery for threads that are between commit processing phases. They are resolved when IBM MQ is reconnected with the controlling CICS, IMS, or RRS subsystem.

When you stop a queue manager, in any mode, the steps are:

1. Connections are ended.
2. IBM MQ ceases to accept commands.
3. IBM MQ ensures that any outstanding updates to the page sets are completed.
4. The DISPLAY USAGE command is issued internally by IBM MQ so that the restart RBA is recorded on the z/OS console log.
5. The shutdown checkpoint is taken and the BSDS is updated.

Terminations that specify quiesce mode do not affect in-doubt units of recovery. Any unit that is in doubt remains in doubt.

Abnormal termination

An abnormal termination can leave data in an inconsistent state, for example:

- A unit of recovery has been interrupted before reaching a point of consistency.
- Committed data has not been written to page sets.
- Uncommitted data has been written to page sets.
- An application program has been interrupted between phase 1 and phase 2 of the commit process, leaving the unit of recovery in doubt.

IBM MQ resolves any data inconsistencies arising from abnormal termination during restart and recovery.

What happens during restart and recovery in IBM MQ for z/OS

z/OS

IBM MQ uses its recovery log and the bootstrap data set (BSDS) to determine what to recover when it restarts. The BSDS identifies the active and archive log data sets, and the location of the most recent IBM MQ checkpoint in the log.

Introduction to restart and recovery

After IBM MQ has been initialized, the queue manager restart process takes place as follows:

- Log initialization
- Current status rebuild
- Forward log recovery
- Backward log recovery
- Queue index rebuilding

When recovery has been completed:

- Committed changes are reflected in the data.
- In-doubt activity is reflected in the data. However, the data is locked and cannot be used until IBM MQ recognizes and acts on the in-doubt decision.
- Interrupted in-flight and in-abort changes have been removed from the queues. The messages are consistent and can be used.
- A new checkpoint has been taken.
- New indexes have been built for indexed queues containing persistent messages (described in “Rebuilding queue indexes” on page 419).

If dual BSDSs are in use, IBM MQ checks the consistency of the time stamps in the BSDS:

- If both copies of the BSDS are current, IBM MQ tests whether the two time stamps are equal. If they are not, IBM MQ issues message CSQJ120E and terminates. This can happen when the two copies of the BSDS are maintained on separate DASD volumes and one of the volumes was restored while the queue manager was stopped. IBM MQ detects the situation at restart.
- If one copy of the BSDS was de-allocated, and logging continued with a single BSDS, a problem could arise. If both copies of the BSDS are maintained on a single volume, and the volume was restored, or if both BSDS copies were restored separately, IBM MQ might not detect the restoration. In that case, log records not noted in the BSDS would be unknown to the system.

Batch applications are not notified when restart occurs *after* the application has requested a connection.

Understanding the log range required for recovery

During restart, the range of log data which must be read is dependent on many factors:

- At the time of an abnormal termination, there are typically many incomplete units of work in the system. As described earlier, restart processing will bring the system to a state of consistency, which may involve backing out inflight units of work, or recovering locks on indoubt units of work. Unit of work recovery requires that all unit of work log records for inflight, in-backout, and in-doubt units of work are available. IBM MQ will 'shunt' old units of work, so that unit of work recovery can be performed using a much smaller range of log data.
- At the time of an abnormal termination, there are typically many persistent updates which are only held in the buffer pool cache. They have not yet been written to disk. These changes must be read

from the log, and reapplied to the data held in page sets. Page set recovery RBAs in the checkpoint describe the lowest log RBA required for updating the page sets to a consistent state.

- If old page sets have been introduced into the system, for example, a page set backup has been introduced to recover from a media failure, all the changes must be read from the log from the time the backup was taken. These changes are reapplied to the data held in the page set being recovered. Page set recovery RBAs held in page 0 of the page set describe the lowest log RBA required for media recovery of a page set.
- If using persistent messages on shared queues, a range of log data is required to recover CFSTRUCTs which are holding persistent messages. The earliest log data that would be required to perform a CFSTRUCT recovery, is from around the time of the old CFSTRUCT BACKUP.

During normal running, the DISPLAY USAGE TYPE(DATASET) command can be used to view the recovery log range associated with these factors (it is unable to provide information due to reintroducing old page sets, of course). To avoid any issues that might prolong a queue manager restart in the event of an abnormal termination, regularly monitor the values output from DISPLAY USAGE TYPE(DATASET).

In addition, the queue manager issues informational messages relating to these factors:

- CSQJ160I and CSQJ161I warn of long running units of work.
- CSQR026I and CSQR027I provide information about whether these long running units of work have been successfully shunted.
- CSQE040I and CSQE041E warn that structure backups are getting old, and consequently a RECOVER CFSTRUCT operation would take a long time.

Determining which application has a long running unit of work

It is possible to determine the application with the long-running unit of work. To do this, you use the DISPLAY CONN command.

The DISPLAY CONN command returns connection information for all the applications connected to the queue manager, together with additional information that helps you determine which application(s) currently have a long-running unit of work. The information returned by the DISPLAY CONN command is similar to the information returned by the DISPLAY QSTATUS command, but the main difference is that DISPLAY CONN displays information about objects, and transactional information for a particular connection, rather than details of which connections are associated with a particular object.

For each connected application, the DISPLAY CONN command returns the following information:

- Basic information including the Connection Id and PID.
- Transactional information for that connection, including the time and date when the transaction was created (that is, when the first MQGET/PUT was made under syncpoint), and when the transaction first wrote to the log.
- Log time information indicating which application still has a long running unit of work.
- A list of all objects that the connection currently has open. Details for each object are returned as a separate message, with the Connection Id used as a key. Because there are different types of object such as queues and queue managers, the information displayed with the object is specific to its object type.

Rebuilding queue indexes

To increase the speed of MQGET operations on a queue where messages are not retrieved sequentially, you can specify that you want IBM MQ to maintain an index of the message or correlation identifiers or groupid for all the messages on that queue.

When a queue manager is restarted, these indexes are rebuilt for each queue. This applies only to persistent messages; nonpersistent messages are deleted at restart. If your indexed queues contain large numbers of persistent messages, this increases the time taken to restart the queue manager.

You can choose to have indexes rebuilt asynchronously to queue manager startup by using the QINDXBLD parameter of the CSQ6SYSP macro. If you set QINDXBLD=NOWAIT, IBM MQ restarts without waiting for indexes to rebuild.

How in-doubt units of recovery are resolved

z/OS

If IBM MQ loses its connection to another resource manager, it typically attempts to recover all inconsistent objects at restart.

If IBM MQ loses its connection to CICS, IMS, or RRS, it normally attempts to recover all inconsistent objects at restart. The information required to resolve in-doubt units of recovery must come from the coordinating system. The next sections describe the process of resolution for different environments.

- How in-doubt units of recovery are resolved from CICS
- How in-doubt units of recovery are resolved from IMS
- How in-doubt units of recovery are resolved from RRS
- How in-doubt units of recovery with a GROUP unit of recovery disposition are resolved

How in-doubt units of recovery are resolved from CICS

Under some circumstances, CICS cannot run the IBM MQ process to resolve in-doubt units of recovery. When this happens, IBM MQ sends one of the following messages:

- CSQC404E
- CSQC405E
- CSQC406E
- CSQC407E

followed by the message CSQC408I.

For details of what these messages mean, see the IBM MQ for z/OS messages, completion, and reason codes manual.

The resolution of in-doubt units does not effect CICS resources. CICS is in control of recovery coordination and, when it restarts, automatically commits or backs out each unit, depending on whether there was a log record marking the beginning of the commit. The existence of in-doubt objects does not lock CICS resources while IBM MQ is being reconnected.

One of the functions of the CICS adapter is to keep data synchronized between CICS and IBM MQ. If a queue manager abends while connected to CICS, it is possible for CICS to commit or back out work without IBM MQ being aware of it. When the queue manager restarts, that work is termed *in doubt*.

IBM MQ cannot resolve these in-doubt units of recovery (that is, commit or back out the changes made to IBM MQ resources) until the connection to CICS is restarted or reconnected.

A process to resolve in-doubt units of recovery is initiated during startup of the CICS adapter. The process starts when the adapter requests a list of in-doubt units of recovery. Then:

- The adapter receives a list of in-doubt units of recovery for this connection ID from IBM MQ, and passes them to CICS for resolution.
- CICS compares entries from this list with entries in its own log. CICS determines from its own list what action it took for each in-doubt unit of recovery.

For all resolved units, IBM MQ updates the queues as necessary and releases the corresponding locks. Unresolved units can remain after restart. Resolve them by the methods described in the Administering IBM MQ for z/OS.

How in-doubt units of recovery are resolved from IMS

Resolving in-doubt units of recovery in IMS does not effect DL/I resources. IMS is in control of recovery coordination and, when it restarts, automatically commits or backs out incomplete DL/I work. The decision to commit or back out for online regions (non-fast-path) is on the presence or absence of IMS log record types X'3730' and X'3801'. The existence of in-doubt units of recovery does not imply that DL/I records are locked until IBM MQ connects.

During queue manager restart, IBM MQ makes a list of in-doubt units of recovery. IMS builds its own list of residual recovery entries (RREs). The RREs are logged at IMS checkpoints until all entries are resolved.

During reconnection of an IMS region to IBM MQ, IMS indicates to IBM MQ whether to commit or back out units of work marked in IBM MQ as in doubt.

When in-doubt units are resolved:

1. If IBM MQ recognizes that it has marked an entry for commit and IMS has marked it to be backed out, IBM MQ issues message CSQQ010E. IBM MQ issues this message for all inconsistencies of this type between IBM MQ and IMS.
2. If IBM MQ has any remaining in-doubt units, the adapter issues message CSQQ008I.

For all resolved units, IBM MQ updates queues as necessary and releases the corresponding locks.

IBM MQ maintains locks on in-doubt work that was not resolved. This can cause a backlog in the system if important locks are being held. The connection remains active so you can resolve the IMS RREs. Recover the in-doubt threads by the methods described in the Administering IBM MQ for z/OS.

All in-doubt work should be resolved unless there are software or operating problems, such as with an IMS cold start. In-doubt resolution by the IMS control region takes place in two circumstances:

1. At the start of the connection to IBM MQ, during which resolution is done synchronously.
2. When a program abends, during which the resolution is done asynchronously.

How in-doubt units of recovery are resolved from RRS

One of the functions of the RRS adapter is to keep data synchronized between IBM MQ and other RRS-participating resource managers. If a failure occurs when IBM MQ has completed phase one of the commit and is waiting for a decision from RRS (the commit coordinator), the unit of recovery enters the in-doubt state.

When communication is reestablished between RRS and IBM MQ, RRS automatically commits or backs out each unit of recovery, depending on whether there was a log record marking the beginning of the commit. IBM MQ cannot resolve these in-doubt units of recovery (that is, commit or back out the changes made to IBM MQ resources) until the connection to RRS is reestablished.

Under some circumstances, RRS cannot resolve in-doubt units of recovery. When this happens, IBM MQ sends one of the following messages to the z/OS console:

- CSQ3011I
- CSQ3013I
- CSQ3014I
- CSQ3016I

For details of what these messages mean, see the IBM MQ for z/OS messages, completion, and reason codes manual.

For all resolved units of recovery, IBM MQ updates the queues as necessary and releases the corresponding locks. Unresolved units of recovery can remain after restart. Resolve them by the method described in the Administering IBM MQ for z/OS.

How in-doubt units of recovery with a GROUP unit of recovery disposition are resolved

In-doubt transactions that have a GROUP unit of recovery disposition can be resolved by the transaction coordinator by any queue manager in the queue-sharing group (QSG) where the GROUPUR queue manager attribute is enabled. Whenever a transaction coordinator reconnects it typically requests a list of any outstanding in-doubt transactions and then resolves them using information from its logs.

When a transaction coordinator, that has connected with a GROUP unit of recovery disposition, requests the list of in-doubt transactions, the list returned comprises all in-doubt transactions with a GROUP unit of recovery disposition that exist throughout the queue-sharing group. This list is not dependent on which queue manager those in-doubt transactions were started on. A queue manager processing such a request compiles the list by communicating with all other active queue managers in the queue-sharing group using the `SYSTEM.QSG.UR.RESOLUTION.QUEUE`. The queue manager then reads the logs of any inactive queue managers, from their last checkpoint, to identify any additional in-doubt transactions that they would have reported had they been active.

When a transaction coordinator requests the resolution of an in-doubt transaction, the queue manager to which it is connected identifies whether the transaction was originated on itself and if so resolves it in the same way as transactions with a QMGR unit of recovery disposition. If the transaction was originated on another active queue manager in the QSG, a request to complete the resolution is routed to that queue manager using the `SYSTEM.QSG.UR.RESOLUTION.QUEUE`. In the case where the transaction was originated on an inactive queue manager in the QSG, any shared-queue work is resolved immediately, and a request to resolve any remaining private queue work is placed on the `SYSTEM.QSG.UR.RESOLUTION.QUEUE`. The inactive queue manager processes this request upon start-up before accepting new work. In this scenario, the original queue manager's logs still reflect that the unit of recovery is in doubt until it has restarted and processed the request.

Shared queue recovery

z/OS

Use this topic to understand IBM MQ recovery and resilience of various components in the queue-sharing group environment.

- “Transactional recovery”
- “Peer recovery” on page 423
- “Shared queue definitions” on page 423
- “Logging” on page 423
- “Coupling facility and structure failures” on page 423
- “Structure failure scenarios” on page 424
- “Resilience to coupling facility connectivity failures” on page 426
- “Managing Resilience to coupling facility connectivity failures” on page 426
- “Operational behavior” on page 428

Transactional recovery

When an application issues a MQBACK call or terminates abnormally (for example, because of an EXEC CICS ROLLBACK or an IMS abend) thread-level information stored in the queue manager ensures that the in-flight unit of work is rolled back. MQPUT and MQGET operations within syncpoint on shared queues are rolled back in the same way as updates to non-shared queues.

Peer recovery

If a queue manager fails, it disconnects abnormally from the coupling facility structures that it is currently connected to. If the connection between the z/OS instance and the coupling facility fails (for example, physical link failure or power-off of a coupling facility or partition) this is also detected as an abnormal termination of the connection between the queue manager and the coupling facility structures involved. Other queue managers in the same queue-sharing group that remain connected to that structure detect the abnormal disconnection and all attempt to initiate *peer recovery* for the failed queue manager on that structure. Only one of these queue managers initiates *peer recovery* successfully, but all the other queue managers cooperate in the recovery of units of work that were owned by the queue manager that failed.

If a queue manager fails when there are no peers connected to a structure, recovery is performed when another queue manager connects to that structure, or when the queue manager that failed restarts.

Peer recovery, often referred to as Peer Level Recovery (PLR), is performed on a structure by structure basis and it is possible for a single queue manager to participate in the recovery of more than one structure at the same time. However, the set of peers cooperating in the recovery of different structures might vary depending on which queue managers were connected to the different structures at the time of failure.

When the failed queue manager restarts, it reconnects to the structures that it was connected to at the time of failure, and recovers any remaining unresolved units of work that were not recovered by peer recovery.

Peer recovery is a multi-phase process. During the first phase, units of work that had progressed beyond the in-flight phase are recovered; this might involve committing messages for units of work that are in-commit and locking messages for units of work that are in-doubt. During the second phase, queues that had threads active against them in the failing queue manager are checked, uncommitted messages related to in-flight units of work are rolled back, and information about active handles on shared queues in the failed queue manager are reset. This means that IBM MQ resets any indicators that the failing queue manager had a shared queue open for input-exclusive, allowing other active queue managers to open the queue for input.

Shared queue definitions

The queue objects that represent the attributes of a shared queue are held in the shared Db2 repository used by the queue-sharing group. Ensure that adequate procedures are in place for the backup and recovery of the Db2 tables used to hold IBM MQ objects. You can also use the IBM MQ CSQUTIL utility to create MQSC commands for replay into a queue manager to redefine IBM MQ objects, including shared queue and group definitions stored in Db2.

Logging

Queue-sharing-groups can support persistent messages, because the messages on shared queues can be logged in the queue manager logs.

Coupling facility and structure failures

There are two types of failure that can be reported for a coupling facility (CF) structure: structure failure and loss of connectivity. Sysplex services for data sharing (XES) inform IBM MQ of a CF structure failure or a CF failure with a structure failure event. If XES creates a loss of connectivity event this does not necessarily indicate that there is a problem with the structure, it might be that there is no connection available to communicate with the structure. It is possible that not all of the queue managers receive a

loss of connectivity event for the structure; it depends on the configuration of connections to the CF. A loss of connectivity event can also be received because of operator commands, for example VARY PATH OFFLINE or CONFIG CHP OFFLINE.

The CF structures that are used by IBM MQ can be configured to use system-managed duplexing. This means that if there is a single failure, system-managed failover processing hides the failure of a structure or the loss of connectivity, and the queue manager is not informed of the failure. If there is a failure of both instances of a duplexed structure or connection, the queue manager receives the appropriate event and handles it in the same way as a failure event for a simplex structure. Details of how the queue manager handles the events are described in Scenarios.

In the unlikely event of a CF or structure failure, any nonpersistent messages stored in the affected application structures are lost. You can recover persistent messages using the RECOVER CFSTRUCT command. If a recoverable application structure has failed, any further application activity to this structure is prevented until the structure has been recovered.

To ensure that you can recover a CF structure in a reasonable period of time, take frequent backups, using the BACKUP CFSTRUCT command. You can choose to perform the backups on any queue managers in the queue-sharing group or dedicate one queue manager to perform all the backups. Automate the process of taking backups to ensure that they are taken on a regular basis.

Each backup is written to the active log data set of the queue manager taking the backup. The shared queue Db2 repository records the name of the CF structure being backed up, the name of the queue manager doing the backup, the RBA range for this backup on that queue manager's log, and the backup time.

The administration structure contains information about incomplete units of work on shared queues at the time of any application structure failure so the administration structure must be available during RECOVER CFSTRUCT processing. If the administration structure has failed, all the queue managers in the queue-sharing group must have rebuilt their administration structure entries before you can issue the RECOVER CFSTRUCT command.

Queue managers rebuild their administration structure entries automatically and without terminating. If a queue manager is not running at the time of the failure, its administration structure entries can be rebuilt by another queue manager in the queue-sharing group that is running at the same or higher level.

To recover an application structure, issue a RECOVER CFSTRUCT command to the queue manager that you want to perform the recovery. You can recover a single CF structure or you can recover several CF structures simultaneously. You can recover using any queue manager in the queue-sharing group, it does not have to be the one that performed the backup, or one that has been connected previously to the failed structure.

The RECOVER CFSTRUCT command uses the backup, located through the Db2 repository information (Db2 must therefore be available on the queue manager where recovery is being carried out), and recovers this to the point of failure.

The RECOVER CFSTRUCT command does this by applying log records from every queue manager in the queue-sharing group that has performed an MQPUT or MQGET between the start of the backup and the time of failure, to any shared queue that maps to the CF structure. The resulting merging of the logs might require reading a considerable amount of log data because all the log data written by participating queue managers since the backup is read. You are strongly recommended to make frequent (for example, hourly) backups, especially if there are large messages within the backup.

Structure failure scenarios

Scenarios

If a failure is reported for a CF structure, the action taken by connected queue managers depends on the following:

- The type of failure reported by the XES component of z/OS to IBM MQ.
- The structure type (application or administration)
- The queue manager level
- The CFLEVEL of the IBM MQ CFSTRUCT object (2, 3, 4 or 5. This is not the CFLEVEL of the CFCC microcode)
- The RECAUTO attribute of an IBM MQ CFSTRUCT object at CFLEVEL(5)

The following scenarios describe what happens when a failure is reported for the administration structure:

- If a structure failure event is received for the administration structure, the structure is reallocated and rebuilt automatically without the queue manager terminating. A new instance of the structure is allocated by XES when a queue manager attempts to connect to it.

When the queue manager has connected to the new instance of the structure, the queue manager writes the entries for itself into the structure. This processing is carried out by the queue manager and is not part of XES rebuild processing.

If a queue manager was not running at the time of the failure, or terminates before recovery of its part of the administration structure has been completed, its administration structure entries can be rebuilt by another queue manager in the queue-sharing group.

Administration structure entries of a queue manager can only be rebuilt by another queue manager running at the same level or higher. If administration structure entries of a queue manager cannot be rebuilt by another queue manager in the queue-sharing group, restart the queue manager so that it can complete the rebuild of its part of the structure.

Certain actions are suspended until the administration structure entries for all queue managers have been rebuilt. The suspended actions include the following:

- Opening and closing of shared queues.
- Committing or backing out units of recovery.
- Serialized applications connecting to or disconnecting from the queue manager.
- Backing up or recovering an application structure.

Any serialized applications that have already connected to the queue manager can continue processing. Any serialized application attempting to connect with the MQCNO_SERIALIZE_CONN_TAG_QSG or MQCNO_RESTRICT_CONN_TAG_QSG parameters receive the MQRC_CONN_TAG_NOT_USABLE return code.

When the administration structure entries for the queue manager have been rebuilt, the suspended actions are resumed.

The following scenarios describe what happens when a failure is reported for an application structure:

- If a structure failure event is received for an application structure, and the CFLEVEL is 1 or 2, the queue manager terminates. Restart the queue manager. The first queue manager to attempt to connect to the structure again causes XES to allocate a new instance of the structure.
- If a structure failure event is received for an application structure, and the CFLEVEL is 3, 4, or 5 the queue managers connected to the structure continue to run. Applications that do not use the queues in the failed structure can continue normal processing.

However, applications that attempt operations on queues in the failed structure receive an MQRC_CF_STRUC_FAILED error until the structure has been successfully rebuilt, at which point the application can open the queues again.

Structure rebuild is started automatically for CFLEVEL(5) application structures defined with RECAUTO(YES). Otherwise, the structure will be rebuilt when the RECOVER CFSTRUCT command is issued.

Resilience to coupling facility connectivity failures

What is resilience to coupling facility connectivity failures?

Resilience to coupling facility connectivity failures refers to the ability of queue managers in a queue-sharing-group to tolerate loss of connectivity to a coupling facility structure without terminating. This function also attempts to rebuild the structure in another coupling facility with better connectivity in order to regain access to shared queues as soon as possible.

What is partial loss of connectivity?

IBM MQ defines partial loss of connectivity as a situation where one or more systems in the sysplex lose connectivity to the coupling facility where the structure being accessed by the system is allocated, but at least one system in the sysplex maintains connectivity to the same coupling facility.

What is total loss of connectivity?

IBM MQ defines a total loss of connectivity as a situation where no systems in the sysplex have connectivity to the coupling facility and the structure allocated within it.

Why would you enable this function?

Resilience to coupling facility connectivity failures improves the availability of IBM MQ, allowing non-shared queues to remain available after a queue manager has lost connectivity to one or more coupling facility structures. Additionally, queue managers that lose connectivity to a coupling facility structure automatically attempt to rebuild the structure in another available coupling facility, improving the availability of the shared queues within the queue-sharing-group.

Considerations when enabling this function

A queue manager that tolerates loss of connectivity to coupling facility structures without terminating may not be able to reconnect to a coupling facility structure for some time if there is no alternative coupling facility available. Shared queues defined on a structure that has suffered loss of connectivity remain unavailable until connectivity to the structure is restored. In this situation, applications that connect into members of the queue-sharing-group in order to perform shared queue work may find that the shared queues they need to access are not available. To avoid this situation it is recommended that queue managers should be configured to terminate when connectivity to a coupling facility structure is lost. This termination forces applications to connect to another member of the queue-sharing-group that has connectivity to the coupling facility structures where the shared queues the application requires are defined.

Managing Resilience to coupling facility connectivity failures

How do I enable this functionality?

The following steps must be performed in order to enable resilience to coupling facility connectivity

1. Ensure that the CFRM couple data set has been formatted to support system-managed rebuild. This allows queue managers to initiate a system-managed rebuild to re-create a structure into an available coupling facility. Use the **DISPLAY XCF, COUPLE, TYPE=CFRM** command to determine the format of the CFRM couple data set. To support system-managed rebuild, the CFRM couple data set should be formatted specifying **"ITEM NAME(SMREBLD) NUMBER(1)"**. Refer to the z/OS MVS Setting Up a Sysplex documentation for more information on formatting a CFRM couple data set.
2. Ensure that an alternative coupling facility is available and is in the CFRM preference list for all IBM MQ coupling facility structures. This enables the queue managers to attempt to rebuild structures into an alternative available coupling facility to restore access to the structures as soon as possible.

IBM MQ structures must be defined with ENFORCEORDER(NO) in CFRM policy, so that XCF is able to choose the optimum CF in the configuration if IBM MQ needs to reallocate the structure.

Refer to the z/OS MVS Setting Up a Sysplex documentation for more information about structure preference lists.

3. Alter all application coupling facility structures that need to tolerate loss of connectivity to CFLEVEL(5). This is the minimum level that can tolerate a loss of connectivity.
4. Determine the values required for the **QMGR CFCONLOS** and the **CFSTRUCT CFCONLOS** attributes and alter these accordingly. The **QMGR CFCONLOS** attribute controls whether loss of connectivity to the administration structure is tolerated, and the **CFSTRUCT CFCONLOS** attribute controls whether loss of connectivity is tolerated by each application coupling facility structure. If the default values for these attributes are retained, the queue manager terminates following loss of connectivity to any coupling facility structure.
5. Determine the values required for the **CFSTRUCT RECAUTO** attribute for each application coupling facility structure, and alter these accordingly. This attribute controls whether coupling facility structures should be automatically recovered using logged data following total loss of connectivity. If the default value for this attribute is retained, no automatic recovery is performed for application structures following total loss of connectivity.

Scenario 1 - Loss of connectivity to the administration structure

Queue managers can tolerate loss of connectivity to the administration structure without terminating if all queue managers in the queue-sharing-group are at Version 7.1 or later. If there are queue managers at a level lower than Version 7.1 in the queue-sharing-group, all queue managers in the queue-sharing-group abend with reason code 00C510AB when connectivity to the administration structure is lost.

When connectivity to the administration structure is lost by any queue manager that has been configured to tolerate loss of connectivity to the administration structure, all members of the queue-sharing-group disconnect from the administration structure. All active queue managers in the queue-sharing-group then attempt to reconnect to the administration structure, causing it to be reallocated in the coupling facility with the best connectivity to all systems in the sysplex, and rebuild the administration structure data.

Note: This may not necessarily be the coupling facility which has the best connectivity to all systems that have active queue managers.

If a queue manager cannot reconnect to the administration structure, for example because none of the coupling facilities in the CFRM preference list for the administration structure are available, some shared queue operations remain unavailable until the queue manager can successfully reconnect to the administration structure and rebuild its administration structure data.

Reconnection occurs automatically when a suitable coupling facility becomes available on the system.

Failure to connect to the administration structure during queue manager startup as a result of a lack of connectivity to the coupling facility, or no suitable coupling facility available to allocate the structure, is not tolerated. All active queue managers in the queue-sharing-group then attempt to reconnect to the administration structure, causing it to be reallocated in another coupling facility if one is available, and rebuild the administration structure data.

Scenario 2- Loss of connectivity to the application structure

Loss of connectivity to application structures at **CFLEVEL(5)** or higher can be tolerated without the queue manager terminating. Queue managers connected to application structures at **CFLEVEL(4)** or lower, or structures at **CFLEVEL(5)** that have not been configured to tolerate loss of connectivity, abend with reason code 00C510AB when connectivity to the structure is lost.

When connectivity is lost to an application structure that has been configured to tolerate loss of connectivity, all queue managers that lost connectivity to the structure disconnect. The subsequent behavior of the queue manager depends on whether the loss of connectivity is partial or total.

Partial loss of connectivity to an application structure

If the loss of connectivity is determined to be partial, queue managers that have lost connectivity to the structure attempt to initiate a system-managed rebuild in order to move the structure to another coupling facility with improved connectivity. If this rebuild is successful, both persistent and non-persistent messages in the structure are copied to the other coupling facility, and access to queues on the structure is restored. Queue managers that did not lose connectivity remain connected to the structure, however, operations that access the structure are delayed during the system-managed rebuild process.

If an application structure cannot be rebuilt to another coupling facility with improved connectivity, or some queue managers still do not have connectivity to the structure after it has been rebuilt in another coupling facility, queues defined on the structure remain unavailable on the queue managers that do not have connectivity to the structure until connectivity is restored to the coupling facility. Queue managers automatically reconnect to the structure when it becomes available and access to the shared queues defined on the structure are restored.

Total loss of connectivity to an application structure

If all MVS systems in the sysplex have lost connectivity to the coupling facility that the application structure is allocated in, z/OS deallocates the structure from the coupling facility whenever an attempt is made to reconnect to the structure. It is possible for the queue manager to attempt to reconnect to the structure for several reasons, such as an attempt by an application to open a shared queue, or a notification from the system that new coupling facility resources may have become available. It is therefore likely that all non-persistent messages in the affected structure are lost following total loss of connectivity to an application structure.

Recoverable application structures are automatically recovered following total loss of connectivity, if they have been defined with **RECAUTO(YES)**. The recovery starts almost immediately if an alternative coupling facility is available to allocate the structure in, or whenever such a coupling facility becomes available. If a structure has not been defined with **RECAUTO(YES)**, recovery can be started by issuing the **RECOVER CFSTRUCT** command. This recovers all persistent messages in the structure, but all non-persistent messages are lost. As this process involves reading the queue manager log it can take some time to complete, therefore it is recommended that structure backups be taken regularly to reduce the time until access to the shared queues on the structure is restored.

Queue managers attempt to reconnect to non-recoverable application structures as soon as an application attempts to open a shared queue that is defined on the structure or a notification is received from the system that new coupling facility resources have become available. If a suitable coupling facility is available to allocate the structure in, a new structure is allocated and access to the shared queues defined on the structure is restored. As persistent messages cannot be put to queues defined in non-recoverable structures, all messages on the shared queues are lost.

Operational behavior

If a Version 7.1, or later, queue manager, configured to tolerate loss of connectivity to a particular coupling facility structure loses connectivity, the members of the queue-sharing group attempt to automatically recover from the failure and reconnect to the structure. This activity may involve reallocating the structure in another coupling facility with better connectivity if one is available. However, operator intervention may still be required to recover from the loss of connectivity.

Typically the required operator action is to:

1. Resolve the cause of the failure that resulting in the loss of connectivity.

2. Ensure that a coupling facility where the IBM MQ structures can be allocated is available on all systems in the sysplex

Any structures that have been automatically reallocated in another coupling facility after the loss of connectivity event, can be moved to the coupling facility with the optimal connectivity to all queue managers in the queue-sharing group. If required, this can be done by initiating the system-managed rebuild command **SETXCF START,REBUILD** as documented in z/OS MVS System Commands Reference.

In the case of a partial loss of connectivity to an application structure, the queue managers that lost connectivity to the structure attempt to initiate a system-managed rebuild. This process only allocates the structure in another coupling facility if that coupling facility has connectivity to all active queue managers currently connected to the structure. Therefore, it is possible that where the majority of queue managers in a queue-sharing group have lost connectivity to an application structure, they are unable to rebuild the structure into another coupling facility due to the queue managers that are still connected to the original structure. In this situation the queue managers that are still connected to the original structure can be shut down to allow the structure to be rebuilt, or the **RESET CFSTRUCT ACTION(FAIL)** command can be issued to fail the structure. Recovery can be initiated on applicable structures by issuing the **RECOVER CFSTRUCT** command.

Note: When failing and recovering the structure, all non-persistent messages on the structure are lost.

Security concepts in IBM MQ for z/OS



Use this topic to understand the importance of security for IBM MQ, and the implications of not having adequate security settings on your system.

Why you must protect IBM MQ resources

IBM MQ handles the transfer of information that is potentially valuable. Applying security ensures that the resources IBM MQ owns and manages are protected from unauthorized access. Such access might lead to the loss or disclosure of the information.

You should ensure that none of the following resources are accessed or changed by any unauthorized user or process:

- Connections to IBM MQ
- IBM MQ objects such as queues, processes, and namelists
- IBM MQ transmission links
- IBM MQ system control commands
- IBM MQ messages
- Context information associated with messages

To provide the necessary security, IBM MQ uses the z/OS system authorization facility (SAF) to route authorization requests to an External Security Manager (ESM), for example Security Server (previously known as RACF). IBM MQ does no security verification of its own. Where distributed queuing or clients are being used, you might require additional security measures, for which IBM MQ provides channel authentication records, channel exits, the MCAUSER channel attribute, and TLS.

The decision to allow access to an object is made by the ESM and IBM MQ follows that decision. If the ESM cannot make a decision, IBM MQ prevents access to the object.

What happens if you do not protect IBM MQ resources

If you do nothing about security, the most likely effect is that *all* users can access and change *every* resource. This includes not only local users, but also those on remote systems using distributed queuing or clients, where the logon security controls might be less strict than is normally the case for z/OS.

To enable security checking you must do the following:

- Install and activate an ESM (for example, Security Server).
- Define the MQADMIN class if you are using an ESM other than Security Server.
- Activate the MQADMIN class.

You must consider whether using mixed-case resource names would be beneficial to your enterprise. If you do use mixed-case resource names in your ESM profiles you must define and activate the MXADMIN class.

z/OS Data Set Encryption

Data Set Encryption (DSE) provides the capability to encrypt z/OS data sets, so that the data they contain can only be viewed or modified by user IDs granted the specific permission. This provides encryption of data at rest in the file system, and prevents inadvertent disclosure of sensitive information to users who have a legitimate business need and permissions to manage the data sets themselves.

IBM MQ for z/OS does not support use of DSE with the active logs, page sets, and shared message data sets (SMDS) that provide the primary persistence mechanisms for IBM MQ messages. Instead, Advanced Message Security provides an end-to-end encryption solution for IBM MQ messaging, which encompasses the entire IBM MQ network, encryption of data in flight, at rest, and even inside the runtime IBM MQ processes.

Other VSAM and sequential data sets used in an IBM MQ subsystem can be encrypted using DSE. For example:

- Bootstrap data set (BSDS)
- Sequential files holding system configuration (MQSC) commands read at startup using CSQINPx DDNAMEs
- IBM MQ archive logs, often used for long term archival of IBM MQ log data for audit purposes.

You can encrypt using DSE by allocating a dataclass that is defined with a data set key label. For more information, see [Planning your log archive storage](#).

Related information:

[Security concepts](#)

[Channel authentication records](#)

[Authority to work with IBM MQ objects on z/OS](#)

[Setting up security on z/OS](#)

[Comparing link level security and application level security](#)

[Cryptographic security protocols: TLS](#)

[Messages for IBM MQ for z/OS](#)

[MQSC reference](#)

Security controls and options in IBM MQ for z/OS



You can specify whether security is turned on for the whole IBM MQ subsystem, and whether you want to perform security checks at queue manager or queue-sharing group level. You can also control the number of user IDs checked for API-resource security.

Subsystem security

Subsystem security is a control that specifies whether any security checking is done for the whole queue manager. If you do not require security checking (for example, on a test system), or if you are satisfied with the level of security on all the resources that can connect to IBM MQ (including clients and channels), you can turn security checking off for the queue manager or queue-sharing group so that no further security checking takes place.

This is the only check that can turn security off completely and determine whether any other security checks are performed or not. That is, if you turn off checking for the queue manager or queue-sharing group, no other IBM MQ checking is done; if you leave it turned on, IBM MQ checks your security requirements for other IBM MQ resources.

You can also turn security on or off for particular sets of resources, such as commands.

Queue manager or queue-sharing group level checking

You can implement security at queue manager level or at queue-sharing group level. If you implement security at queue-sharing group level, all the queue managers in the group share the same profiles. This means that there are fewer profiles to define and maintain, making security management easier. It also makes it easy to add a new queue manager to the queue-sharing group because it inherits the existing security profiles.

It is also possible to implement a combination of both if your installation requires it, for example, during migration or if you have one queue manager in the queue-sharing group that requires different levels of security to the other queue managers in the group.

Queue-sharing group level security

Queue-sharing group level security checking is performed for the entire queue-sharing group. It enables you to simplify security administration because it requires you to define fewer security profiles. The authorization of a user ID to use a particular resource is handled at the queue-sharing group level, and is independent of which queue manager that user ID is using to access the resource.

For example, say a server application runs under user ID SERVER and wants access to a queue called SERVER.REQUEST, and you want to run an instance of SERVER on each z/OS image in the sysplex. Rather than permitting SERVER to open SERVER.REQUEST on each queue manager individually (queue manager level security), you can permit access only at the queue-sharing group level.

You can use queue-sharing group level security profiles to protect all types of resource, whether local or shared.

Queue manager level security

You can use queue manager level security profiles to protect all types of resource, whether local or shared.

Combination of both levels

You can use a combination of both queue manager and queue-sharing group level security.

You can override queue-sharing group level security settings for a particular queue manager that is a member of that group. This means that you can perform a different level of security checks on an individual queue manager to those performed on the other queue managers in the group.

For more information, see Profiles to control queue-sharing group or queue manager level security.

Controlling the number of user IDs checked

RESLEVEL is a Security Server profile that controls the number of user IDs checked for IBM MQ resource security. Normally, when a user attempts to access an IBM MQ resource, Security Server checks the relevant user ID or IDs to see if access is allowed to that resource. By defining a RESLEVEL profile you can control whether zero, one or, where applicable, two user IDs are checked.

These controls are done on a connection by connection basis, and last for the life of the connection.

There is only one RESLEVEL profile for each queue manager. Control is implemented by the access that a user ID has to this profile.

Mixed case or uppercase IBM MQ RACF classes

You can now use mixed case RACF profile support, which allows you to use mixed case resource names and define IBM MQ RACF profiles to protect them.

You can choose to either:

- Continue using uppercase only IBM MQ RACF Classes as in previous releases, or
- Use the new mixed case IBM MQ RACF classes.

Without the use of mixed case RACF profiles, you can still use mixed case resource names in IBM MQ for z/OS ; however, these resource names can only be protected by generic RACF profiles in the uppercase IBM MQ classes. When using mixed case IBM MQ RACF profile support you can provide a more granular level of protection by defining IBM MQ RACF profiles in the mixed case IBM MQ classes.

Resources you can protect in IBM MQ for z/OS

z/OS

When a queue manager starts, or when instructed by an operator command, IBM MQ for z/OS determines which resources you want to protect.

You can control which security checks are performed for each individual queue manager. For example, you can implement a number of security checks on a production queue manager, but none on a test queue manager.

Connection security

Connection security checking is carried out either when an application program tries to connect to a queue manager. It is done by issuing an MQCONN or MQCONNX request, or when the channel initiator, or CICS or IMS adapter issues a connection request.

If you are using queue manager level security, you can turn connection security checking off for a particular queue manager. However, if you do this any user can connect to that queue manager.

For the CICS adapter, only the CICS address space user ID is used for the connection security check, not the individual CICS terminal user ID. For the IMS adapter, when the IMS control or dependent regions connect to IBM MQ, the IMS address space user ID is checked. For the channel initiator, the user ID used by the channel initiator address space is checked.

You can turn connection security checking on or off at either queue manager or queue-sharing group level.

Command security

Command security checking is carried out when a user issues an MQSC command from any of the sources described in *Issuing commands*. You can make a separate check on the resource specified by the command as described in *“Command resource security.”*

If you turn off command checking, issuers of commands are not checked to see whether they have the authority to issue the command.

If MQSC commands are entered from a console, the console must have the z/OS SYS console authority attribute. Commands that are issued from the CSQINP1 or CSQINP2 data sets, or internally by the queue manager, are exempt from all security checking while those for CSQINPX use the user ID of the channel initiator address space. You must control who is allowed to update these data sets through normal data set protection.

You can turn command security checking on or off at either queue manager or queue-sharing group level.

Command resource security

Some MQSC commands, for example defining a local queue, involve the manipulation of IBM MQ resources. When command resource security is active, each time a command involving a resource is issued, IBM MQ checks to see if the user is allowed to change the definition of that resource.

You can use command resource security to help enforce naming standards. For example, a payroll administrator might be allowed to delete and define only queues with names beginning "PAYROLL". If command resource security is inactive, no security checks are made on the resource that is being manipulated by the command. Do not confuse command resource security with command security; the two are independent.

Turning off command resource security checking does not affect the resource checking that is done specifically for other types of processing that do not involve commands.

You can turn command resource security checking on or off at either queue manager or queue-sharing group level.

Channel security considerations

Channel security

When you are using channels, the security features available depend on which communications protocol you are going to use. If you use TCP, there are no security features provided with the communications protocol, although you can use TLS. If you are using APPC, you can flow user ID information from the sending MCA through the network to the destination MCA for verification.

For both protocols, you can specify which user IDs you want to check for security purposes, and how many. Again, the choices available to you depend on which protocol you are using, what you specify when you define the channel, and the RESLEVEL settings for the channel initiator.

For more information about the types of channel security available see *Channel authentication records* and *Security exit overview*

Related reference:

“API-resource security in IBM MQ for z/OS”

Resources are checked when an application opens an object with an MQOPEN or an MQPUT1 call. The access needed to open an object depends on what open options are specified when the queue is opened.

API-resource security in IBM MQ for z/OS: 

Resources are checked when an application opens an object with an MQOPEN or an MQPUT1 call. The access needed to open an object depends on what open options are specified when the queue is opened.

API-resource security is subdivided into the following checks:

- Queue
- Process
- Namelist
- Alternate user
- Context

No security checks are performed when opening the queue manager object or when accessing storage class objects.

Queue

Queue security checking controls who is allowed to open which queue, and what options they are allowed to open it with. For example, a user might be allowed to open a queue called PAYROLL.INCREASE.SALARY to browse the messages on the queue (using the MQOO_BROWSE option), but not to remove messages from the queue (using one of the MQOO_INPUT_* options). If you turn checking for queues off, any user can open any queue with any valid open option (that is, any valid MQOO_* option on an MQOPEN or MQPUT1 call).

You can turn queue security checking on or off at either queue manager or queue-sharing group level.

Process

Process security checking is carried out when a user opens a process definition object. If you turn checking for processes off, any user can open any process.

You can turn process security checking on or off at either queue manager or queue-sharing group level.

Namelist

Namelist security checking is carried out when a user opens a namelist. If you turn checking for namelists off, any user can open any namelist.

You can turn namelist security checking on or off at either queue manager or queue-sharing group level.

Alternate user

Alternate user security controls whether one user ID can use the authority of another user ID to open an IBM MQ object.

For example:

- A server program running under user ID PAYSERV retrieves a request message from a queue that was put on the queue by user ID USER1.
- When the server program gets the request message, it processes the request and puts the reply back into the reply-to queue specified with the request message.

- Instead of using its own user ID (PAYSERV) to authorize opening the reply-to queue, the server can specify some other user ID, in this case, USER1. In this example, alternate user security would control whether user ID PAYSERV is allowed to specify user ID USER1 as an alternative user ID when opening the reply-to queue.

The alternative user ID is specified in the *AlternateUserId* field of the object descriptor (MQOD).

You can use alternative user IDs on any IBM MQ object, for example, processes or namelists. It does not affect the user ID used by any other resource managers, for example, for CICS security or for z/OS data set security.

If alternate user security is not active, any user can use any other user ID as an alternative user ID.

You can turn alternate user security checking on or off at either queue manager or queue-sharing group level.

Context

Context is information that is applicable to a particular message and is contained in the message descriptor (MQMD) that is part of the message. The context information comes in two sections:

Identity section

The user of the application that first put the message to a queue. It consists of the following fields:

- *UserIdentifier*
- *AccountingToken*
- *ApplIdentityData*

Origin section

The application that put the message on the queue where it is currently stored. It consists of the following fields:

- *PutApplType*
- *PutApplName*
- *PutDate*
- *PutTime*
- *ApplOriginData*

Applications can specify the context data when either an MQPUT or an MQPUT1 call is made. The application might generate the data, the data might be passed on from another message, or the queue manager might generate the data by default. For example, server programs can use context data to check the identity of the requester, that is, did this message come from the correct application? Typically, the *UserIdentifier* field is used to determine the user ID of an alternative user.

You use context security to control whether the user can specify any of the context options on any MQOPEN or MQPUT call. For information about the context options, see the MQOPEN options relating to message context. For descriptions of the message descriptor fields relating to context, see MQMD - Message descriptorMQMD - Message descriptor.

If you turn context security checking off, any user can use any of the context options that the queue security allows.

You can turn context security checking on or off at either queue, queue manager or queue-sharing group level.

Availability on z/OS

z/OS

IBM MQ for z/OS has many features for high availability. This topic describes some of the considerations for availability.

Several features of IBM MQ can increase system availability if the queue manager or channel initiator fails. For more information about these features, see the following sections:

- Sysplex considerations
- Shared queues
- Shared channels
- IBM MQ network availability
- Using the z/OS Automatic Restart Manager (ARM)
- Using the z/OS Extended Recovery Facility (XRF)
- Using the z/OS GROUPUR attribute for recovery in a queue-sharing group
- Where to find more information about availability

Sysplex considerations

In a *sysplex*, a number of z/OS operating system images collaborate in a single system image and communicate using a coupling facility. IBM MQ can use the facilities of the sysplex environment for enhanced availability.

Removing the affinities between a queue manager and a particular z/OS image allows a queue manager to be restarted on a different z/OS image in the event of an image failure. The restart mechanism can be manual, use ARM, or use system automation, if you ensure the following:

- All page sets, logs, bootstrap data sets, code libraries, and queue manager configuration data sets are defined on shared volumes.
- The subsystem definition has sysplex scope and a unique name within the sysplex.
- The level of *early code* installed on every z/OS image at IPL time is at the same level.
- TCP virtual IP addresses (VIPA) is available on each TCP stack in the sysplex, and you have configured IBM MQ TCP listeners and inbound connections to use VIPAs rather than default host names.

For more information about using TCP in a sysplex, see *TCP/IP in a sysplex*, SG24-5235, an IBM Redbooks publication.

You can additionally configure multiple queue managers running on different operating system images in a sysplex to operate as a queue-sharing group, which can take advantage of shared queues and shared channels for higher availability and workload balancing.

Shared queues

In the queue-sharing group environment, an application can connect to any of the queue managers within the queue-sharing group. Because all the queue managers in the queue-sharing group can access the same set of shared queues, the application does not depend on the availability of a particular queue manager; any queue manager in the queue-sharing group can service any queue. This gives greater availability if a queue manager stops because all the other queue managers in the queue-sharing group can continue processing the queue. For information about high availability of shared queues, see “Advantages of using shared queues” on page 343.

To further enhance the availability of messages in a queue-sharing group, IBM MQ detects if another queue manager in the group disconnects from the coupling facility abnormally, and completes units of work for that queue manager that are still pending, where possible. This is known as *peer recovery* and is described in “Peer recovery” on page 423.

Peer recovery cannot recover units of work that were in doubt at the time of the failure. You can use the Automatic Restart Manager (ARM) to restart all the systems involved in the failure (CICS, Db2, and IBM MQ for example), and to ensure that they are all restarted on the same new processor. This means that they can resynchronize, and gives rapid recovery of in-doubt units of work. This is described in “Using the z/OS Automatic Restart Manager (ARM).”

Shared channels

In the queue-sharing group environment, IBM MQ provides functions that give high availability to the network. The channel initiator enables you to use networking products that balance network requests across a set of eligible servers and hide server failures from the network (for example, VTAM generic resources). IBM MQ uses a generic port for inbound requests so that attach requests can be routed to any available channel initiator in the queue-sharing group. This is described in “Shared channels” on page 364.

Shared outbound channels take the messages they send from a shared transmission queue. Information about the status of a shared channel is held in one place for the whole queue-sharing group level. This means that a channel can be restarted automatically on a different channel initiator in the queue-sharing group if the channel initiator, queue manager, or communications subsystem fails. This is called *peer channel recovery* and is described in Shared outbound channels.

IBM MQ network availability

IBM MQ messages are carried from queue manager to queue manager in an IBM MQ network using channels. You can change the configuration at a number of levels to improve the network availability of a queue manager, and the ability of an IBM MQ channel to detect a network problem and to reconnect.

TCP *Keepalive* is available for TCP/IP channels. It causes TCP to send packets periodically between sessions to detect network failures. The KAINTE channel attribute determines the frequency of these packets for a channel.

AdoptMCA allows a channel, blocked in receive processing as a result of a network outage, to be terminated and replaced by a new connection request. You control AdoptMCA using the ADOPTMCA queue manager property with the MQSC utility or the AdoptNewMCAType property with the Programmable Command Formats interface.

ReceiveTimeout prevents a channel from being permanently blocked in a network receive call. The RCVTIME and RCVTMIN channel initiator parameters, determine the receive timeout characteristics for channels, as a function of their heartbeat interval. See Queue manager parameter for more details.

Using the z/OS Automatic Restart Manager (ARM)

You can use IBM MQ for z/OS in conjunction with the z/OS automatic restart manager (ARM). If a queue manager or a channel initiator has failed, ARM restarts it on the same z/OS image. If z/OS fails, a whole group of related subsystems and applications also fail. ARM can restart all the failed systems automatically, in a predefined order, on another z/OS image within the sysplex. This is called a cross-system restart.

ARM enables rapid recovery of in-doubt transactions in the shared queue environment. It also gives higher availability if you are not using queue-sharing groups.

You can use ARM to restart a queue manager on a different z/OS image within the sysplex in the event of z/OS failure.

To enable automatic restart, you must do the following:

1. Set up an ARM coupling data set.
2. Define the automatic restart actions that you want z/OS to perform in an *ARM policy*.
3. Start the ARM policy.

If you want to restart queue managers in different z/OS images automatically, every queue manager in each z/OS image on which that queue manager might be restarted must be defined with a sysplex-wide unique 4-character subsystem name.

Using ARM with IBM MQ is described in *Using ARM in an IBM MQ network*.

Using the z/OS Extended Recovery Facility (XRF)

You can use IBM MQ in an extended recovery facility (XRF) environment. All IBM MQ-owned data sets (executable code, BSDSs, logs, and page sets) must be on DASD shared between the active and alternative XRF processors.

If you use XRF for recovery, you must stop the queue manager on the active processor and start it on the alternative processor. For CICS, you can do this using the command list table (CLT) provided by CICS, or the system operator can do it manually. For IMS, this is a manual operation and you must do it after the coordinating IMS system has completed the processor switch.

IBM MQ utilities must be completed or terminated before the queue manager can be switched to the alternative processor. Consider the effect of this potential interruption carefully when planning your XRF recovery plans.

Take care to prevent the queue manager starting on the alternative processor before the queue manager on the active processor terminates. A premature start can cause severe integrity problems in data, the catalog, and the log. Using global resource serialization (GRS) helps avoid the integrity problems by preventing simultaneous use of IBM MQ on the two systems. You must include the BSDS as a protected resource, and you must include the active and alternative XRF processors in the GRS ring.

Using the z/OS GROUPUR attribute for recovery in a queue-sharing group

Queue-sharing groups (QSG) allow additional transactional facilities which are described in this topic. The GROUPUR attribute allows XA client applications to have any in-doubt transaction recovery that may be required, performed on any member of the QSG.

If an XA client application connects to a queue-sharing group (QSG) through a Sysplex it cannot guarantee which specific queue manager it connects to. Use of the GROUPUR attribute by queue managers within the queue-sharing group can enable any in-doubt transaction recovery that may be necessary to occur on any member of the QSG. Even if the queue manager to which the application was initially connected is not available, transaction recovery can take place.

This feature frees the XA client application from any dependency on specific members of the QSG and thus extends the availability of the queue manager. The queue-sharing group appears to the transactional application as a single entity providing all the IBM MQ features and without a single queue manager point of failure.

This functionality is not apparent to the transactional application.

Where to find more information about availability

You can find more information about these topics from the following sources:

Table 28. Where to find more information about availability

Topic	Where to look
Queue-sharing groups	“Shared queues and queue-sharing groups” on page 322
System parameters	Configuring system parameters
Using the Automatic Restart Manager Utility programs	Using ARM in an IBM MQ network
MQSC commands	MQSC commands

Monitoring and statistics on IBM MQ for z/OS



IBM MQ for z/OS has a set of facilities for monitoring the queue manager, and gathering statistics.

IBM MQ supplies facilities for monitoring the system and collecting statistics. For further information about these facilities, see the following sections:

- “Online monitoring”
- “IBM MQ trace”
- “Events” on page 440

Online monitoring

IBM MQ includes the following commands for monitoring the status of IBM MQ objects:

- DISPLAY CHSTATUS displays the status of a specified channel.
- DISPLAY QSTATUS displays the status of a specified queue.
- DISPLAY CONN displays the status of a specified connection.

For more information about these commands, see The MQSC commands.

IBM MQ trace

IBM MQ supplies a trace facility that you can use to gather the following information while the queue manager is running:

Performance statistics

The statistics trace gathers the following information to help you monitor performance and tune your system:

- Counts of different MQI requests (message manager statistics)
- Counts of different object requests (data manager statistics)
- Information about Db2 usage (Db2 manager statistics)
- Information about Coupling Facility usage (Coupling Facility manager statistics)
- Information about SMDS usage (shared message data set statistics)
- Information about buffer pool usage (buffer manager statistics)
- Information about logging (log manager statistics)
- Information about storage usage (storage manager statistics)
- Information about lock requests (lock manager statistics)

Accounting data

- The accounting trace gathers information about the processor time spent processing MQI calls and about the number of MQPUT and MQGET requests made by a particular user.
- IBM MQ can also gather information about each task using IBM MQ. This data is gathered as a thread-level accounting record. For each thread, IBM MQ also gathers information about each queue used by that thread.

The data generated by the trace is sent to the System Management Facility (SMF) or the generalized trace facility (GTF).

Events

IBM MQ events provide information about errors, warnings, and other significant occurrences in a queue manager. By incorporating these events into your own system management application, you can monitor the activities across many queue managers, for multiple IBM MQ applications. In particular, you can monitor all the queue managers in your system from a single queue manager.

Events can be reported through a user-written reporting mechanism to an administration application that supports the presentation of the events to an operator. Events also enable applications acting as agents for other administration networks, for example NetView[®], to monitor reports and create the appropriate alerts.

Related information:

Using IBM MQ trace

Using IBM MQ events

Unit of recovery disposition on z/OS



Certain transactional applications can use a GROUP, rather than a QMGR, unit of recovery disposition when connected to a queue manager in a queue-sharing group (QSG) by specifying the QSG name when they connect instead of the queue manager name. This allows transaction recovery to be more flexible and robust by removing the requirement to reconnect to the same queue manager in the QSG.

Transactions started by applications that have connected using the queue-sharing group name also have a GROUP unit of recovery disposition.

When a transactional application connects with a GROUP unit of recovery disposition it is logically connected to the queue-sharing group and does not have an affinity to any specific queue manager. Any 2-phase commit transactions that it has started that have completed phase-1 of the commit process, that is, they are in doubt, can be inquired and resolved, when connected to any queue manager within the QSG. In a recovery scenario this means that the transaction coordinator does not have to reconnect to the same queue manager, which may be unavailable at that time.

Applications that connect with a QMGR unit of recovery disposition have a direct affinity to the queue manager to which they are connected. In a recovery scenario the transaction coordinator must reconnect to the same queue manager to resolve any in-doubt transactions, irrespective of whether the queue manager belongs to a queue-sharing group.

When applications specify a queue-sharing group name, and thus connect to a queue manager in a QSG with a GROUP unit of recovery disposition, the queue-sharing group is logically a separate resource manager. This means that in-doubt transactions are only visible to an application if it reconnects with the same unit of recovery disposition. In-doubt transactions with a QMGR unit of recovery disposition are not visible to applications that have connected with a GROUP unit of recovery disposition and vice versa.

Related concepts:

“Enabling GROUP units of recovery”

A queue-sharing group can configure and enable support for GROUP units of recovery.

“Application support” on page 442

Use this page to determine which applications can connect with a GROUP unit of recovery disposition.

Enabling GROUP units of recovery

z/OS

A queue-sharing group can configure and enable support for GROUP units of recovery.

To use GROUP units of recovery on a queue manager within a QSG, enable the GROUPPUR queue manager attribute. For more information about this concept, see “Unit of recovery disposition on z/OS” on page 440 before reading the rest of this topic.

When the GROUPPUR queue manager attribute is enabled, the queue manager accepts new connections with a GROUP unit of recovery disposition. If you disable this attribute new connections with this disposition are not accepted, although applications already connected is unaffected until they disconnect.

When an application connects with a GROUP unit of recovery disposition and either inquires what transactions are in doubt or attempts to resolve a transaction that was started elsewhere in the queue-sharing group (QSG), the queue manager to which it is now connected must be able to communicate with the other members of the queue-sharing group so that it can process the request. To do this it uses a shared queue called SYSTEM.QSG.UR.RESOLUTION.QUEUE. This queue must be on a recoverable application structure called CSQSYSAPPL. The structure must be recoverable because persistent messages are stored on this queue when processing resolution requests.

Before you can enable GROUP units of recovery, you must ensure that the coupling facility structure and the shared queue are defined. You can use the definitions in the CSQ4INSS sample. When the queue is defined, or detected during startup, each queue manager in the queue-sharing group opens the queue so that it can receive incoming requests. If you want to delete or move the queue because it has been defined incorrectly you can request that the queue managers close their open handles on it by updating the queue object to inhibit MQGET requests. When you have made the necessary corrections, permitting applications to get messages from the queue once more directs each queue manager to reopen it. Use the DISPLAY QSTATUS command to identify what handles are open on a queue.

When you have completed this setup you can then enable GROUP units of recovery on each queue manager that you want transactional applications to be able to connect to with a GROUP unit of recovery disposition. This need not be all of the queue managers within the queue-sharing group but if you choose to only enable this functionality on a subset of the queue-sharing group you must ensure that your applications only attempt to connect to queue managers on which you have enabled it. For more information, see “Application support” on page 442.

When you attempt to enable the GROUPPUR queue manager attribute, a number of configuration checks are performed. The queue manager checks that:

- It belongs to a queue-sharing group.
- The shared-queue called SYSTEM.QSG.UR.RESOLUTION.QUEUE has been defined, according to the definition in CSQ4INSS.
- The SYSTEM.QSG.UR.RESOLUTION.QUEUE is on a recoverable CF structure called CSQSYSAPPL.
- Group Units of Recovery are not restricted by its mode of operation (OPMODE).

If any of the above checks fail, the GROUPPUR attribute remains disabled and a message code is returned.

These configuration checks are also performed at queue manager startup if the queue manager attribute is enabled. If any of the checks fail during startup GROUP units of recovery is disabled and the queue

manager issues a message identifying which check failed. When you have performed the necessary corrective action you must then re-enable the queue manager attribute.

Application support

z/OS

Use this page to determine which applications can connect with a GROUP unit of recovery disposition.

Support for the GROUP unit of recovery disposition is limited to certain types of transactional applications for which IBM MQ for z/OS is a resource manager but not the transaction coordinator. Currently supported transactional applications are:

- IBM MQ extended transactional client applications
- IBM MQ classes for JMS applications running in an application server, such as WebSphere Application Server.
- CICS applications running in CICS TS 4.2, when the CICS MQCONN resource definition is configured with RESYNCMEMBER(GROUPRESYNC).

Related concepts:

“IBM MQ extended transactional client applications”

Use this page to determine how IBM MQ extended transactional client applications can use the GROUP unit of recovery disposition.

“CICS applications” on page 443

Use this page to determine how CICS can use the GROUP unit of recovery disposition.

IBM MQ extended transactional client applications: z/OS

Use this page to determine how IBM MQ extended transactional client applications can use the GROUP unit of recovery disposition.

An example of an IBM MQ extended transactional client application is one that uses JMS and runs in WebSphere Application Server, connecting to IBM MQ over TCP/IP, rather than local bindings. These client applications connect to IBM MQ for z/OS over network connections, such as via TCP/IP. For these applications, it is the value specified for the QMNAME parameter of the xa_info string passed in the xa_open call that specifies whether a QMGR or GROUP unit of recovery disposition is used. For more information about xa_open, see The format of an xa_open string and Additional error processing for xa_open. For JMS applications this is done by specifying the name of the queue-sharing group (QSG) in the ConnectionFactory instead of the name of a specific queue manager.

For XA client applications to take advantage of using the GROUP unit of recovery disposition you must configure your TCP/IP setup to allow your client applications to be routed to the queue managers in the queue-sharing group that have the GROUPUR attribute enabled, rather than a specific queue manager. One of the dynamic virtual IP address technologies that you can use to do this is the z/OS SysPlex Distributor. See z/OS Communications Server and z/OS Basic Skills: Dynamic virtual addressing for more details. If you want to enable GROUP units of recovery on a subset of the queue managers in your queue-sharing group, ensure that your client applications cannot be routed to those on which it is not enabled. This includes any queue manager at a version earlier than Version 7.0.1.

To use the group unit of recovery disposition feature in a JMS client application you must use the Version 7.0.1 JMS client libraries. Earlier levels of JMS client libraries use queue manager (QMGR) unit of recovery disposition.

Your client applications do not have to connect to the queue-sharing group using shared channels.

CICS applications:

Use this page to determine how CICS can use the GROUP unit of recovery disposition.

CICS 4.2 and later provides the group resynchronization option, RESYNCMEMBER(GROUPRESYNC) in an MQCONN resource definition. A CICS configured with this option can connect to any suitable queue manager in a queue-sharing group which is running on the same LPAR as that CICS region. To support the CICS GROUPRESYNC option, a queue manager must be running at MQ V7.1 or later, and be enabled for GROUPUR support.

Transactions running within a CICS region connected to MQ using GROUPRESYNC create units of work with GROUP unit of recovery disposition.

You can use RESYNCMEMBER(GROUPRESYNC) to enable faster recovery after a queue manager failure as it enables the CICS region to immediately connect to an alternative eligible queue manager running on the same LPAR, resolving any indoubt transactions as necessary, without waiting for queue manager restart.

RESYNCMEMBER(GROUPRESYNC) also enables more flexible restart options for CICS. A CICS region with its MQ connection configured to use GROUPRESYNC and MQ shared queues can be restarted on any LPAR where there is a queue manager running as a member of the same queue-sharing group.

IBM MQ and other z/OS products

Use this topic to understand how IBM MQ can work with other z/OS products.

Related concepts:

“IBM MQ and CICS”

All the CICS versions supported by IBM MQ Version 9.0.0, and later, use the CICS supplied version of the adapter and bridge.

“IBM MQ for z/OS and WebSphere Application Server” on page 450

Use this topic to understand the use of IBM MQ for z/OS by the WebSphere Application Server.

Related reference:

“IBM MQ and IMS” on page 445

Use this topic to understand how IBM MQ works with IMS. The IMS adapter allows you to connect your queue manager to IMS, and enables IMS applications to use the MQI.

“IBM MQ and the z/OS Batch, TSO, and RRS adapters” on page 449

Use this topic to understand how IBM MQ works with the z/OS Batch, TSO, and RRS adapters.

IBM MQ and CICS

All the CICS versions supported by IBM MQ Version 9.0.0, and later, use the CICS supplied version of the adapter and bridge.

For more information about configuring the IBM MQ CICS adapter, and the IBM MQ CICS bridge components, see the Configuring connections to MQ section of the CICS documentation.

Related information:

Using IBM MQ with CICS

CICS group attach

z/OS

CICS group attach provides the ability for a CICS region to connect to any active member of an IBM MQ queue-sharing group on the same LPAR rather than specifying an individual queue manager. CICS still connects to a single queue manager at a time.

You require at least two queue managers on the LPAR to support CICS group attach. Using group attach provides higher availability as you do not need a particular queue manager to be active. CICS connects to any queue manager in the queue-sharing group on the LPAR.

For more information, see the CICS documentation on the MQCONN resource.

CICS attempts to connect to MQNAME passed as if it were a queue manager:

- If the queue manager exists and is active, the connection will work.
- If the connection fails, CICS queries the status of queue managers in the group to ascertain which are active on same LPAR.
- If multiple queue managers are active, CICS checks for RESYNCMEMBER(YES) and the UOW status to determine whether CICS needs to connect, or should connect, to a particular member, or wait if not active.
- If there is no need to connect to a particular member, CICS selects a queue manager (using a randomizing algorithm).
- CICS attempts to connect to chosen queue manager.
- If the attempt fails then, depending upon the return code, CICS chooses the next member, then goes through the selection loop again.
- If no queue managers are active, CICS issues multiple connections to the list of queue managers and waits on ECBLIST until the first queue manager becomes available.

Related concepts:

“Group units of recovery (GROUPUR) for CICS”

The IBM MQ GROUPUR for CICS provides peer recovery for in-doubt units of work in a queue-sharing group (QSG). One IBM MQ queue manager can resolve in-doubt units of work on behalf of another queue manager in the queue-sharing group. This means that if CICS reconnects through group attach to a different queue manager in the QSG, it can resolve indoubt transactions from a previous IBM MQ connection.

Related information:

 Support for IBM MQ queue-sharing groups

Group units of recovery (GROUPUR) for CICS

z/OS

The IBM MQ GROUPUR for CICS provides peer recovery for in-doubt units of work in a queue-sharing group (QSG). One IBM MQ queue manager can resolve in-doubt units of work on behalf of another queue manager in the queue-sharing group. This means that if CICS reconnects through group attach to a different queue manager in the QSG, it can resolve indoubt transactions from a previous IBM MQ connection.

If a CICS region is working with a queue manager, and the queue manager ends abnormally, then any indoubt transactions are recovered. This eliminates the need for the CICS region to wait for the queue manager that it was working with to restart, and then resolve any in doubt units of work. This means

that you need at least two queue managers on the LPAR, so that CICS can connect to another queue manager in the event of an abnormal termination of the first queue manager.

The new RESYNCMEMBER(GROUPRESYNC) setting on the CICS MQCONN definition:

- Uses the IBM MQ group attach function and peer recovery.
- Requires a queue manager with the GROUPPUR attribute enabled.
- Still supports the existing CICS MQCONN RESYNCMEMBER settings (YES and NO):
 - Uses the existing CICS group attach function and no peer recovery.
 - Changing RESYNCMEMBER settings takes effect next time CICS connects to IBM MQ.

Related concepts:

“Enabling GROUP units of recovery” on page 441

A queue-sharing group can configure and enable support for GROUP units of recovery.

IBM MQ and IMS

▶ z/OS

Use this topic to understand how IBM MQ works with IMS. The IMS adapter allows you to connect your queue manager to IMS, and enables IMS applications to use the MQI.

The optional additional IBM MQ - IMS bridge enables applications to run an IMS application that does not use the MQI. This means that you can use your legacy applications with IBM MQ, without the need to rewrite them.

For more information about these components, see the following subtopics:

Related information:

Setting up the IMS adapter

Setting up the IMS bridge

Operating the IMS adapter

IMS and IMS bridge applications on IBM MQ for z/OS

MQIIH - IMS information header

The IMS adapter

▶ z/OS

The IMS adapter is an interface between IMS application programs and an IBM MQ subsystem.

The IBM MQ adapters enable different application environments to send and receive messages through a message queuing network. The IMS adapter is the interface between IMS application programs and an IBM MQ subsystem. It makes it possible for IMS application programs to use the MQI.

The IMS adapter receives and interprets requests for access to IBM MQ using the External Subsystem Attach Facility (ESAF) provided by IMS. This facility is described in the *IMS Customization Guide*. Usually, IMS connects to IBM MQ automatically without operator intervention.

The IMS adapter provides access to IBM MQ resources for programs running in the following modes or states:

- Task (TCB) mode
- Problem state
- Non-cross-memory mode
- Non-access register mode

The adapter provides a connection thread from an application task control block (TCB) to IBM MQ.

The adapter supports a two-phase commit protocol for changes made to resources owned by IBM MQ with IMS acting as the syncpoint coordinator. Conversations where IMS is not the syncpoint coordinator, for example APPC-protected (SYNCLVL=SYNCPT) conversations, are not supported by the IMS adapter.

The adapter also provides a trigger monitor transaction (CSQQTRMN). This is described in “The IMS trigger monitor.”

You can use IBM MQ with the IMS Extended Recovery Facility (XRF) to aid recovery from a IMS error. For more information about XRF, see the *IMS Administration Guide: System* manual.

Using the adapter

The application programs and the IMS adapter run in the same address space. The queue manager is separate, in its own address space.

You must link-edit each program that issues one or more MQI calls to a suitable IMS language interface module, and, unless it uses dynamic MQI calls, the IBM MQ-supplied API stub program, CSQQSTUB. When the application issues an MQI call, the stub transfers control to the adapter through the IMS external subsystem interface, which manages the processing of the request by the message queue manager.

System administration and operation with IMS

An authorized IMS terminal operator can issue IMS commands to control and monitor the connection to IBM MQ. However, the IMS terminal operator has no control over the IBM MQ address space. For example, the operator cannot shut down IBM MQ from an IMS address space.

Restrictions

The following IBM MQ API calls are not supported within an application using the IMS adapter:

- MQCB
- MQCB_FUNCTION
- MQCTL

The IMS trigger monitor

The IMS trigger monitor (**CSQQTRMN**) is an IBM MQ-supplied IMS application that starts an IMS transaction when an IBM MQ event occurs, for example, when a message is put onto a specific queue.

How it works

When a message is put onto an application message queue, a trigger is generated if the trigger conditions are met. The queue manager then writes a message (containing some user-defined data), known as a *trigger message*, to the initiation queue that has been specified for that message queue. In an IMS environment, you can start an instance of CSQQTRMN to monitor an initiation queue and to retrieve the trigger messages from it as they arrive. Typically, CSQQTRMN schedules another IMS transaction by an INSERT (ISRT) to the IMS message queue. The started IMS application reads the message from the application message queue and then processes it. CSQQTRMN must run as a non-message BMP.

Each copy of CSQQTRMN services a single initiation queue. When it has started, the trigger monitor runs until IBM MQ or IMS ends.

The APPLCTN macro for CSQQTRMN must specify SCHDTYP=PARALLEL.

Because the trigger monitor is a batch-oriented BMP, IMS transactions that are started by the trigger monitor contain the following:

- Blanks in the LTERM field of the IOPCB
- The PSB name of the trigger monitor BMP in the Userid field of the IOPCB

If the target IMS transaction is protected by Security Server (previously known as RACF), you might need to define CSQQTRMN as a user ID to Security Server.

The IBM MQ - IMS bridge

z/OS

The IBM MQ - IMS bridge is the component of IBM MQ for z/OS that allows direct access from IBM MQ applications to applications on your IMS system.

The IBM MQ - IMS bridge enables *implicit MQI support*. This means that you can re-engineer legacy applications that were controlled by 3270-connected terminals to be controlled by IBM MQ messages, without having to rewrite, recompile, or re-link them. The bridge is an IMS *Open Transaction Manager Access (OTMA)* client.

In bridge applications there are no IBM MQ calls within the IMS application. The application gets its input using a GET UNIQUE (GU) to the IOPCB and sends its output using an ISRT to the IOPCB. IBM MQ applications use the IMS header (the MQIIH structure) in the message data to ensure that the applications can execute as they did when driven by nonprogrammable terminals. If you are using an IMS application that processes multi-segment messages, note that all segments should be contained within one IBM MQ message.

The IMS bridge is illustrated in Figure 78.

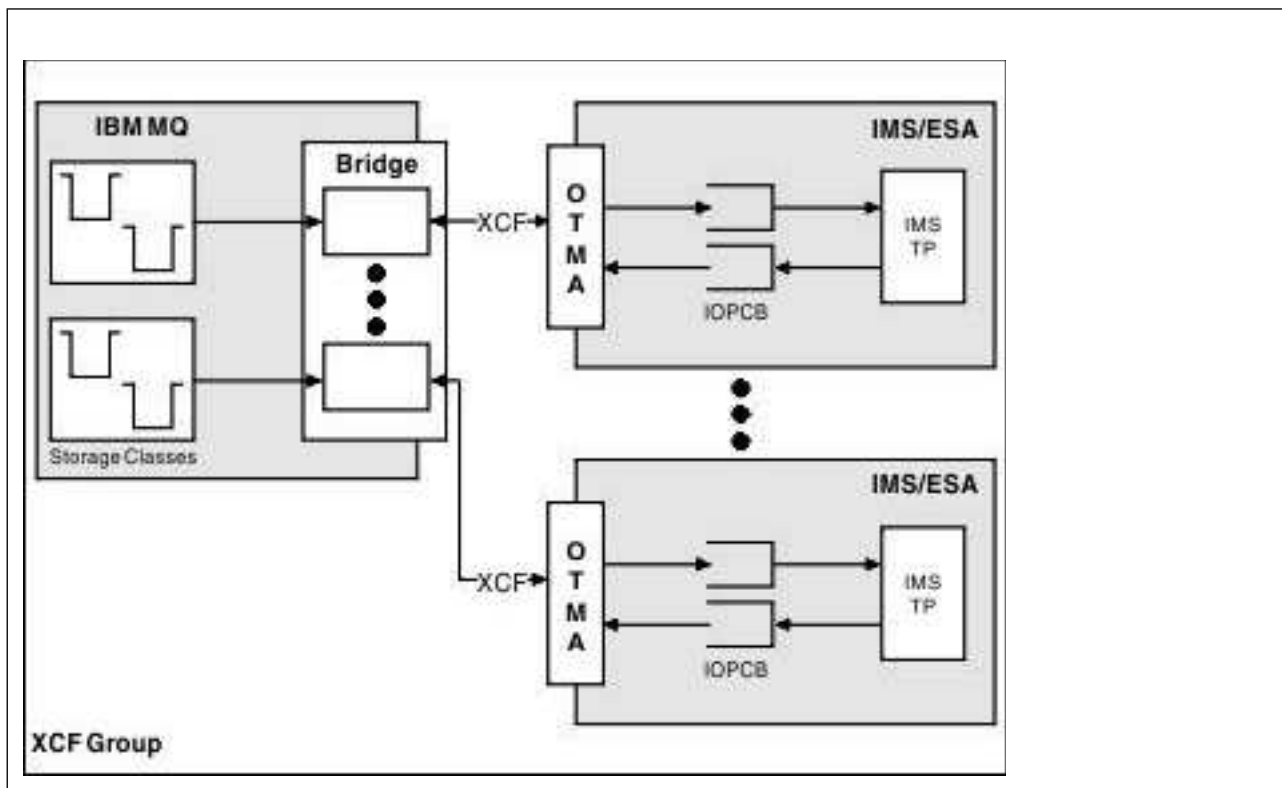


Figure 78. The IBM MQ - IMS bridge

A queue manager can connect to one or more IMS systems, and more than one queue manager can connect to one IMS system. The only restriction is that they must all belong to the same XCF group and must all be in the same sysplex.

What is OTMA?

The IMS OTMA facility is a transaction-based connectionless client/server protocol that runs on IMS Version 5.1 or later. It functions as an interface for host-based communications servers accessing IMS TM applications through the *z/OS Cross Systems coupling facility (XCF)*.

OTMA enables clients to connect to IMS to provide high performance for interactions between clients and IMS for a large network or large number of sessions. OTMA is implemented in a z/OS sysplex environment. Therefore, the domain of OTMA is restricted to the domain of XCF.

OTMA Resource Monitoring

Support for the x'3C' OTMA protocol messages, available in IMS v10 or higher, has been added to the IBM MQ - IMS bridge in IBM MQ for z/OS v7.1. These messages are sent to OTMA clients by IMS to report its health status. If an IMS partner is unable to process the volume of transaction requests being sent then it will notify IBM MQ that a flood warning has occurred. In response IBM MQ will slow down the rate at which requests are sent over the bridge. If IMS is still unable to process the transaction requests and a full flood condition occurs all TPIPEs to the IMS partner are suspended. Upon notification from the IMS partner that the flood or flood-warning condition has been relieved IBM MQ will resume all suspended TPIPEs, if appropriate, and gradually increase the rate at which transaction requests are sent until the maximum rate is achieved. Console messages are issued by IBM MQ in response to a change in the status of IMS partners.

IBM MQ for z/OS v7.1 will take no action in response to the x'3C' messages unless new functions have been enabled - see *Z/OS: OPMODE*.

If IMS v10 partners are being used you should ensure that PTF UK45082 has been applied.

Submitting IMS transactions from IBM MQ

To submit an IMS transaction that uses the bridge, applications put messages on an IBM MQ queue as usual. The messages contain IMS transaction data; they can have an IMS header (the MQIIH structure) or allow the IBM MQ - IMS bridge to make assumptions about the data in the message.

IBM MQ then puts the message to an IMS queue (it is queued in IBM MQ first to enable the use of syncpoints to assure data integrity). The storage class of the IBM MQ queue determines whether the queue is an *OTMA queue* (that is, a queue used to transmit messages to the IBM MQ - IMS bridge) and the particular IMS partner to which the message data is sent.

Remote queue managers can also start IMS transactions by writing to these OTMA queues on IBM MQ for z/OS.

Data returned from the IMS system is written directly to the IBM MQ reply-to queue specified in the message descriptor structure (MQMD). (This might be a transmission queue to the queue manager specified in the *ReplyToQMgr* field of the MQMD.)

Related reference:

“IBM MQ and IMS” on page 445

Use this topic to understand how IBM MQ works with IMS. The IMS adapter allows you to connect your queue manager to IMS, and enables IMS applications to use the MQI.

Related information:

Customizing the IMS bridge

IMS and IMS bridge applications on IBM MQ for z/OS

IBM MQ and the z/OS Batch, TSO, and RRS adapters



Use this topic to understand how IBM MQ works with the z/OS Batch, TSO, and RRS adapters.

Introduction to the Batch adapters

The Batch/TSO adapters are the interface between z/OS application programs running under JES, TSO, or UNIX and Linux System Services and IBM MQ. They enable z/OS application programs to use the MQI.

The adapters provide access to IBM MQ resources for programs running in the following modes or states:

- Task (TCB) mode
- Problem or supervisor state
- Non-cross-memory mode
- Non-access register mode

Connections between application programs and IBM MQ are at the task level. The adapters provide a connection thread from an application task control block (TCB) to IBM MQ.

The Batch/TSO adapter supports a single-phase commit protocol for changes made to resources owned by IBM MQ. It does not support multi-phase commit protocols. The RRS adapter enables IBM MQ applications to participate in two-phase commit protocols with other RRS-enabled products, coordinated by z/OS Resource Recovery Services (RRS).

The adapters use the z/OS STIMERM service to schedule an asynchronous event every second. This event runs an interrupt request block (IRB) that does not involve any waiting by the batch application's task. This IRB checks to see if the IBM MQ termination ECB has been posted. If the termination ECB has been posted, the IRB posts any application ECBs that are waiting on an event in IBM MQ (for example, a signal or a wait).

The Batch/TSO adapter

The IBM MQ Batch/TSO adapter provides IBM MQ support for z/OS Batch and TSO applications. All application programs that run under z/OS Batch or TSO must have the API stub program CSQBSTUB link-edited with them. The stub provides the application with access to all MQI calls. You use single-phase commit and backout for applications by issuing the MQI calls **MQCMIT** and **MQBACK**.

The RRS adapter

Resource Recovery Services (RRS) is a subcomponent of z/OS that provides a system-wide service for coordinating two-phase commit across z/OS products. The IBM MQ Batch/TSO RRS adapter (the RRS adapter) provides IBM MQ support for z/OS Batch and TSO applications that want to use these services. The RRS adapter enables IBM MQ to become a full participant in RRS coordination. Applications can participate in two-phase commit processing with other products that support RRS (for example, Db2).

The RRS adapter provides two stubs; you must link-edit application programs that want to use RRS with one of these stubs.

CSQBRSTB

This stub allows you to use two-phase commit and backout for applications by using the RRS callable resource recovery services instead of the MQI calls **MQCMIT** and **MQBACK**.

You must also link-edit module ATRSCSS from library SYS1.CSSLIB with your application. If you use the MQI calls **MQCMIT** and **MQBACK**, you will receive return code MQRC_ENVIRONMENT_ERROR.

CSQBRRSI

This stub allows you to use MQI calls **MQCMIT** and **MQBACK**; IBM MQ actually implements these calls as the **SRRCMIT** and **SRRBACK** RRS calls.

For information about building application programs that use the RRS adapter, see The RRS batch adapter.

Where to find more information about the z/OS Batch, TSO, and RRS adapters

You can find more information about the topics in this section in the following sources:

Table 29. Where to find more information about using z/OS Batch with IBM MQ

Topic	Where to look
Setting up the Batch adapters	Task 19: Set up Batch, TSO, and RRS adapters
RRS callable resource recovery services	<i>MVS Programming: Callable Services for High Level Languages</i>

IBM MQ for z/OS and WebSphere Application Server

z/OS

Use this topic to understand the use of IBM MQ for z/OS by the WebSphere Application Server.

Applications written in Java that are running under WebSphere Application Server can use the Java Messaging Service (JMS) specification to perform messaging. Point-to-point messaging in this environment can be provided by an IBM MQ for z/OS queue manager.

A benefit of using an IBM MQ for z/OS queue manager to provide the messaging is that connecting JMS applications can participate fully in the functionality of an IBM MQ network. For example, they can use the IMS bridge, or exchange messages with queue managers running on other platforms.

Connection between WebSphere Application Server and a queue manager

You can choose either *client transport* or *bindings transport* for the queue connection factory object. If you choose *bindings transport*, the WebSphere Application Server and the queue manager must both exist on the same z/OS image.

Note that you require native libraries for *bindings transport*.

Both types of connection support transactional applications: the *client transport* by using XA protocols; the *bindings transport* by using a WebSphere Application Server stub, CSQBWSTB, which uses RRS services.

For more information about configuring queue connection factories see *IBM MQ Using Java*.

Using IBM MQ functions from JMS applications

By default, JMS messages held on IBM MQ queues use an MQRFH2 header to hold some of the JMS message header information. Many legacy IBM MQ applications cannot process messages with these headers, and require their own characteristic headers, for example the MQCIH for CICS Bridge, or MQWIH for IBM MQ Workflow applications. For more details about these special considerations, see Mapping JMS messages onto IBM MQ messages.

Index

A

- abnormal termination
 - maintaining consistency 416
 - what happens to IBM MQ 417
- accessibility features for this product 93
- accounting trace 440
- active log
 - introduction 320, 391
- adapters, illustration 316
- addressing information 193
- administration
 - object name transformation 178
 - queue manager name transformation 178, 180
 - understanding IBM MQ file names 178, 180
- advantages of using IBM MQ
 - clients 289
- alias queue
 - examples of when to use 162
- alias queue, default definition 403
- alternate channels 190
- alternate user security 434
- application environments,
 - introduction 322
- application programs
 - receiving messages 154
 - retrieving messages from queues 154
 - sending messages 154
 - time-independent applications 147
- applications on IBM MQ MQI
 - clients 289
- archive log
 - introduction 320, 391
- archiving, system parameters 401
- ARM (Automatic Restart Manager)
 - concepts 437
 - shared queues 437
- attributes
 - queue manager 167
 - queues 165
- authentication information
 - default definition 403
- authentication information objects
 - description of 169
- authorization service 297
- Automatic Restart Manager (ARM)
 - concepts 437
 - shared queues 437
- availability
 - Automatic Restart Manager (ARM) 437
 - coupling facility 326
 - example 343
 - Extended Recovery Facility (XRF) 438
 - increased 436
 - network 363, 437
 - shared channels 437
 - shared queues 324, 436
 - sysplex considerations 436

B

- back out 414
- backlevel clients 295
- backup
 - coupling facility structures 328, 423
- base queue 162
- Batch adapter 449
- benefits
 - intra-group queuing 372
- benefits of using IBM MQ clients 289
- bootstrap data set (BSDS)
 - dual mode 399
 - introduction 320, 399
- BSDS (bootstrap data set)
 - dual mode 399
 - introduction 320, 399
- buffer pools
 - illustration 388
 - introduction 320, 388
 - performance statistics 439
 - relationship to messages and page sets 388
 - sample definitions 408
- buffers 384

C

- capacity planning 327
- case of RACF classes 432
- CF (coupling facility) 181
- CF structures
 - amount of data held 327
 - backup and recovery 328
 - definition 327, 330
 - introduction 326
 - name 327, 330
 - peer recovery 423
- CFSTRUCT
 - OFFLOAD 328
- channel
 - client-connection 170, 172, 187
 - cluster-receiver 187
 - cluster-receiver, overview 200
 - cluster-sender 187
 - description of 170
 - message 170, 172
 - MQI 170, 172, 187
 - overview 170, 172
 - receiver 187
 - requester 187
 - requester-sender 187
 - requester-server 187
 - segregating messages 190
 - sender-receiver 187
 - server-connection 170, 172, 187
 - server-receiver 187
 - sharing 190
 - using alternate channels 190
- channel authentication queue 406

- channel initiator
 - and queue-sharing groups,
 - illustration 363
 - generic port 364
 - illustration 317
 - increased availability 436
 - introduction 317
 - required queues 404
 - sample object definitions 409, 410
 - sample startup definitions 412
 - shared channels 364
 - system parameters 401
 - workload balancing 365
- channel listener
 - introduction 318
- channels
 - default definition 403
 - features 365
 - peer recovery 365
 - security 433
 - shared 364
- channels, alternate to 190
- checkpoint log records 394
- CICS
 - adapter 443
 - bridge 443
 - consistency with IBM MQ 414
 - working with 443
- CICS adapter
 - sample object definitions 410
- client connection channels
 - description of 173
- client to server connection 293
- client-connection channel 187
 - defining 170, 172
- clients
 - compatibility 295
 - version 295
- clients and servers
 - definitions 287
- CLUSRCVR
 - overview 200
- cluster channels
 - overview 200
- cluster components 183
- cluster queue 199
- cluster queue manager
 - overview 198
- cluster-receiver 187
- cluster-receiver channel 187
- cluster-sender 187
- cluster-sender channel 187
- clustered topics
 - publish/subscribe 201
- clustering
 - with intra-group queuing 378
- clusters
 - and queue-sharing groups,
 - illustration 367
 - cluster transmission queues 166
 - components 183

- clusters (*continued*)
 - queue-sharing groups 367
 - sample object definitions 409, 410
 - using 190
- clusters, use of
- components 197
 - objects 202
- command queues
 - description of 166
- command resource security 433
- command security 433
- commit
 - single-phase 414
 - two-phase 414
- commit point, definition 413
- communication
 - connectionless 149
 - time-independent 150
- communication protocol
 - overview 172
- communications
 - overview 172
- compensating log records 394
- components
 - cluster 197
- components of distributed-queuing environment
 - message channel 187
- components, cluster 183
- concepts
 - Automatic Restart Manager (ARM) 437
 - bootstrap data set (BSDS) 390
 - buffer pools 388
 - channel initiator 317
 - commit point 413
 - logging 390
 - monitoring 439
 - page sets 385
 - point of consistency 413
 - queue managers 316
 - queue-sharing groups 322, 342
 - security 429
 - shared queues 322, 342
 - statistics 439
 - storage classes 386
 - storage management 384
 - syncpoint 413
 - system parameters 401
 - termination 417
 - unit of recovery 413
- concepts of
 - intra-group queuing 370
- configurations
 - intra-group queuing 375
- connecting, WebSphere Application Server 450
- connection
 - client to server 293
 - overview 172
- connection environment, system
 - parameters 401
- connection security 432
- connectionless communication 149
- controlling log compression 393
- coupling facility (CF) 181
 - abnormal disconnection from 423

- coupling facility (CF) (*continued*)
 - amount of data held 327
 - backup and recovery 328, 423
 - failure 423
 - illustration 324
 - introduction 326
- Coupling Facility (CF)
 - performance statistics 439
 - SMDS 439
- coupling facility structures
 - amount of data held 327
 - backup and recovery 328
 - definition 327, 330
 - introduction 326
 - name 327, 330
 - peer recovery 423
- creating
 - a dynamic (temporary) queue 155
 - a model queue 155
 - a predefined (permanent) queue 155
- CSQINP1
 - sample 408
- CSQINP2
 - samples 408

D

- data manager, performance statistics 439
- data set space management 386
- data-sharing group 325
- Db2
 - in a queue-sharing group, illustration 325
 - performance statistics 439
 - shared repository 324
- DCE Generic Security Service (GSS)
 - name service, installable service 297
- dead-letter header, MQDLH 190
- dead-letter queue 160
 - introduction 405
 - overview 190
 - sample definition 410
- dead-letter queues
 - description of 166
 - MQDLH, dead-letter header 190
- DEFAULT storage class 405
- default transmission queue, sample definition 410
- defaults
 - objects 182
- distributed queuing
 - and queue-sharing groups, illustration 363
 - components 187
 - default transmission queue 406
 - generic port 364
 - illustration 317
 - intra-group queuing 366
 - introduction 317
 - required queues 404
 - sample object definitions 409, 410
 - sample startup definitions 412
 - shared channels 364
 - with intra-group queuing 376
 - workload balancing 365
- Distributed queuing and clusters 192
- dual logging 392

- dynamic queue
 - overview 163
- dynamic queues
 - description of 155

E

- event publications 205
- event queue 160
- event queues
 - description of 166
- event-driven processing 150
- events 440
- example
 - cluster of queue managers 186
 - message channels
 - cluster-receiver 187
 - cluster-sender 187
 - requester-sender 187
 - requester-server 187
 - sender-receiver 187
 - multi-hopping 190
 - multiple queue manager configuration 232
 - passing through intermediate queue managers 190
 - publish/subscribe queue manager configuration 231
 - reply-to-queue alias 196
 - sending messages 184
 - sending messages in both directions 183
 - sharing a transmission queue 190
 - using multiple channels 192
- expired messages 407
- Extended Recovery Facility (XRF) 438
- extended transactional client
 - defined 292
 - hardware and software requirements 293
 - introduction 292
 - supported platforms 293
 - supported transaction managers 293
- extending queue manager facilities 297

F

- file names 178, 180
- files
 - names 178
 - naming conventions 178
 - understanding names 178, 180
- full function IBM MQ 450

G

- generic port 364
- global units of work
 - definition of 296
- glossary 103
- group unit of work 422, 438
- GROUPUR 422, 438

H

- hardware and software requirements 293
- high availability
 - coupling facility 326
 - example 343
 - network 363
 - shared queues 324

I

- IBM JMS extensions
 - introduction
 - for programmers 298
- IBM MQ classes for JMS
 - introduction
 - for programmers 298
- IBM MQ for z/OS
 - illustration 316
- IBM MQ MQI clients and servers 149
- IMS
 - consistency with IBM MQ 414
 - resolving in-doubt units of recovery 421
- IMS adapter
 - IMS language interface module 446
 - introduction 445
 - MQ API restrictions 446
 - trigger monitor 446
 - using the adapter 446
- IMS bridge
 - illustration 447
 - introduction 447
 - submitting transactions 448
- in-backout unit of recovery 416
- in-commit unit of recovery 416
- in-doubt unit of recovery
 - definition 416
 - resolving from IMS 421
 - resolving from RRS 421
- in-flight unit of recovery 416
- index queues
 - rebuilding indexes 419
- initialization parameters 321
- initiation queue 160
- initiation queues
 - description of 165
- installable services
 - authorization service 297
 - installable services, list of 297
 - name service 297
 - service component 297
- installation verification program (IVP)
 - sample definitions 412
- intercommunication
 - concepts 192
- intra-group queuing
 - benefits 372
 - concepts of 370
 - configurations 375
 - introduction 366
 - messages 380
 - required queues 405
 - sample object definitions 409
 - with clustering 378
 - with distributed queuing 376

- Intra-group queuing benefits 372
- Intra-group queuing concepts 370
- Intra-group queuing configurations 375
- Intra-group queuing with clustering 378
- Intra-group queuing with distributed queuing 376
- Introduction to
 - distributed queuing 183
- IVP (installation verification program)
 - sample definitions 412

J

- Java Messaging Service
 - use with WebSphere Application Server 450
 - using IBM MQ functions 451
- JMS
 - use with WebSphere Application Server 450
 - using IBM MQ functions 451
- joining a cluster 198

L

- listener
 - introduction 318
- listener objects
 - description of 173
- local queue manager 183
- local queue, default definition 403
- local queues
 - specific queues used by IBM MQ 165
- local unit of work
 - definition of 296
- lock manager, performance statistics 439
- log compression 392
- log offload, illustration 397
- log record sequence number (LRSN) 393
- logging
 - active log 391
 - adding log data sets 396
 - archive log 391
 - dual logging 392
 - illustration 396
 - introduction 320, 390
 - log compression 392
 - log record sequence number (LRSN) 393
 - performance statistics 439
 - relative byte address (RBA) 393
 - shared queue messages 328
 - single logging 392
 - system parameters 401
 - when the log is offloaded 397
 - when the log is written 396
- LRSN (log record sequence number) 393

M

- MCA (message channel agent) 190
- MCA, introduction 317
- message
 - definition 151
 - descriptor 151

- message channel 170, 172
 - cluster-receiver 187
 - cluster-sender 187
 - receiver 187
 - requester 187
 - requester-sender 187
 - requester-server 187
 - sender 187
 - sender-receiver 187
 - server 187
 - server-receiver 187
- message channel agent (MCA) 190
- message channel agent, introduction 317
- message queuing 147
 - features 149
- message-driven processing 147
- messages
 - application data 154
 - definition of 154
 - intra-group queuing 380
 - maximum length for shared queues 327
 - message descriptor 154
 - message lengths 154
 - message-driven processing 147
 - queuing 147
 - relationship to buffer pools and page sets 388
 - retrieval algorithms 155
 - retrieving messages from queues 154
 - sending and receiving 154
 - storing 384
- Messages put to SYSTEM.QSG.TRANSMIT.QUEUE 380
- model queue, default definition 403
- model queues
 - creating a model queue 155
- mover 317
- moving service component 183
- MQ
 - reduced function 450
- MQ commands
 - object attribute names 181
- MQ extended transactional client 292
- MQ JMS extensions
 - introduction
 - for programmers 298
- MQ object
 - creating 156
 - queue 157
 - queue manager 167
 - storage class 173
 - topic 174
- MQ server
 - platform support 290
- MQDLH, dead-letter header 190
- MQI (message-queuing interface)
 - definition of 149
 - receiving messages 154
 - sending messages 154
- MQI channel 287
- MQI channels 170, 172, 187
- MQI, performance statistics 439
- multi-hopping 190

N

- name resolution 178
 - introduction 193
- name server, introduction 318
- name service 297
- name transformations 178
- namelists
 - default definition 403
 - description of 168
 - security 434
- naming conventions
 - files 178
 - object names 175
 - queue manager name transformation 178
 - rules 175
- NetBIOS 183
- NetView 440
- network, availability 437
- NODEFINE storage class 405
- normal termination 417

O

- OAM (object authority manager)
 - authorization service, installable service 297
 - overview 285
- object
 - creating 156
 - queue 157
 - queue manager 167
 - storage class 173
 - topic 174
- object definitions
 - backing up shared 423
 - where stored 385
- object name transformation 178
- objects 202
 - attributes of 181
 - description of 170
 - name transformation 178
 - naming conventions 175
 - rules 175
 - object name transformation 178
 - process definitions 168
 - system default objects 182
 - types of 157
- online monitoring 439
- Open Transaction Manager Access (OTMA) 448
- OTMA (Open Transaction Manager Access) 448

P

- page set control log records 395
- page sets
 - illustration 388
 - introduction 320, 385
 - maximum size 385
 - page set zero 385
 - relationship to messages and buffer pools 388
 - relationship to queues and storage classes 387

- page sets (*continued*)
 - sample definitions 408
 - space management 386
- PCF (programmable command format)
 - object attribute names 181
- peer recovery
 - shared channels 365
 - shared queues 344
- performance
 - trace 439
- permanent (predefined) queues 155
- persistent messages, queue-sharing groups 423
- platforms
 - for IBM MQ MQI clients 290
 - for IBM MQ servers 290
- point of consistency, definition 413
- port, generic 364
- predefined (permanent) queues 155
- private channels, features 365
- process definitions
 - description of 168
- processes
 - default definition 403
 - security 434
- proxy subscription aggregation
 - publish/subscribe 238
- publication aggregation
 - publish/subscribe 238
- publications under syncpoint
 - introduction 206
- publish/subscribe
 - proxy subscription aggregation 238
 - publication aggregation 238
 - publication scope 247
 - PUBSCOPE topic attribute 247
 - SUBSCOPE topic attribute 247
 - subscription scope 247
 - wildcard rules 239
- publish/subscribe clustered topics 201
- publish/subscribe clusters 235
- publish/subscribe hierarchies 238
- Publish/Subscribe initialization
 - sample startup definitions 411
- publish/subscribe objects 402
- publisher
 - introduction 204

Q

- QSG (queue-sharing group)
 - what it is 181
- queue
 - alias 162
 - base 162
 - cluster 199
 - creating 157
 - dead letter 160
 - dead-letter 160
 - event 160
 - initiation 160
 - introduction to 157
 - name resolution 178
 - remote
 - using 161
 - shared 199
 - system command 160

- queue (*continued*)
 - system default 160
 - transmission 159
 - undelivered message 160
- queue manager
 - alias 193
 - attributes 167
 - definition 151
 - joining a cluster 198
 - number per system 152
 - types 183
 - workload management 168
- queue manager alias
 - introduction 193
- queue managers
 - communication between 317, 366
 - extending queue manager facilities 297
 - illustration 316
 - in a queue-sharing group, illustration 324
 - increased availability 436
 - introduction 316
 - name transformation 178, 180
- queue-sharing group
 - high availability 324
 - maximum message length 327
- queue-sharing group (QSG)
 - what it is 181
- queue-sharing groups
 - advantages 343
 - and clusters, illustration 367
 - and distributed queuing, illustration 363
 - Automatic Restart Manager (ARM) 437
 - availability 436
 - clusters 367
 - definition 325
 - distributed queuing 363
 - illustration 324
 - increased availability 436
 - introduction 320
 - peer recovery 423
 - persistent messages 423
 - recovery 328, 422
 - required queues 405
 - resolving units of work
 - manually 344
 - sample object definitions 409
 - security 431
 - shared repository 324
 - specifying name 326
 - transactional recovery 422
- queues
 - attributes 165
 - definition of 154
 - dynamic (temporary) queues 155
 - extending queue manager facilities 297
 - mapping to page sets 386
 - model queues 155
 - predefined (permanent) queues 155
 - relationship to storage classes and page sets 387
 - retrieving messages from 154
 - security 434

- queues (*continued*)
 - specific local queues used by IBM MQ 165
 - storing 384
- queuing
 - definition 147
 - features 149

R

- RBA (relative byte address) 393
- receiver
 - channel 187
- receiver channel definition
 - overview 183
- recovery 151
 - after abnormal termination 416
 - after coupling facility failure 423
 - by peers in shared channel environment 365
 - by peers in shared queue environment 344
 - CF structures 328
 - coupling facility structures 328
 - introduction 321
 - queue-sharing groups 422
 - shared queue messages 328
 - XRF 438
- reduced function IBM MQ 450
- relative byte address (RBA) 393
- remote queue
 - using 161
- remote queue definition
 - introduction 190
 - what it is 190
- remote queue manager 183
- remote queue, default definition 403
- REMOTE storage class 405
- replicated local queue 364
- reply-to queue
 - alias definition 195
 - aliases 193
 - preparing for 196
 - specifying 195
- reply-to queues
 - description of 166
- repository
 - building 198
 - keeping up-to-date 198
 - sending information to 198
- repository queue manager, overview 198
- requester channel 187
- RESLEVEL security profile 432
- resolution of queue names 178
- resource manager 295
- Resource Recovery Services (RRS)
 - adapter 449
 - resolving in-doubt units of recovery 421
- resources
 - updating under sync point control 295
- restart
 - after abnormal termination 416
 - after coupling facility failure 423
 - introduction 321
 - queue-sharing groups 422

- retained publication
 - introduction 205
- retrieval algorithms for messages 155
- RRS (Resource Recovery Services)
 - adapter 449
 - resolving in-doubt units of recovery 421

S

- sample definitions, system objects 407
- security 151
 - channel security 285
 - channels 433
 - enabling 430
 - introduction 429
 - name service security, overview 285
 - number of user IDs checked 432
 - OAM 285
 - object authority manager (OAM) 285
 - queue manager level 431
 - queue-sharing group level 431
 - RACF class, case of 432
 - RESLEVEL profile 432
 - resources you can protect 432
 - sample definitions 408
 - TLS 285
- segregating messages 190
- sender channel 187
- sender channel definition
 - overview 183
- server
 - channel 187
 - platform support 290
- server queue manager 287
- server-connection channel 187
 - defining 170, 172
- servers 287
- service component 297
- service objects
 - description of 174
- setting up IBM MQ clients 291
- shared channels
 - availability 437
 - features 365
 - inbound 364
 - increased availability 437
 - outbound 365
 - peer recovery 365
 - required queues 405
 - sample object definitions 409
 - status table 366
 - workload balancing 365
- shared message data set 328
- shared queue 199
 - what it is 181
- shared queues 422, 438
 - accessing 325
 - advantages 343, 367
 - and clusters, illustration 367
 - and distributed queuing, illustration 363
 - Automatic Restart Manager (ARM) 437
 - availability 436
 - clusters 367
 - default definition 403

- shared queues (*continued*)
 - definition 323
 - distributed queuing 363
 - high availability 324
 - illustration 343
 - increased availability 436
 - introduction 320
 - maximum message length 327
 - peer recovery 344, 423
 - recovery 328, 422
 - required queues 405
 - resolving units of work manually 344
 - shared repository 324
 - transactional recovery 422
 - where messages are held 326
- shared repository
 - advantages 324
 - introduction 324
- shared transmission queue 365
- sharing channels 190
- single logging 392
- single-phase commit 414
- SMDS
 - coupling facility 328
- SMS, using with IBM MQ 396
- SNA 183
- source queue manager 183
- SPX 183
- state publications 205
- storage classes 384
 - changing 387
 - default definition 403
 - illustration 387
 - introduction 386
 - relationship to queues and page sets 387
 - required definitions 405
 - sample definitions 410
- storage management 384
- Storage Management Subsystem (SMS), using with IBM MQ 396
- storage manager, performance statistics 439
- store-and-forward 150
- subscriber
 - introduction 204
- subsystem security 431
- supervisor, introduction 318
- supported transaction managers 293
- sync point 295
 - overview 151
- sync point control 295
- sync point coordinator 295
- syncpoints
 - definition 413
 - queue manager performance 406
- SYSLNGLV storage class 405
- sysplex 181
- sysplex, increased availability 436
- system administration objects
 - introduction 404
 - sample definitions 408
- system command objects
 - introduction 403
 - sample definitions 408
- system command queue 160

- system default objects 182
 - sample definitions 408
- system default queue 160
- system design 232
- system objects
 - introduction 402
 - sample command to display 411
 - sample definitions 407
- system parameters 401
- SYSTEM storage class 405
- SYSTEM.ADMIN.* queues 404
- SYSTEM.CHANNEL.INITQ 404
- SYSTEM.CHANNEL.REPLY.INFO queue 404
- SYSTEM.CHANNEL.SYNCQ 404
- SYSTEM.CLUSTER.COMMAND
 - .QUEUE 198, 202
- SYSTEM.CLUSTER.HISTORY
 - .QUEUE 202
- SYSTEM.CLUSTER.REPOSITORY
 - .QUEUE 198, 202
- SYSTEM.CLUSTER.TRANSMIT
 - .QUEUE 202
- SYSTEM.COMMAND.* queues 403
- SYSTEM.DEF.CLUSRCVR 202
- SYSTEM.DEF.CLUSSDR 202
- SYSTEM.QSG.* queues 405
- SYSTEM.QSG.TRANSMIT.QUEUE 366
- SYSVOLAT storage class 405

T

- target queue manager 183
- task list 291
- TCP/IP 183
- temporary (dynamic) queues 155
- termination
 - abnormal 417
 - normal 417
- threads
 - accounting statistics 440
- time-independent applications 147
- time-independent communication 150
- TLS tasks, introduction 318
- topic attribute
 - PUBSCOPE 247
 - SUBSCOPE 247
- topic object
 - attributes 174
- topics
 - introduction 204
- trace
 - introduction 439
 - system parameters 401
- transaction 295
- transaction manager
 - definition 295
 - external 295
- transactional support
 - updating under sync point control 295
- transmission header
 - alias
 - definition 194
- transmission queue 159
 - default 406
 - sample definition 410

- transmission queue (*continued*)
 - shared 365
 - sharing 190
- transmission queues
 - cluster transmission queues 166
 - description of 165
- Transport Layer Security (TLS)
 - overview 285
- transport type
 - supported 183
- TSO adapter 449
- two phase commit protocol 295
- two-phase commit
 - illustration 415
 - introduction 414

U

- undo/redo log records 394
- unit of recovery
 - back out 414
 - definition 413
 - illustration 413
 - illustration of back out 414
 - in-backout 416
 - in-commit 416
 - in-doubt 416
 - in-flight 416
 - log records 394
 - peer recovery 344
 - resolving from IMS 421
 - resolving from RRS 421
- unit of work 295
- UNIX operating system
 - object authority manager (OAM) 285
- unresolved unit of work, peer recovery 344
- user ID
 - maximum length 175
- user IDs, number checked for security 432

V

- VTAM generic resources 364

W

- WebSphere Application Server,
 - connecting to queue manager 450
- WILDCARD parameter
 - ALTER TOPIC 221, 239
- wildcard rules
 - publish/subscribe 239
- Windows operating system
 - object authority manager (OAM) 285
- workload balancing
 - shared channels 365
 - shared queues 343, 367
- workload management
 - queue manager 168

X

- XA Interface 295

Notices

This information was developed for products and services offered in the U.S.A.

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
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: 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 states 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 Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

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

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Software Interoperability Coordinator, Department 49XA
3605 Highway 52 N
Rochester, MN 55901
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

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.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

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 the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Programming interface information

Programming interface information, if provided, is intended to help you create application software for use with this program.

This book contains information on intended programming interfaces that allow the customer to write programs to obtain the services of WebSphere MQ.

However, this information may also contain diagnosis, modification, and tuning information. Diagnosis, modification and tuning information is provided to help you debug your application software.

Important: Do not use this diagnosis, modification, and tuning information as a programming interface because it is subject to change.

Trademarks

IBM, the IBM logo, ibm.com[®], are trademarks of IBM Corporation, registered in many jurisdictions worldwide. A current list of IBM trademarks is available on the Web at “Copyright and trademark information” www.ibm.com/legal/copytrade.shtml. Other product and service names might be trademarks of IBM or other companies.

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

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

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

This product includes software developed by the Eclipse Project (<http://www.eclipse.org/>).

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

Sending your comments to IBM

We appreciate your input on this publication. Feel free to comment on the clarity, accuracy, and completeness of the information or give us any other feedback that you might have.

Use one of the following methods to send us your comments:

- Send an email to ibmkc@us.ibm.com
- Use the form on the web here: www.ibm.com/software/data/rcf/

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate, without incurring any obligation to you.

Include the following information:

- Your name and address
- Your email address
- Your telephone or fax number
- The publication title and order number
- The topic and page number related to your comment
- The text of your comment

IBM or any other organizations will only use the personal information that you supply to contact you about the issues that you submit.

Thank you for your participation.

