IBM WebSphere MQ

Product Overview

Version 7 Release 1

Note

Before using this information and the product it supports, read the information in “Notices” on page 361 (*WebSphere MQ V7.1 Installing Guide*).

This edition applies to version 7 release 1 of WebSphere 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.

Contents

Figures

Tables

Product overview

This section provides introductory information to help you get started with IBM® WebSphere® MQ:

- "Introduction to IBM WebSphere MQ"
- "Introduction to IBM WebSphere MQ Telemetry" on page 3
- "WebSphere MQ Version 7.1 PDF documentation" on page 8
- "What's new in IBM WebSphere MQ Version 7.1" on page 8
- "Mappings from the old IBM WebSphere MQ books to the new product documentation" on page 58
- "IBM WebSphere MQ Technical overview" on page 72
- "Glossary" on page 306
- "Accessibility features for IBM WebSphere MQ" on page 351
- "Legal information for WebSphere MQ" on page 352

Related concepts:

Designing a WebSphere MQ architecture (*WebSphere MQ V7.1 Installing Guide*)

Introduction to IBM WebSphere MQ

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

What is IBM WebSphere MQ?

- Software that enables programs to communicate across a network of unlike components (processors, operating systems, subsystems, and communication protocols) using a simple and consistent application programming interface. It is *messaging and queuing* middleware, with *publish/subscribe*:
  - Messaging: programs communicate by sending each other data in messages rather than by calling each other directly.
  - Queuing: the messages are placed on queues in storage, so that programs can run independently of each other, at different speeds and times, in different locations, and without having a logical connection between them.
  - Publish/subscribe: a program can send (*publish*) data to a single destination, and let IBM WebSphere MQ deal with the distribution of that data to other programs (*subscribers*). The publisher defines a *topic* for the information, and the subscriber specifies what topics it wants to receive.
  - Multicast: more efficient than traditional unicast publish/subscribe messaging, multicast can be scaled to a high number of subscribers without detrimental effects in performance. Fair delivery enables near simultaneous delivery, ensuring that no recipient gains an advantage. Multicast enables the user to configure which, if any, of the message attributes are transmitted along with the message, reducing the size of the message and enabling low latency messaging.
  - Telemetry: IBM WebSphere MQ Telemetry features extend the universal messaging backbone provided by IBM WebSphere MQ to a wide range of remote sensors, actuators and telemetry devices. Telemetry extends IBM WebSphere MQ so that it can interconnect intelligent enterprise applications, services, and decision makers with networks of instrumented devices.

What can it do for me?

- IBM WebSphere MQ sends and receives data between your applications, and over networks.
- Message delivery is *assured* and *decoupled* from the application.
- Your application programmers do not need to have communications programming knowledge.

How do I use it?

- Use the IBM WebSphere MQ Explorer and its associated tools to create your initial configuration.
- Use the MQ API (*MQI*) in your application to *connect* to a queue manager, *open* queues and topics, and *put* and *get* messages.

How does it work?

- A queue or a topic is owned and run by a *queue manager*.
- Objects such as queues and queue managers exist as sets of information, together with their data, kept by IBM WebSphere MQ in storage.
- The queue manager is started by a command you enter interactively, or by a program that runs when the operating system boots. For example, on Windows the queue manager can run under the IBM WebSphere MQ Windows service (called `IBM MQSeries`).
- When your application wants to transfer data to another application, it puts data into messages, and then puts the messages onto a queue (or publishes them with a topic). Then there are five main ways that the messages can be retrieved:
  - Another application attached to the same queue manager (or another part of the same application) retrieves the messages from the same queue where they were put.
  - The queue manager is configured to send the messages through a *channel* over the network to a *remote queue* on a queue manager on another computer. An application on that computer retrieves them from the remote queue.
  - An application on another computer pulls the messages across the network when it needs them.
  - A subscriber application is sent the published message by a queue manager.
  - On z/OS®, you can configure multiple queue managers to share messages, and an application attached to any one of these queue managers can get them.
- You can have many queues and topics on one queue manager.
- You can have more than one queue manager on one computer.
- You can have a computer with a *client* installation with no queue manager. The client uses the queue manager on a *server* installation on another computer for messaging.

What tools and resources come with IBM WebSphere MQ?

- On Windows, and Linux x86 and x86-64 platforms:The *WebSphere MQ Explorer*, in which you manage your queue manager and queue configuration, and from where you can launch the following utilities:
  - The *Postcard* application to quickly demonstrate messaging and verify your installation
  - Tutorials
- The *control* commands.
- The *WebSphere MQ Script commands (MQSC)*.
- The *Programmable Command Format (PCF)* commands.
- Sample programs.
- The documentation guide and reference material, supplied on CD in Eclipse plug-in format, viewable and installable on Windows and Linux.

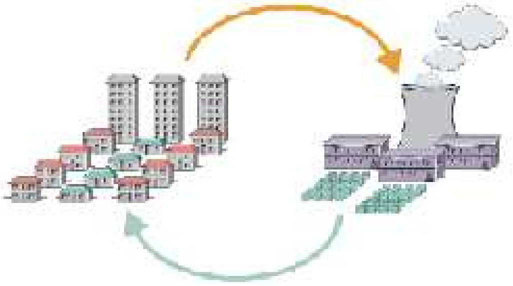
See also:

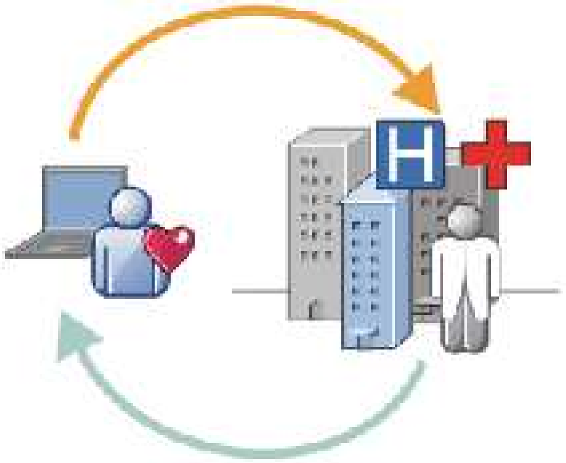
- For a comprehensive introduction, see Introduction to message queuing.

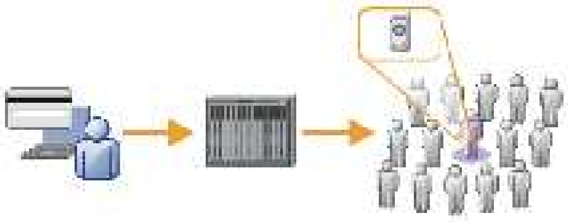
Introduction to IBM WebSphere MQ Telemetry

People, businesses, and governments increasingly want to use IBM WebSphere MQ Telemetry to interact more smartly with the environment we live and work in. IBM WebSphere MQ Telemetry connects all kinds of devices to the internet and to the enterprise, and reduces the costs of building applications for smart devices.

The following diagrams demonstrate some typical uses of IBM WebSphere MQ Telemetry:

Telemetry: Smart Electricity	
	• MQTT message containing energy usage data sent to service provider. • IBM WebSphere MQ Telemetry sends CONTROL COMMANDS based on analysis of energy usage data. • For more information, see the following scenario: "Telemetry scenario: Home energy monitoring and control" on page 106

Telemetry: Smart Health Services	
• IBM WebSphere 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 scenario: "Telemetry scenario: Home patient monitoring" on page 104	

Telemetry: One in a Crowd	
	• A simple card transaction is sent to the bank's server. • IBM WebSphere MQ Telemetry identifies the one person from the thousands, alerting the customer that their card has been used. • IBM WebSphere MQ Telemetry can use the simplest input of information, and locate that individual.

What is WebSphere MQ Telemetry?

- It is a feature of IBM WebSphere MQ that extends the universal messaging backbone provided by IBM WebSphere MQ to a wide range of remote sensors, actuators and telemetry devices. IBM WebSphere MQ Telemetry extends IBM WebSphere MQ so that it can interconnect intelligent enterprise applications, services, and decision makers with networks of instrumented devices.

- The two core parts of WebSphere MQ Telemetry are:
  1. The IBM WebSphere MQ Telemetry service that runs inside of the IBM WebSphere MQ server.
  2. IBM WebSphere MQ Telemetry clients that are distributed to devices together with the applications.

What can it do for me?

- MQ Telemetry uses the MQ Telemetry Transport (MQTT) to send and receive data between your applications and the IBM WebSphere MQ Queue Manager.
- 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 might have limited resources.
- MQTT works efficiently on networks where the bandwidth might be 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 may be other telemetry applications, MQI, JMS or enterprise messaging applications.

How do I use it?

- Use the IBM WebSphere MQ Explorer and its associated tools to administer the WebSphere MQ Telemetry feature of MQ.
- Use MQTT clients in your applications to connect to a queue manager, publish and subscribe for messages.
- Distribute your application with the MQTT client to the device where your application is to run.

How does it work?

- The MQ Telemetry service turns an IBM WebSphere MQ queue manager into an MQTT server
- The MQTT server understands the MQTT message transport and can receive messages from and send messages to MQTT clients.
- MQ Telemetry ships with a number of Telemetry clients that implement the MQTT message transport. These are often referred to as MQTT clients.
- A basic Telemetry client works like a standard MQ client but can run on a much wider variety of platforms and networks.
- An Advanced Telemetry Client acts as a network concentrator to connect an even greater number of MQTT clients to a single queue manager. It can also provide store and forward for small devices that lack a means to buffer messages during short network outages.
- IBM WebSphere MQ Telemetry daemon for devices is an Advanced Telemetry client that is part of IBM WebSphere MQ Telemetry. See “Telemetry daemon for devices” on page 129 for more information.
- MQTT is a publish subscribe protocol:
  - An MQTT client application can publish messages to an MQTT server.
  - When an IBM WebSphere 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.
  - An MQTT client can subscribe for messages that are sent by applications that connect to an MQ queue manager.
  - 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.

IBM WebSphere MQ Telemetry replaces the SCADA nodes that were withdrawn in version 7 of IBM WebSphere Message Broker and runs on Windows, Linux, and AIX®. Migration of telemetry applications from using IBM WebSphere Message Broker version 6 to use IBM WebSphere MQ Telemetry and IBM WebSphere Message Broker version 7.0 (*WebSphere MQ V7.1 Installing Guide*) provides information to help you migrate applications from using the SCADA nodes in IBM WebSphere Message Broker V6. Telemetry applications using IBM WebSphere Message Broker version 7 subscribe to topics that are common to MQTT clients. They receive publications from MQTT clients using MQInput nodes and publish to MQTT clients using publication nodes.

Related concepts:

"Telemetry concepts and scenarios for monitoring and control" on page 103

Installing IBM WebSphere MQ Telemetry (*WebSphere MQ V7.1 Installing Guide*)

Administering IBM WebSphere MQ Telemetry (*WebSphere MQ V7.1 Administering Guide*)

Related tasks:

Migration of telemetry applications from using IBM WebSphere Message Broker version 6 to use IBM WebSphere MQ Telemetry and IBM WebSphere Message Broker version 7.0 (*WebSphere MQ V7.1 Installing Guide*)

Developing applications for IBM WebSphere MQ Telemetry (*WebSphere MQ V7.1 Programming Guide*)

Troubleshooting for IBM WebSphere MQ Telemetry (*WebSphere MQ V7.1 Administering Guide*)

Related reference:

IBM WebSphere MQ Telemetry Reference (*WebSphere MQ V7.1 Reference*)

Related information:

"IBM WebSphere MQ Telemetry" on page 100

http://www-01.ibm.com/software/integration/wmq/requirements/

IBM WebSphere MQ information roadmap

The information roadmap contains links to a variety of IBM WebSphere MQ resources.

This roadmap brings together information from different sources to help you find out more about a particular area of IBM WebSphere MQ. Click the links to each section in the roadmap to see what resources are available.

- Product overview
- Technical overview
- Planning
- Migrating and upgrading
- Installing
- Security
- Configuring
- Administering
- Developing applications
- Monitoring and performance
- Troubleshooting and support
- Reference

Table 1. IBM WebSphere MQ information roadmap table

Category	Information resources
Product overview	Overview of the overall purpose, capabilities, and new features of IBM WebSphere MQ. **“Product overview” on page 1** Introductory information to help you get started with IBM WebSphere MQ Version 7.1, including an introduction to the product and an overview of what is new and what is changed for this release. **WebSphere MQ Version 7.1** This IBM Redbooks publication covers the core enhancements made in IBM WebSphere MQ Version 7.1 and the concepts that must be understood. **WebSphere MQ product web page** Product web page with links to resources and additional information. **WebSphere MQ system requirements** Web page with links to the system requirements for the different releases of IBM WebSphere MQ. **WebSphere MQ documentation library page** Links to the product documentation in IBM Knowledge Center, downloadable versions of the product documentation, and PDF versions of the product documentation.
Technical overview	**“IBM WebSphere MQ Technical overview” on page 72** Information to help you to find out about message queuing and other features that IBM WebSphere MQ Version 7.1 provides.
Planning	**Planning (***WebSphere MQ V7.1 Installing Guide***)** When planning your IBM WebSphere MQ environment, consider the support that IBM WebSphere 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	**Migrating and upgrading (***WebSphere MQ V7.1 Installing Guide***)** To migrate a queue manager to run on a new level of code, you must first upgrade IBM WebSphere MQ to install the new code level. When you have verified the upgrade is successful, migrate the queue manager and all the applications and resources associated with it. Before starting this process, create a migration plan, based on the information in this migration guide. If you are applying maintenance, no migration is necessary. However you should test applications with the new level of IBM WebSphere MQ code **WebSphere MQ Migration Guide for distributed systems** This guide provides information to help you plan the process of migrating from an older version to a new version on distributed systems. You can either view the guide in your web browser or download it as a PDF file. **WebSphere 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 1. IBM WebSphere MQ information roadmap table (continued)

Category	Information resources
Installing	**Installing and uninstalling (***WebSphere MQ V7.1 Installing Guide***)** 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.
Security	**Security (***WebSphere MQ V7.1 Administering Guide***)** Aspects of security to consider in your IBM WebSphere MQ installation including identification and authentication, authorization, auditing, confidentiality, and data integrity.
Configuring	**Configuring (***WebSphere MQ V7.1 Installing Guide***)** 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.
Administering	**Administering IBM WebSphere MQ (***WebSphere MQ V7.1 Administering Guide***)** Administer your queue managers and associated resources.
Developing applications	**Developing applications (***WebSphere MQ V7.1 Programming Guide***)** Develop applications to send and receive messages, and to manage your queue managers and related resources. IBM WebSphere MQ support applications written in procedural languages, and object oriented languages and frameworks.
Monitoring and performance	**Monitoring and performance (***WebSphere MQ V7.1 Administering Guide***)** 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.
Troubleshooting and support	**Troubleshooting and support (***WebSphere MQ V7.1 Administering Guide***)** Techniques to help you diagnose and solve problems with your queue manager network or IBM WebSphere MQ applications. **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. **IBM Support Portal web page** IBM Support Portal for IBM WebSphere MQ.
Reference	**Reference (***WebSphere MQ V7.1 Reference***)** Reference information for configuration, administration, developing applications, telemetry, security, monitoring, troubleshooting and support, and diagnostic messages.

WebSphere MQ Version 7.1 PDF documentation

You can download the WebSphere MQ Version 7.1 documentation as a series of PDF files.

WebSphere MQ product documentation PDF files

Table 2. Mapping of PDF files to product documentation sections.

PDF file name and download link	Product documentation section
wmq71.overview.pdf	Product overview
wmq71.installconfig.pdf	Planning Installing Migrating and upgrading Configuring
wmq71.administer.pdf	Administering Security Monitoring and performance Troubleshooting and support
wmq71.develop.pdf	Developing applications
wmq71.reference.pdf	Reference
MQ_Migration_Guide.pdf	Migration guide for WebSphere MQ on distributed systems
WMQ_zOS_Migration.pdf	Migration Guide for WebSphere MQ for z/OS

Note: The PDF files must be in the same folder for links between PDF files to function correctly.

Related concepts:

"Mappings from the old IBM WebSphere MQ books to the new product documentation" on page 58

"IBM WebSphere MQ information roadmap" on page 5

What's new in IBM WebSphere MQ Version 7.1

Learn about the main new functions in IBM WebSphere MQ Version 7.1.

- "IBM WebSphere MQ Telemetry" on page 9
- "Channel access control" on page 9
- "Setting security on objects that do not exist on the local queue manager" on page 9
- "Relocatable Installations" on page 10
- "Coexistence on UNIX, Linux, and Windows systems" on page 10
- "Dead Letter Queue usage on channels and topics" on page 10
- "Remote Object Authorisation" on page 10
- "Dump MQ Configuration" on page 10
- "Federal Information Processing Standards (FIPS) 140-2 compliance for UNIX, Linux, and Windows" on page 10
- "Federal Information Processing Standards (FIPS) 140-2 compliance for z/OS" on page 11
- "More information about the subject and issuer distinguished name" on page 11
- "Increased CipherSpec and algorithm support" on page 11
- "NSA Suite B support" on page 11
- "Additional changes specific to IBM WebSphere MQ for z/OS" on page 11

- “Run multi-instance queue managers on a non-domain controller on Windows” on page 12

IBM WebSphere MQ Telemetry

People, businesses, and governments increasingly want to use telemetry to interact more smartly with the environment we live and work in. Telemetry connects all kinds of devices to the internet and to the enterprise, and reduces the costs of building applications for smart devices.

IBM WebSphere MQ Telemetry provides small client libraries that can be embedded into smart devices running on a number of different device platforms. Applications built with the clients use WebSphere MQ Telemetry Transport (MQTT) and the WebSphere MQ Telemetry service to publish and subscribe messages reliably with WebSphere MQ. For more information, see “Introduction to IBM WebSphere MQ Telemetry” on page 3.

IBM WebSphere MQ Telemetry is no longer a stand-alone feature, as it was in IBM WebSphere MQ Version 7.0.1. It is now an integrated feature of the main IBM WebSphere MQ IBM WebSphere MQ Version 7.1 product. Installation consists of selecting IBM WebSphere MQ Telemetry from the optional components list, instead of a separate installation. For more information, see WebSphere MQ Telemetry.

Channel access control

To exercise more precise control over the access granted to connecting systems at a channel level, you can use channel authentication records. Channel authentication records can be created to perform the following functions:

- To block connections from specific IP addresses.
- To block connections from specific user IDs.
- To set an MCAUSER value to be used for any channel connecting from a specific IP address.
- To set an MCAUSER value to be used for any channel asserting a specific user ID.
- To set an MCAUSER value to be used for any channel having a specific SSL or TLS Distinguished Name (DN).
- To set an MCAUSER value to be used for any channel connecting from a specific queue manager.
- To block connections claiming to be from a certain queue manager unless the connection is from a specific IP address.
- To block connections claiming to be from a certain client user ID unless the connection is from a specific IP address.
- To block connections presenting a certain SSL or TLS certificate unless the connection is from a specific IP address.

For more information see Channel authentication records (*WebSphere MQ V7.1 Administering Guide*).

Setting security on objects that do not exist on the local queue manager

In IBM WebSphere MQ for z/OS, by using RACF®, you can set authorities for objects that do not yet exist, or do not exist on the local queue manager. This has not been possible on other platforms.

In IBM WebSphere MQ Version 7.1, you can use the **`setmqaut`** command to set the security on an object irrespective of whether it currently exists on the local queue manager. You can also use the new object type `rqmname` in **`setmqaut`** to set authority for a remote queue manager. These enhancements allow you to set authority at a more granular level than previously, for example for individual cluster queues. For more information, see setmqaut (*WebSphere MQ V7.1 Reference*), Changing and revoking access to a WebSphere MQ object (*WebSphere MQ V7.1 Administering Guide*), and Authorizing putting messages on remote cluster queues (*WebSphere MQ V7.1 Administering Guide*).

Relocatable Installations

On AIX, HP-UX, Linux, and Solaris, it is now possible to install IBM WebSphere MQ to a location of your choice. For more information about how to do so, see Choosing your installation location (*WebSphere MQ V7.1 Installing Guide*) and Installing a server (*WebSphere MQ V7.1 Installing Guide*).

Coexistence on UNIX, Linux, and Windows systems

On UNIX, Linux, and Windows systems, it is now possible to have more than one installation of IBM WebSphere MQ on a single machine. A maximum of 128 copies of IBM WebSphere MQ can be installed on a system at a time. One installation can be an installation of IBM WebSphere MQ Version 7.0.1 at fix pack level 6, or later. For more information, see Multiple installations (*WebSphere MQ V7.1 Installing Guide*).

Dead Letter Queue usage on channels and topics

The Use Dead-Letter Queue (**USEDLQ**) attribute can be used to provide increased granular control over which channels and topics use the Dead-Letter queue when a message cannot be delivered. This attribute enables the configuration of selected channels to not use the Dead-Letter queue. **USEDLQ** also enables topics to be individually configured to determine whether the Dead-Letter queue is to be used for messages that cannot be delivered to subscribers. For more information, see Dead-Letter Queues and Topic usage notes.

Remote Object Authorisation

Remote Object Authorisation enables you to secure access to objects hosted on remote queue managers. An example of this type of object can be a remote queue that is directly addressed or a cluster queue residing on a remote queue manager. Remote Object Authorisation provides selective and granular access control to those objects residing remotely, by enabling the user to select specific named objects to be accessed or restricted on a remote queue manager. For more information, see Security (*WebSphere MQ V7.1 Installing Guide*).

Dump MQ Configuration

You can use the Dump MQ configuration command (**dmpmqcfg**) on UNIX, Linux, and Windows systems, and DMPMQMCFG on IBM i to dump the configuration of queue managers in various scripting formats including MQSC, setmqaut and GRTMQMAUT. The scripts produced by the Dump MQ Configuration command can be used to restore or rebuild a queue manager. If you have previously used the Category 2 SupportPac MS03 - Save Queue Manager, you can use this command instead.

Federal Information Processing Standards (FIPS) 140-2 compliance for UNIX, Linux, and Windows

When cryptography is required on an SSL or TLS channel on UNIX, Linux, and Windows systems, IBM WebSphere MQ uses a cryptography package called IBM Crypto for C (ICC). On the UNIX, Linux, and Windows platforms, the ICC software has passed the Federal Information Processing Standards (FIPS) Cryptomodule Validation Program of the US National Institute of Standards and Technology, at level 140-2. For more information, see Federal Information Processing Standards for UNIX, Linux. and Windows (*WebSphere MQ V7.1 Administering Guide*).

Federal Information Processing Standards (FIPS) 140-2 compliance for z/OS

When cryptography is required on an SSL or TLS channel on z/OS, IBM WebSphere MQ uses a service called System SSL. The objective of System SSL is to provide the capability to execute securely in a mode designed to adhere to the Federal Information Processing Standards (FIPS) Cryptomodule Validation Program of the US National Institute of Standards and Technology, at level 140-2. For more information, see Federal Information Processing Standards for z/OS (*WebSphere MQ V7.1 Administering Guide*).

Increased CipherSpec and algorithm support

The range and type of CipherSpecs and algorithms supported are increased in this release of IBM WebSphere MQ on z/OS, UNIX, Linux, and Windows platforms. On z/OS two new CipherSpecs have been introduced

UNIX, Linux, and Windows platforms now include Elliptic Curve cryptography which utilizes the mathematics of elliptic curves to form the basis of powerful encryption algorithms. As well as Elliptic Curve cryptography, this release of IBM WebSphere MQ also supports Secure Hash Algorithm - 2 (SHA-2) for use in CipherSpecs, this increased functionality provides a more robust level of data integrity. For more information, see CipherSpecs and CipherSuites (*WebSphere MQ V7.1 Administering Guide*) and GSKit: Digital certificate signature algorithms compliant with FIPS 140-2 (*WebSphere MQ V7.1 Reference*).

More information about the subject and issuer distinguished name

In IBM WebSphere MQ Version 7.1, support is added for a range of new security attributes. In order to access these attributes IBM WebSphere MQ Version 7.1 obtains the Distinguished Encoding Rules (DER) encoding of a certificate and uses it to determine the subject and issuer distinguished names.

The subject and issuer distinguished names appear in the `SSLPEER` and `SSLCERTI` fields. A `SERIALNUMBER` attribute is also included in the subject distinguished name and contains the serial number for the certificate of the remote partner.

These new values are returned by the channel status commands and in data passed to channel security exits; see Channel security exit programs (*WebSphere MQ V7.1 Programming Guide*).

NSA Suite B support

The US government produces technical advice on IT systems and security, including data encryption. The US National Security Agency (NSA) recommends a set of interoperable cryptographic algorithms in its Suite B standard. IBM WebSphere MQ supports Suite B on UNIX, Linux, and Windows platforms. For more information, see National Security Agency (NSA) Suite B Cryptography (*WebSphere MQ V7.1 Administering Guide*).

Additional changes specific to IBM WebSphere MQ for z/OS

You can choose to offload message data for shared queues from the coupling facility to either DB2®, or a IBM WebSphere MQ managed data set called a shared message data set (SMDS). See Planning your coupling facility and offload storage environment (*WebSphere MQ V7.1 Installing Guide*).

IBM WebSphere MQ Version 7.1 provides the level of GROUPUR support required to enable CICS® Transaction Server for z/OS (CICS TS) Version 4.2 to exploit group unit of recovery by way of the CICS

RESYNCMEMBER(GROUPRESYNC) configuration in an MQCONN resource definition. For more information, see “Unit of recovery disposition” on page 296.

On z/OS, performance monitoring has been changed so that the QESD DD statement has been added to thlqual.SCSQPROC(CSQ4SMFJ). to provide a data definition for printing SMDS statistics. For more information about performance and resource monitoring on z/OS, see Monitoring performance and resource usage (*WebSphere MQ V7.1 Administering Guide*).

A new DDNAME, CSQINPT, has been added to the queue manager startup jcl procedure. MQSC commands provided via this DDNAME are processed at the end of pub/sub startup.

Support for the x'3C' OTMA protocol messages, which report IMS health status, has been added to the WebSphere MQ-IMS Bridge. For more information see The IMS Bridge.

Resilience to coupling facility connectivity failures support is included in IBM WebSphere MQ for z/OS and is used to improve the availability of IBM WebSphere MQ. For more information about planning and implementing this function see “Resilience to coupling facility connectivity failures” on page 282 and “Managing Resilience to coupling facility connectivity failures” on page 283.

Additional functions, ANALYZE and SLOAD (selective load) have been added to The CSQUTIL utility (*WebSphere MQ V7.1 Administering Guide*). Using these functions, a dataset containing messages offloaded using the COPY or SCOPY functions can be analyzed and selected message data reloaded into the same, or a different queue. For more information see, Analyzing the queue data copied to a data set by COPY or SCOPY using ANALYZE (*WebSphere MQ V7.1 Reference*).

You can use the new MQSC command **ACTION(FAIL)** to make a CFSTRUCT appear to be failed. This command might be useful for testing purposes, or, to stop an in progress CFSTRUCT recovery initiated using RECOVER CFSTRUCT. or through automatic structure recovery introduced in this release. For more information see, RESET CFSTRUCT (*WebSphere MQ V7.1 Reference*).

On z/OS, the default PutApplName origin context field of the MQMD for IBM WebSphere MQ applications using the RRS adaptor is changed. In previous releases of IBM WebSphere MQ for z/OS, when a z/OS application puts a message on the queue, the PutApplName field in the MQMD is RRSBATCH. In IBM WebSphere MQ for z/OS Version 7.1 the PutApplName field in the MQMD is the jobname of the address space where the task is running. This change primarily affects users of IBM WebSphere Application Server, IBM IBM WebSphere Message Broker, and IBM Db2 stored procedures running on the z/OS platform. For more information see, PutApplName (MQCHAR28) (*WebSphere MQ V7.1 Reference*) and Overview for MQMD (*WebSphere MQ V7.1 Reference*).

Run multi-instance queue managers on a non-domain controller on Windows

In IBM WebSphere MQ for Windows Version 7.1, you can run multi-instance queue managers on Windows servers or workstations that are part of a domain. In Version 7.0.1, multi-instance queue managers had to run on domain controllers. For more information; see Create a multi-instance queue manager on domain workstations or servers (*WebSphere MQ V7.1 Installing Guide*).

Related concepts:
"What's new in IBM WebSphere MQ Version 7.1" on page 8
"What's changed in IBM WebSphere MQ Version 7.1" on page 17
"What's changed in IBM WebSphere MQ Version 7.1 Fix Packs" on page 55
Related information:

IBM WebSphere MQ requirements

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

IBM WebSphere MQ client for HP Integrity NonStop Server

IBM WebSphere MQ now supports the client for the HP Integrity NonStop Server platform.

Overview

For an overview of IBM WebSphere MQ clients, including the client for the HP Integrity NonStop Server platform, see "Overview of IBM WebSphere MQ MQI clients" on page 167.

For a technical overview of the IBM WebSphere MQ client for HP Integrity NonStop Server platform, see "IBM WebSphere MQ client for HP Integrity NonStop Server technical overview" on page 176.

For details of IBM WebSphere MQ client for HP Integrity NonStop Server supported environments and features, see "IBM WebSphere MQ client for HP Integrity NonStop Server supported environments and features" on page 177.

Planning

For help when you are planning your IBM WebSphere MQ client for HP Integrity NonStop Server environment, see Planning your IBM WebSphere MQ client environment on HP Integrity NonStop Server (*WebSphere MQ V7.1 Installing Guide*).

Installing

Help about installing the IBM WebSphere MQ client for HP Integrity NonStop Server.

- Choosing what to install, see IBM WebSphere MQ client components for HP Integrity NonStop Server (*WebSphere MQ V7.1 Installing Guide*).
- Planning your installation, see Planning your installation on HP Integrity NonStop Server (*WebSphere MQ V7.1 Installing Guide*)
  - File system (*WebSphere MQ V7.1 Installing Guide*)
- Hardware and software requirements, see Hardware and software requirements on HP Integrity NonStop Server systems (*WebSphere MQ V7.1 Installing Guide*).
- Verifying that you have the correct software, see Verifying system software prerequisites (*WebSphere MQ V7.1 Installing Guide*).
- Preparing your system, see Setting up the user and group on HP Integrity NonStop Server (*WebSphere MQ V7.1 Installing Guide*).
- Installing the client, see Installing IBM WebSphere MQ client on HP Integrity NonStop Server systems (*WebSphere MQ V7.1 Installing Guide*).
- Verifying your installation, see Verifying a client installation (*WebSphere MQ V7.1 Installing Guide*).

- Uninstalling, see Uninstalling IBM WebSphere MQ on HP Integrity NonStop Server (*WebSphere MQ V7.1 Installing Guide*).

HP Integrity NonStop Server client commands

The following commands are applicable to the IBM WebSphere MQ client for HP Integrity NonStop Server OSS and Guardian environments:

- dspmqver (*WebSphere MQ V7.1 Reference*)
- endmqtrc (*WebSphere MQ V7.1 Reference*)
- mqrc (*WebSphere MQ V7.1 Reference*)
- runmqras (*WebSphere MQ V7.1 Reference*)
- runmqtmc (*WebSphere MQ V7.1 Reference*)
- strmqtrc (*WebSphere MQ V7.1 Reference*)

The following command is applicable to the IBM WebSphere MQ client for HP Integrity NonStop Server OSS environment:

- dspmqtrc (*WebSphere MQ V7.1 Reference*)

New Product Identifier, MQNC, added to the DISPLAY CHSTATUS (*WebSphere MQ V7.1 Reference*) command, Product Identifier values table.

Security

To secure your IBM WebSphere MQ client for HP Integrity NonStop Server environment, see:

- Information about how the IBM WebSphere MQ client for HP Integrity NonStop Server identifies itself to the queue manager added to Planning authentication for a client application (*WebSphere MQ V7.1 Administering Guide*).
- Setting up security on HP Integrity NonStop Server (*WebSphere MQ V7.1 Administering Guide*)
  - OpenSSL (*WebSphere MQ V7.1 Administering Guide*)
  - Entropy Daemon (*WebSphere MQ V7.1 Administering Guide*)
- WebSphere MQ support for SSL and TLS (*WebSphere MQ V7.1 Administering Guide*)
- Working with SSL or TLS on HP Integrity NonStop Server (*WebSphere MQ V7.1 Administering Guide*)
  - Certificate management (*WebSphere MQ V7.1 Administering Guide*)
  - Personal certificate store (*WebSphere MQ V7.1 Administering Guide*)
  - Certificate trust store (*WebSphere MQ V7.1 Administering Guide*)
  - Pass phrase stash file (*WebSphere MQ V7.1 Administering Guide*)
  - Certificate revocation list file (*WebSphere MQ V7.1 Administering Guide*)

Transaction Management Facility

For information about the Transaction Management Facility (TMF), refer to the following sections and topics.

- Planning your IBM WebSphere MQ client environment on HP Integrity NonStop Server (*WebSphere MQ V7.1 Installing Guide*)
  - Preparing the HP Integrity NonStop Server environment (*WebSphere MQ V7.1 Installing Guide*)
  - IBM WebSphere MQ and HP NonStop TMF (*WebSphere MQ V7.1 Installing Guide*)
  - Using HP NonStop TMF (*WebSphere MQ V7.1 Installing Guide*)
    - Using global units of work (*WebSphere MQ V7.1 Installing Guide*)
    - Avoiding long running transactions (*WebSphere MQ V7.1 Installing Guide*)
    - 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 (*WebSphere MQ V7.1 Programming Guide*).
- Configuring HP Integrity NonStop Server (*WebSphere MQ V7.1 Installing Guide*)
  - Gateway process overview (*WebSphere MQ V7.1 Installing Guide*)
  - Configuring Gateway to run under Pathway (*WebSphere MQ V7.1 Installing Guide*)
  - TMF and TMF/Gateway stanzas (*WebSphere MQ V7.1 Installing Guide*)
  - Configuring the client initialization file (*WebSphere MQ V7.1 Installing Guide*)
  - Granting permissions to channels (*WebSphere MQ V7.1 Installing Guide*)
- Administering HP Integrity NonStop Server (*WebSphere MQ V7.1 Administering Guide*)
  - Manually starting the TMF/Gateway from Pathway (*WebSphere MQ V7.1 Administering Guide*)
  - Stopping the TMF/Gateway from Pathway (*WebSphere MQ V7.1 Administering Guide*)
- Troubleshooting IBM WebSphere MQ client for HP Integrity NonStop Server (*WebSphere MQ V7.1 Administering Guide*)

Developing applications

For information about developing applications for your IBM WebSphere MQ client on the HP Integrity NonStop Server platform, see:

- Building your application on HP Integrity NonStop Server (*WebSphere MQ V7.1 Programming Guide*)
  - OSS and Guardian headers and public libraries (*WebSphere MQ V7.1 Programming Guide*)
  - Preparing C programs in HP Integrity NonStop Server (*WebSphere MQ V7.1 Programming Guide*)
  - Preparing COBOL programs (*WebSphere MQ V7.1 Programming Guide*)
  - Preparing pTAL programs (*WebSphere MQ V7.1 Programming Guide*)
- For information about coding in pTAL, see Coding in pTAL (*WebSphere MQ V7.1 Programming Guide*).

- For information about preparing JMS programs for the IBM WebSphere MQ client for HP Integrity NonStop Server, see Preparing JMS programs for the IBM WebSphere MQ client for HP Integrity NonStop Server (*WebSphere MQ V7.1 Programming Guide*).

New messages

The following are new messages for the IBM WebSphere MQ client on HP Integrity NonStop Server:

- AMQ5000-5999: Installable services
  - AMQ5370
  - AMQ5371
  - AMQ5372
  - AMQ5373
  - AMQ5374
  - AMQ5375
  - AMQ5376
  - AMQ5377
  - AMQ5378
  - AMQ5379
  - AMQ5380
  - AMQ5390
  - AMQ5391
  - AMQ5392
  - AMQ5393
  - AMQ5394
  - AMQ5395
  - AMQ5396
  - AMQ5397
  - AMQ5398
  - AMQ5399
- AMQ9000-9999: 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 (*WebSphere MQ V7.1 Administering Guide*)
- 2355 (0933) (RC2355): MQRC_UOW_MIX_NOT_SUPPORTED (*WebSphere MQ V7.1 Administering Guide*)
- 2072 (0818) (RC2072): MQRC_SYNCPOINT_NOT_AVAILABLE (*WebSphere MQ V7.1 Administering Guide*)
- 2003 (07D3) (RC2003): MQRC_BACKED_OUT (*WebSphere MQ V7.1 Administering Guide*)

Samples

For information about the techniques demonstrated by the sample programs for the IBM WebSphere MQ client on HP Integrity NonStop Server, see Samples for IBM WebSphere MQ client for HP Integrity NonStop Server (*WebSphere MQ V7.1 Programming Guide*).

Troubleshooting and support

For troubleshooting and support information for the IBM WebSphere MQ client on HP Integrity NonStop Server, see the following topics:

- Troubleshooting IBM WebSphere MQ client for HP Integrity NonStop Server (*WebSphere MQ V7.1 Administering Guide*)
- Error logs on HP Integrity NonStop Server (*WebSphere MQ V7.1 Administering Guide*)
- Using trace on HP Integrity NonStop Server (*WebSphere MQ V7.1 Administering Guide*)
- FFST: WebSphere MQ for HP Integrity NonStop Server (*WebSphere MQ V7.1 Administering Guide*)

What's changed in IBM WebSphere MQ Version 7.1

Review the list of changes carefully before upgrading queue managers to IBM WebSphere MQ Version 7.1. Decide whether you must plan to make changes to existing applications, scripts, and procedures before starting to migrate systems to Version 7.1.

The following links are to information within the Migrating and upgrading section of the product documentation. New functions, and changes that do not affect existing applications, administrative procedures, and administrative scripts are not listed here; see "What's new in IBM WebSphere MQ Version 7.1" on page 8.

Changes that affect migration (*WebSphere MQ V7.1 Installing Guide*)

- Changes in IBM WebSphere MQ Version 7.1 (*WebSphere MQ V7.1 Installing Guide*)

Related concepts:

"What's new in IBM WebSphere MQ Version 7.1" on page 8

"What's changed in IBM WebSphere MQ Version 7.1 Fix Packs" on page 55

Related information:

IBM WebSphere MQ requirements

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

Behavior that has changed between Version 7.0.1 to Version 7.1

Between Version 7.0.1 and Version 7.1 some aspects of IBM WebSphere MQ have changed that might affect existing applications, administrative scripts, or management procedures.

The following list of changes, are copied from the migration guide; Migrating and upgrading WebSphere MQ (*WebSphere MQ V7.1 Installing Guide*). They might affect the operation of existing applications, administrative scripts, and management procedures. New functions, and changes that do not affect existing applications, administrative procedures, and administrative scripts are not listed here; see What's new in IBM WebSphere MQ Version 7.1.

Review the list of changes carefully before upgrading queue managers to Version 7.1. Decide whether you must plan to make changes to existing applications, scripts, and procedures before starting to migrate systems to IBM WebSphere MQ Version 7.1.

Related tasks:

Planning IBM WebSphere MQ Version 7.0.1 to IBM WebSphere MQ Version 7.1 migration by platform (*WebSphere MQ V7.1 Installing Guide*)

New features and their supported APIs for Version 7.0.1

Use this information to learn about the WebSphere MQ Version 7.0.1 APIs with features and environments that might not be as fully supported as the C MQI.

The following topics contain tables that show supported WebSphere MQ Version 7.0.1 APIs. Use the information in these tables in conjunction with the information in the system requirements web pages.

For links to system requirements information for all releases of WebSphere MQ, see the System Requirements web page.

New features and their supported APIs for Version 7.0.1: JMS:

Use this information to learn about the WebSphere MQ Version 7.0.1 APIs with features and environments that might not be as fully supported as the C MQI.

JMS

The following table shows which features are supported for JMS.

	JMS										
	Distributed platforms				z/OS						
API	Basic[1]	WAS and other Java™ EE[2]	Other TM	Db2 SP[3]	Basic	WAS[2]	Other JavaEE	Db2 SP[3]	Other TM	CICS	IMS
V7 Publish Subscribe	Y	Y	Y	N/A[4]	N	N	N/A	N/A[4]		N/A	N/A
Browse with Mark	N	N	N	N/A[4]	Y	Y	N/A	N/A[4]		N/A	N/A
Message Properties	Y[5]	Y[5]	Y[5]	N/A[4]	Y[5]	Y[5]	N/A	N/A[4]	Y[5]	N/A	N/A
Message Selectors	Y	Y	Y	N/A[4]			N/A	N/A[4]		N/A	N/A
Asynchronous consume	Y	Y	Y	N/A[4]			N/A	N/A[4]		N/A	N/A
Content filtering Publish/Subscribe	N	N	N	N/A[4]	N	N	N/A	N/A[4]	N	N/A	N/A
Group/Segment messages	N	N	N	N/A[4]	N	N	N/A	N/A[4]	N	N/A	N/A
PCF Classes	Y[6]	Y[6]	Y[6]	N/A[4]	Y[6]	Y[6]	N/A	N/A[4]	Y[6]	N/A	N/A
Global TX Participant	Y[7]	Y	Y[7]	N/A[4]	Y[8]	Y	N/A	N/A[4]	Y[8]	N/A	N/A
Global TX Coordinator	N	N	N	N/A[4]	N	N	N/A	N/A	N	N/A	N/A

Note:

1. Also OSGI.
2. WebSphere Application Server.
3. Db2 Stored Procedures.
4. Db2 SP Distributed: cannot work reliably because of the restricted Java environment.
5. Cannot see the whole MQI namespace, therefore an MQI application might generate messages whose properties cannot be read.
6. Can build/parse messages but not use the MessageAgent classes & methods.
7. Open source JTA coordinators might not be supported.
8. RRS is the coordinator.

	JMS										
	Distributed platforms				z/OS						
Client communication (not running on top of C client)	**Basic[1]**	**WAS and other Java EE[2]**	**Other TM**	**Db2 SP[3]**	**Basic**	**WAS[2]**	**Other Java EE**	**Db2 SP[3]**	**Other TM**	**CICS**	**IMS**
Read ahead	Y	Y	Y	N/A	N/A	Y	N/A	N/A	N/A	N/A	N/A
Asynchronous put	Y	Y	Y	N/A	N/A	Y	N/A	N/A	N/A	N/A	N/A
Pre-connect exit	N	N	N	N/A	N/A	N	N/A	N/A	N/A	N/A	N/A
CCDT	Y	Y	Y	N/A	N/A	Y	N/A	N/A	N/A	N/A	N/A
Multiple CONNAME	Y	Y	Y	N/A	N/A	Y	N/A	N/A	N/A	N/A	N/A
Auto reconnect	Y	Y[5]	Y	N/A	N/A	Y[5]	N/A	N/A	N/A	N/A	N/A
Channel compression	Y	Y	Y	N/A	N/A	Y	N/A	N/A	N/A	N/A	N/A
Exit chaining	Y	Y	Y	N/A	N/A	Y	N/A	N/A	N/A	N/A	N/A
MQCSP/ MQXR_SEC_PARMS	Y[4]	Y[4]	Y[4]	N/A	N/A	Y[4]	N/A	N/A	N/A	N/A	N/A
Native SSL	Y	Y	Y	N/A	N/A	Y	N/A	N/A	N/A	N/A	N/A

Note:

1. Also OSGI.
2. WebSphere Application Server.
3. Db2 Stored Procedures.
4. Not consistent across all clients.
5. Use RA features for reconnect, not the FAP-level operation.

Related reference:

Running IBM WebSphere MQ classes for Java applications within Java platform Enterprise Edition

Related information:

WebSphere MQ resource adapter V7.1 and later statement of support

New features and their supported APIs for Version 7.0.1: Java:

Use this information to learn about the WebSphere MQ Version 7.0.1 APIs with features and environments that might not be as fully supported as the C MQI.

Java

The following table shows which features are supported for Java.

	Base Java									
	Distributed platforms				z/OS					
API	Basic[1]	WAS and other Java EE[2]	Other TM	Db2 SP[3]	Basic	WAS and other Java EE[2]	Db2 SP[3]	Other TM	CICS	IMS
V7 Publish Subscribe	Y	N/A	N/A	N/A[6]	Y	N/A	N/A[6]	N/A	Y	N/A
Browse with Mark	Y	N/A	N/A	N/A[6]	Y	N/A	N/A[6]	N/A	Y	N/A
Message Properties	N	N/A	N/A	N/A[6]	N	N/A	N/A[6]	N/A	N	N/A
Message Selectors	Y	N/A	N/A	N/A[6]	N	N/A	N/A[6]	N/A	N	N/A
Asynchronous consume	N	N/A	N/A	N/A[6]	N	N/A	N/A[6]	N/A	N	N/A
Content filtering Publish/ Subscribe	N	N/A	N/A	N/A[6]	N	N/A	N/A[6]	N/A	N	N/A
Group/ Segment messages	Y	N/A	N/A	N/A[6]	Y	N/A	N/A[6]	N/A	Y	N/A
PCF Classes	Y	N/A	N/A	N/A[6]	Y	N/A	N/A[6]	N/A	Y	N/A
Global TX Participant	N	N/A	N/A	N/A[6]	N	N/A	N/A[6]	N/A	N	N/A
Global TX Coordinator	Y	N/A	N/A	N/A[6]	N/A[5]	N/A	N/A	N/A	N/A	N/A

Note:

1. Also OSGI.
2. WebSphere Application Server.
3. Db2 Stored Procedures.
4. While it might work (some customers have used this environment), it is not recommended or fully supported.
5. RRS is the coordinator.
6. Db2 SP on Distributed platform cannot work reliably because of the restricted Java environment.

	Base Java									
	Distributed platforms				z/OS					
Client communication (not running on top of C client)	Basic[1]	WAS and other Java EE[2]	Other TM	Db2 SP[3]	Basic	WAS and other Java EE[2]	Db2 SP[3]	Other TM	CICS	IMS
Read ahead	Y	N/A	N/A	N/A	N/A[5]	N/A	N/A	N/A	N/A	N/A
Asynchronous put	Y	N/A	N/A	N/A	N/A[5]	N/A	N/A	N/A	N/A	N/A
Pre-connect exit	N	N/A	N/A	N/A	N/A[5]	N/A	N/A	N/A	N/A	N/A
CCDT	Y	N/A	N/A	N/A	N/A[5]	N/A	N/A	N/A	N/A	N/A
Multiple CONNAME	N	N/A	N/A	N/A	N/A[5]	N/A	N/A	N/A	N/A	N/A
Auto reconnect	N	N/A	N/A	N/A	N/A[5]	N/A	N/A	N/A	N/A	N/A
Channel compression	Y	N/A	N/A	N/A	N/A[5]	N/A	N/A	N/A	N/A	N/A
Exit chaining	Y	N/A	N/A	N/A	N/A[5]	N/A	N/A	N/A	N/A	N/A
MQCSP/ MQXR_SEC_PARMS	Y[4]	N/A	N/A	N/A	N/A[5]	N/A	N/A	N/A	N/A	N/A
Native SSL	Y	N/A	N/A	N/A	N/A[5]	N/A	N/A	N/A	N/A	N/A

Note:

1. Also OSGI.
2. WebSphere Application Server.
3. Db2 Stored Procedures.
4. Not consistent across all clients.
5. Java client not supported in any environment on z/OS

Related reference:

Running IBM WebSphere MQ classes for Java applications within Java platform Enterprise Edition

Related information:

WebSphere MQ resource adapter V7.1 and later statement of support

New features and their supported APIs for Version 7.0.1: XMS .NET:

Use this information to learn about the WebSphere MQ Version 7.0.1 APIs with features and environments that might not be as fully supported as the C MQI.

XMS .NET

The following table shows which features are supported for XMS .NET.

	XMS .NET				
	Distributed platforms				
API	Basic[1]	WAS[2]	Other JEE	Other TM	Db2 SP[3]
V7 Publish Subscribe	Y	N/A	N/A	N/A	N/A
Browse with Mark	N	N/A	N/A	N/A	N/A
Message Properties	Y[4]	N/A	N/A	N/A	N/A
Message Selectors	Y	N/A	N/A	N/A	N/A
Asynchronous consume	Y	N/A	N/A	N/A	N/A
Content filtering Publish/Subscribe	N	N/A	N/A	N/A	N/A
Group/Segment messages	N	N/A	N/A	N/A	N/A
PCF Classes	N	N/A	N/A	N/A	N/A
Global TX Participant	Y[5]	N/A	N/A	N/A	N/A
Global TX Coordinator	N	N/A	N/A	N/A	N/A

Note:

1. Also OSGI.
2. WebSphere Application Server.
3. Db2 Stored Procedures.
4. Cannot see the whole MQI namespace, therefore an MQI application might generate messages whose properties cannot be read.
5. Works with Microsoft DTC only.

	XMS .NET				
	Distributed				
Client communication (not running on top of C client)	Basic[1]	WAS[2]	Other JEE	Other TM	Db2 SP[3]
Read ahead	Y	N/A	N/A	N/A	N/A
Asynchronous put	Y	N/A	N/A	N/A	N/A
Pre-connect exit	N	N/A	N/A	N/A	N/A
CCDT	Y	N/A	N/A	N/A	N/A
Multiple CONNAME	N	N/A	N/A	N/A	N/A
Auto reconnect	N	N/A	N/A	N/A	N/A
Channel compression	N	N/A	N/A	N/A	N/A
Exit chaining	N	N/A	N/A	N/A	N/A

	XMS .NET				
	Distributed				
Client communication (not running on top of C client)	**Basic[1]**	**WAS[2]**	**Other JEE**	**Other TM**	**Db2 SP[3]**
MQCSP/ MQXR_SEC_PARMS	N	N/A	N/A	N/A	N/A
Native SSL	N	N/A	N/A	N/A	N/A

Note:

1. Also OSGI.
2. WebSphere Application Server.
3. Db2 Stored Procedures.

New features and their supported APIs for Version 7.0.1: XMS C & C++:

Use this information to learn about the WebSphere MQ Version 7.0.1 APIs with features and environments that might not be as fully supported as the C MQI.

XMS C & C++

The following table shows which features are supported for XMS C & C++.

	XMS C & C++				
	Distributed				
API	**Basic[1]**	**WAS[2]**	**Other JEE**	**Other TM**	**Db2 SP[3]**
V7 Publish Subscribe	Y	N/A	N/A	N/A	N/A
Browse with Mark	N	N/A	N/A	N/A	N/A
Message Properties	Y	N/A	N/A	N/A	N/A
Message Selectors	Y	N/A	N/A	N/A	N/A
Asynchronous consume	Y	N/A	N/A	N/A	N/A
Content filtering Publish/Subscribe	N	N/A	N/A	N/A	N/A
Group/Segment messages	N	N/A	N/A	N/A	N/A
PCF Classes	N	N/A	N/A	N/A	N/A
Global TX Participant	N	N/A	N/A	N/A	N/A
Global TX Coordinator	N	N/A	N/A	N/A	N/A

Note:

1. Also OSGI.
2. WebSphere Application Server.

3. Db2 Stored Procedures.

	XMS C & C++				
	Distributed				
Client communication (not running on top of C client)	Basic[1]	WAS[2]	Other JEE	Other TM	Db2 SP[3]
Read ahead	Y	N/A	N/A	N/A	N/A
Asynchronous put	Y	N/A	N/A	N/A	N/A
Pre-connect exit	N	N/A	N/A	N/A	N/A
CCDT	N	N/A	N/A	N/A	N/A
Multiple CONNAME	N	N/A	N/A	N/A	N/A
Auto reconnect	N	N/A	N/A	N/A	N/A
Channel compression	Y	N/A	N/A	N/A	N/A
Exit chaining	Y	N/A	N/A	N/A	N/A
MQCSP/ MQXR_SEC_PARMS	N/A	N/A	N/A	N/A	N/A
Native SSL	Y	N/A	N/A	N/A	N/A

Note:

1. Also OSGI.
2. WebSphere Application Server.
3. Db2 Stored Procedures.

New features and their supported APIs for Version 7.0.1: .NET:

Use this information to learn about the WebSphere MQ Version 7.0.1 APIs with features and environments that might not be as fully supported as the C MQI.

.NET

The following table shows which features are supported for .NET.

	.NET				
	Distributed				
API	Basic[1]	WAS[2]	Other JEE	Other TM	Db2 SP[3]
V7 Publish Subscribe	Y	N/A	N/A	N/A	N/A
Browse with Mark	Y	N/A	N/A	N/A	N/A
Message Properties	Y[4]	N/A	N/A	N/A	N/A
Message Selectors	N	N/A	N/A	N/A	N/A
Asynchronous consume	N	N/A	N/A	N/A	N/A

	.NET				
	Distributed				
API	Basic[1]	WAS[2]	Other JEE	Other TM	Db2 SP[3]
Content filtering Publish/Subscribe	N	N/A	N/A	N/A	N/A
Group/Segment messages	N	N/A	N/A	N/A	N/A
PCF Classes	N	N/A	N/A	N/A	N/A
Global TX Participant	Y[5]	N/A	N/A	N/A	N/A
Global TX Coordinator	N	N/A	N/A	N/A	N/A

Note:

1. Also OSGI.
2. WebSphere Application Server.
3. Db2 Stored Procedures.
4. Cannot see the whole MQI namespace, therefore an MQI application might generate messages whose properties cannot be read.
5. Works with Microsoft DTC only.

	.NET				
	Distributed				
Client communication (not running on top of C client)	Basic[1]	WAS[2]	Other JEE	Other TM	Db2 SP[3]
Read ahead	Y	N/A	N/A	N/A	N/A
Asynchronous put	Y	N/A	N/A	N/A	N/A
Pre-connect exit	N	N/A	N/A	N/A	N/A
CCDT	Y	N/A	N/A	N/A	N/A
Multiple CONNAME	N	N/A	N/A	N/A	N/A
Auto reconnect	N	N/A	N/A	N/A	N/A
Channel compression	N	N/A	N/A	N/A	N/A
Exit chaining	N	N/A	N/A	N/A	N/A
MQCSP/ MQXR_SEC_PARMS	N	N/A	N/A	N/A	N/A
Native SSL	N[4]	N/A	N/A	N/A	N/A

Note:

1. Also OSGI.
2. WebSphere Application Server.
3. Db2 Stored Procedures.

4. Support available in Unmanaged mode only.

Channel authentication

From IBM WebSphere MQ Version 7.1, when you migrate a queue manager from an earlier release, channel authentication using channel authentication records is disabled. Channels continue to work as before. If you create a queue manager the later release to which you are migrating, channel authentication using channel authentication records is enabled, but with minimal additional checking. Some channels might fail to start.

Migrated queue managers

Channel authentication is disabled for migrated queue managers.

To start using channel authentication records you must run this MQSC command:

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

New queue managers

Channel authentication is enabled for new queue managers.

You want to connect existing queue managers or IBM WebSphere MQ MQI client applications to a newly created queue manager. Most connections work without specifying any channel authentication records. The following exceptions are to prevent privileged access to the queue manager, and access to system channels.

1. Privileged user IDs asserted by a client-connection channel are blocked by means of the special value *MQADMIN.

   ```
   SET CHLAUTH('*') TYPE(BLOCKUSER) USERLIST('*MQADMIN') +
   DESCR('Default rule to disallow privileged users')
   ```
2. Except for the channel used by IBM WebSphere MQ Explorer, all SYSTEM.* channels are blocked.

   ```
   SET CHLAUTH('SYSTEM.*') TYPE(ADDRESSMAP) ADDRESS('*') USERSRC(NOACCESS) +
   DESCR('Default rule to disable all SYSTEM channels')

   SET CHLAUTH(SYSTEM.ADMIN.SVRCONN) TYPE(ADDRESSMAP) ADDRESS('*') +
   USERSRC(CHANNEL) DESCR('Default rule to allow MQ Explorer access')
   ```

Note: This behaviour is default for all new IBM WebSphere MQ Version 7.1 queue managers on startup.

If you must work around the exceptions, you can run an MQSC command to add in more rules to allow channels blocked by the default rules to connect, or disable channel authentication checking:

```
ALTER QMGR CHLAUTH(DISABLED)
```

Related concepts:

Channel authentication records (*WebSphere MQ V7.1 Administering Guide*)

Changes to cluster error recovery on servers other than z/OS

Before Version 7.1, if a queue manager detected a problem with the local repository manager managing a cluster, it updated the error log. In some cases, it then stopped managing clusters. The queue manager continued to exchange applications messages with a cluster, relying on its increasingly out of date cache of cluster definitions. From Version 7.1 onwards, the queue manager reruns operations that caused problems, until the problems are resolved. If, after five days, the problems are not resolved, the queue manager shuts down to prevent the cache becoming more out of date. As the cache becomes more out of date, it causes a greater number of problems. The changed behavior regarding cluster errors in Version 7.1 does not apply to z/OS.

Every aspect of cluster management is handled for a queue manager by the local repository manager process, `amqrrmfa`. The process runs on all queue managers, even if there are no cluster definitions.

Before Version 7.1, if the queue manager detected a problem in the local repository manager, it stopped the repository manager after a short interval. The queue manager kept running, processing application messages and requests to open queues, and publish or subscribe to topics.

With the repository manager stopped, the cache of cluster definitions available to the queue manager became more out of date. Over time, messages were routed to the wrong destination, and applications failed. Applications failed attempting to open cluster queues or publication topics that had not been propagated to the local queue manager.

Unless an administrator checked for repository messages in the error log, the administrator might not realize the cluster configuration had problems. If the failure was not recognized over an even longer time, and the queue manager did not renew its cluster membership, even more problems occurred. The instability affected all queue managers in the cluster, and the cluster appeared unstable.

From Version 7.1 onwards, IBM WebSphere MQ takes a different approach to cluster error handling. Rather than stop the repository manager and keep going without it, the repository manager reruns failed operations. If the queue manager detects a problem with the repository manager, it follows one of two courses of action.

1. If the error does not compromise the operation of the queue manager, the queue manager writes a message to the error log. It reruns the failed operation every 10 minutes until the operation succeeds. By default, you have five days to deal with the error; failing which, the queue manager writes a message to the error log, and shuts down. You can postpone the five day shutdown.
2. If the error compromises the operation of the queue manager, the queue manager writes a message to the error log, and shuts down immediately.

An error that compromises the operation of the queue manager is an error that the queue manager has not been able to diagnose, or an error that might have unforeseeable consequences. This type of error often results in the queue manager writing an FFST file. Errors that compromise the operation of the queue manager might be caused by a bug in IBM WebSphere MQ, or by an administrator, or a program, doing something unexpected, such as ending a IBM WebSphere MQ process.

The point of the change in error recovery behavior is to limit the time the queue manager continues to run with a growing number of inconsistent cluster definitions. As the number of inconsistencies in cluster definitions grows, the chance of abnormal application behavior grows with it.

The default choice of shutting down the queue manager after five days is a compromise between limiting the number of inconsistencies and keeping the queue manager available until the problems are detected and resolved.

You can extend the time before the queue manager shuts down indefinitely, while you fix the problem or wait for a planned queue manager shutdown. The five-day stay keeps the queue manager running through a long weekend, giving you time to react to any problems or prolong the time before restarting the queue manager.

Corrective actions

You have a choice of actions to deal with the problems of cluster error recovery. The first choice is to monitor and fix the problem, the second to monitor and postpone fixing the problem, and the final choice is to continue to manage cluster error recovery as in releases before Version 7.1.

1. Monitor the queue manager error log for the error messages `AMQ9448` and `AMQ5008`, and fix the problem.

AMQ9448 indicates that the repository manager has returned an error after running a command. This error marks the start of trying the command again every 10 minutes, and eventually stopping the queue manager after five days, unless you postpone the shutdown.

AMQ5008 indicates that the queue manager was stopped because a IBM WebSphere MQ process is missing. AMQ5008 results from the repository manager stopping after five days. If the repository manager stops, the queue manager stops.

2. Monitor the queue manager error log for the error message AMQ9448, and postpone fixing the problem.

   If you disable getting messages from SYSTEM.CLUSTER.COMMAND.QUEUE, the repository manager stops trying to run commands, and continues indefinitely without processing any work. However, any handles that the repository manager holds to queues are released. Because the repository manager does not stop, the queue manager is not stopped after five days.

   Run an MQSC command to disable getting messages from SYSTEM.CLUSTER.COMMAND.QUEUE:

   ```
   ALTER QLOCAL(SYSTEM.CLUSTER.COMMAND.QUEUE) GET(DISABLED)
   ```

   To resume receiving messages from SYSTEM.CLUSTER.COMMAND.QUEUE run an MQSC command:

   ```
   ALTER QLOCAL(SYSTEM.CLUSTER.COMMAND.QUEUE) GET(ENABLED)
   ```

3. Revert the queue manager to the same cluster error recovery behavior as before Version 7.1.

   You can set a queue manager tuning parameter to keep the queue manager running if the repository manager stops.

   The tuning parameter is TolerateRepositoryFailure, in the TuningParameter stanza of the qm.ini file. To prevent the queue manager stopping, if the repository manager stops, set TolerateRepositoryFailure to TRUE; see Figure 1.

   Restart the queue manager to enable the TolerateRepositoryFailure option.

   If a cluster error has occurred that prevents the repository manager starting successfully, and hence the queue manager from starting, set TolerateRepositoryFailure to TRUE to start the queue manager without the repository manager.

Special consideration

Before Version 7.1, some administrators managing queue managers that were not part of a cluster stopped the amqrrmfa process. Stopping amqrrmfa did not affect the queue manager.

Stopping amqrrmfa in Version 7.1 causes the queue manager to stop, because it is regarded as a queue manager failure. You must not stop the amqrrmfa process in Version 7.1, unless you set the queue manager tuning parameter, TolerateRepositoryFailure.

Example

```
TuningParameters:
    TolerateRepositoryFailure=TRUE
```

Figure 1. Set TolerateRepositoryFailure to TRUE in qm.ini

Related concepts:

Queue manager configuration files, qm.ini (*WebSphere MQ V7.1 Installing Guide*)

Related information:

AMQ9448

Change in behavior of MQS_REPORT_NOAUTH

In IBM WebSphere MQ Version 7.1, the default behavior of MQS_REPORT_NOAUTH has been changed to *TRUE*.

This change causes UNIX platforms to behave like Windows, and log authorization failures to the error log.

Prior to IBM WebSphere MQ Version 7.1, this would only happen if you set MQS_REPORT_NOAUTH.

For more information, see MQS_REPORT_NOAUTH.

Connect to multiple queue managers and use `MQCNO_FASTPATH_BINDING`

Applications that connect to queue managers using the `MQCNO_FASTPATH_BINDING` binding option might fail with an error and reason code `MQRC_FASTPATH_NOT_AVAILABLE`.

An application can connect to multiple queue managers from the same process. In releases earlier than Version 7.1, an application can set any one of the connections to `MQCNO_FASTPATH_BINDING`. In Version 7.1, only the first connection can be set to `MQCNO_FASTPATH_BINDING`. See "Fast path" for the complete set of rules.

To assist with migration, you can set a new environment variable, `AMQ_SINGLE_INSTALLATION`. The variable reinstates the same behavior as in earlier releases, but prevents an application connecting to queue managers associated with other installations in the same process.

Fast path

On a server with multiple installations, applications using a fast path connection to IBM WebSphere MQ Version 7.1 or later must follow these rules:

1. The queue manager must be associated with the same installation as the one from which the application loaded the IBM WebSphere MQ run time libraries. The application must not use a fast path connection to a queue manager associated with a different installation. An attempt to make the connection results in an error, and reason code `MQRC_INSTALLATION_MISMATCH`.
2. Connecting non-fast path to a queue manager associated with the same installation as the one from which the application has loaded the IBM WebSphere MQ run time libraries prevents the application connecting fast path, unless either of these conditions are true:
   - The application makes its first connection to a queue manager associated with the same installation a fast path connection.
   - The environment variable, `AMQ_SINGLE_INSTALLATION` is set.
3. Connecting non-fast path to a queue manager associated with a different Version 7.1 or later installation, has no effect on whether an application can connect fast path.
4. You cannot combine connecting to a queue manager associated with a Version 7.0.1 installation and connecting fast path to a queue manager associated with a Version 7.1, or later installation.

With `AMQ_SINGLE_INSTALLATION` set, you can make any connection to a queue manager a fast path connection. Otherwise almost the same restrictions apply:

- The installation must be the same one from which the IBM WebSphere MQ run time libraries were loaded.

- Every connection on the same process must be to the same installation. If you attempt to connect to a queue manager associated with a different installation, the connection fails with reason code `MQRC_INSTALLATION_MISMATCH`. Note that with `AMQ_SINGLE_INSTALLATION` set, this restriction applies to all connections, not only fast path connections.
- Only connect one queue manager with fast path connections.

Related reference:

2587 (0A1B) (RC2587): MQRC_HMSG_NOT_AVAILABLE (*WebSphere MQ V7.1 Administering Guide*)

2590 (0A1E) (RC2590): MQRC_FASTPATH_NOT_AVAILABLE (*WebSphere MQ V7.1 Administering Guide*)

Related information:

Binding options

Custom scripts

In IBM WebSphere MQ for Windows Version 7.1, custom scripts that have been used in an earlier release to install packages might fail if any packages have been renamed, removed, or added in the later release to which you are migrating.

Custom scripts to install IBM WebSphere MQ can be incomplete as they have been added or removed.

For a detailed explanation of each component, see Choosing what to install (*WebSphere MQ V7.1 Installing Guide*) and select your platform.

Changes to data types

A number of data types have changed between IBM WebSphere MQ Version 7.0.1 to IBM WebSphere MQ Version 7.1 and new data types have been added. This topic lists the changes for data types that have a new current version in Version 7.1.

The current version of a data type is incremented if the length of a data type is extended by adding new fields. The addition of new constants to the values that can be set in a data type does not result in a change to the current version value.

Table 3 lists the data types that have new versions. Click on the links to read about the new fields.

Table 3. New fields added to existing data types

Data type	New version	New fields
Channel definition	`MQCD_VERSION_10`	BatchDataLimit (MQLONG) DefReconnect (MQLONG) UseDLQ (MQLONG) (*WebSphere MQ V7.1 Reference*)
Channel exit	`MQCXP_VERSION_8`	MCAUserSource (MQLONG) (*WebSphere MQ V7.1 Reference*) pEntryPoints (PMQIEP) (*WebSphere MQ V7.1 Reference*)
Data conversion exit	`MQDXP_VERSION_2`	pEntryPoints (PMQIEP) (*WebSphere MQ V7.1 Reference*)
Pre-connect exit	`MQNXP_VERSION_2`	pEntryPoints (PMQIEP) (*WebSphere MQ V7.1 Reference*)
Publish exit publication context	`MQPBC_VERSION_2`	MQPSXP - Publish exit data structure (*WebSphere MQ V7.1 Reference*)
Publish exit	`MQPSXP_VERSION_2`	pEntryPoints (PMQIEP)

Table 3. New fields added to existing data types (continued)

Data type	New version	New fields
Cluster workload exit	`MQWXP_VERSION_4`	Fields in MQWXP - Cluster workload exit parameter structure (*WebSphere MQ V7.1 Reference*)
SSL configurations options	`MQSCO_VERSION_3`	EncryptionPolicySuiteB(MQLONG) (*WebSphere MQ V7.1 Reference*)
SSL configurations options	`MQSCO_VERSION_4`	CertificateValPolicy (MQLONG) (*WebSphere MQ V7.1 Reference*)

Default transmission queue restriction

The product documentation in previous versions of IBM WebSphere MQ warned about defining the default transmission queue as `SYSTEM.CLUSTER.TRANSMIT.QUEUE`. In Version 7.1, any attempt to set or use a default transmission queue that is defined as `SYSTEM.CLUSTER.TRANSMIT.QUEUE` results in an error.

In earlier versions of IBM WebSphere MQ no error was reported when defining the default transmission queue as `SYSTEM.CLUSTER.TRANSMIT.QUEUE`. MQOPEN or MQPUT1 MQI calls that resulted in referencing the default transmission queue did not return an error. Applications might have continued working and failed later on. The reason for the failure was hard to diagnose.

The change ensures that any attempt to set the default transmission queue to `SYSTEM.CLUSTER.TRANSMIT.QUEUE`, or use a default transmission queue set to `SYSTEM.CLUSTER.TRANSMIT.QUEUE`, is immediately reported as an error.

Related reference:

"MQI and PCF reason code changes" on page 43 (*WebSphere MQ V7.1 Installing Guide*)

Changes to `dspmqver` output

From Version 7.1, new types of information are displayed by **`dspmqver`** to support multiple installations. The changes might affect existing administrative scripts you have written to manage IBM WebSphere MQ.

The changes in output from **`dspmqver`** that might affect existing command scripts that you have written are twofold:

1. Version 7.1 has extra `-f` field options. If you do not specify a `-f` option, output from all the options is displayed. To restrict the output to the same information that was displayed in earlier releases, set the `-f` option to a value that was present in the earlier release. Compare the output for `dspmqver` in Figure 2 and Figure 3 on page 33 with the output for `dspmqver -f 15` in Figure 4 on page 33.

```
dspmqver

Name:        WebSphere MQ
Version:     7.0.1.6
CMVC level:  p701-L110705
BuildType:   IKAP - (Production)
```

Figure 2. Default **`dspmqver`** *options in IBM WebSphere MQ Version 7.0.1*

```
dspmqver

Name:        WebSphere MQ
Version:     7.1.0.0
Level:       p000-L110624
BuildType:   IKAP - (Production)
Platform:    WebSphere MQ for Windows
Mode:        32-bit
O/S:         Windows XP, Build 2600: SP3
InstName:    110705
InstDesc:    July 5 2011
InstPath:    C:\Program Files\IBM\WebSphere MQ_110705
DataPath:    C:\Program Files\IBM\WebSphere MQ
Primary:     No
MaxCmdLevel: 710

Note there are a number (1) of other installations,
use the '-i' parameter to display them.
```

Figure 3. Default **dspmqver** *options in IBM WebSphere MQ Version 7.1*

```
dspmqver -f 15

Name:        WebSphere MQ
Version:     7.1.0.0
Level:       p000-L110624
BuildType:   IKAP - (Production)
```

Figure 4. **dspmqver** *with option to make IBM WebSphere MQ Version 7.1 similar to IBM WebSphere MQ Version 7.0.1*

2. The heading of the build level row has changed from `CMVC level:` to `Level:`.

Related reference:

dspmqver (*WebSphere MQ V7.1 Reference*)

Exits and installable services

When migrating to IBM WebSphere MQ Version 7.1, if you install IBM WebSphere MQ in a non-default location, you must update your exits and installable services. Data conversion exits generated using the **crtmqcvx** command must be regenerated using the updated command.

When writing new exits and installable services, you do not need to link to any of the following IBM WebSphere MQ libraries:

- mqmzf
- mqm
- mqmvx
- mqmvxd
- mqic
- mqutl

For more information about updating existing exits, and writing exits and installable services that do not link to the libraries, see Writing and compiling exits and installable services (*WebSphere MQ V7.1 Programming Guide*).

Fewer IBM WebSphere MQ MQI client log messages

A IBM WebSphere MQ MQI client used to report every failed attempt to connect to a queue manager when processing a connection name list. From Version 7.1, only if the failure occurs with the last connection in the list is a message written to the queue manager error log.

Reporting the last failure and no others, reduces growth of the queue manager error log.

GSKit: Changes from GSKit V7.0 to GSKit V8.0

For distributed platforms, from IBM WebSphere MQ Version 7.1, GSKit V8.0 is integrated with IBM WebSphere MQ and is the only version of GSKit provided with the product. GSKit V7.0 is no longer provided.

In IBM WebSphere MQ Version 7.0.1, if you select SSL and TLS support during installation, GSKit Version 7.0 is installed and run by default. IBM WebSphere MQ Version 7.0.1, Fix Pack 4 and later also contain an alternative, separate copy of GSKit Version 8.0 that you can install and run instead of, or in addition to, GSKit Version 7.0. From IBM WebSphere MQ Version 7.1, GSKit Version 8.0 is the only version of GSKit that is provided.

Some functions in GSKit V8.0 are different from the functions in GSKit V7.0. These differences are described in the following subtopics.

GSKit: Some FIPS 140-2 compliant channels do not start:

Three CipherSpecs are no longer FIPS 140-2 compliant. If a client or queue manager is configured to require FIPS 140-2 compliance, channels that use the following CipherSpecs do not start after migration.

- FIPS_WITH_DES_CBC_SHA
- FIPS_WITH_3DES_EDE_CBC_SHA
- TLS_RSA_WITH_DES_CBC_SHA

To restart a channel, alter the channel definition to use a FIPS 140-2 compliant CipherSpec. Alternatively, configure the queue manager, or the client in the case of a IBM WebSphere MQ MQI client, not to enforce FIPS 140-2 compliance.

Earlier versions of IBM WebSphere MQ enforced an older version of the FIPS 140-2 standard. The following CipherSpecs were considered FIPS 140-2 compliant in earlier versions of IBM WebSphere MQ and are also compliant in version 7.1:

- TLS_RSA_WITH_3DES_EDE_CBC_SHA
- TLS_RSA_WITH_AES_128_CBC_SHA
- TLS_RSA_WITH_AES_256_CBC_SHA
- TLS_RSA_WITH_AES_128_CBC_SHA256 (only when AltGSKit version 8 is used with Fix Pack 7.0.1.4 or later)
- TLS_RSA_WITH_AES_256_CBC_SHA256 (only when AltGSKit version 8 is used with Fix Pack 7.0.1.4 or later)

Use these CipherSpecs if you want IBM WebSphere MQ version 7.1 to interoperate in a FIPS 140-2 compliant manner with earlier versions.

Previous IBM WebSphere MQ releases enforced an older version of the FIPS 140-2 standard. The following CipherSpecs were considered FIPS 140-2 compliant by previous IBM WebSphere MQ releases and are also considered compliant by IBM WebSphere MQ version 7.1:

- TLS_RSA_WITH_3DES_EDE_CBC_SHA
- TLS_RSA_WITH_AES_128_CBC_SHA
- TLS_RSA_WITH_AES_256_CBC_SHA

Use these CipherSpecs if you need IBM WebSphere MQ version 7.1 to interoperate in a FIPS 140-2 compliant manner with earlier IBM WebSphere MQ releases.

Related concepts:

Federal Information Processing Standards (FIPS) (*WebSphere MQ V7.1 Administering Guide*)

Specifying that only FIPS-certified CipherSpecs are used at run time on the MQI client (*WebSphere MQ V7.1 Administering Guide*)

Related reference:

FipsRequired (MQLONG) (*WebSphere MQ V7.1 Reference*)

MQSSLFIPS (*WebSphere MQ V7.1 Installing Guide*)

SSLFIPSRequired (MQLONG) (*WebSphere MQ V7.1 Reference*)

Related information:

AMQ9196

Federal Information Processing Standards (*WebSphere MQ V7.1 Administering Guide*)

GSKit: Certificate Common Name (CN) not mandatory:

In GSKit V8.0, the **iKeyman** command accepts any element of the distinguished name (DN), or a form of the subject alternative name (SAN). It does not mandate that you provide it with a common name. In GSKit V7.0, if you create a self-signed certificate using the **iKeyman** command you had to specify a common name.

The implication is that applications searching for a certificate might not able to assume that a certificate has a common name. You might need to review how applications search for certificates, and how applications handle errors involving the common name. Alternatively, you might choose to check that all self-signed certificates are given common names.

Some other certificate tools that you might also be using, do not require a common name. It is therefore likely the change to GSKit is not going to cause you a problem.

Related concepts:

Distinguished Names (*WebSphere MQ V7.1 Administering Guide*)

GSKit: Commands renamed:

The command name **gsk7cmd** is replaced with **runmqckm**; **gsk7ikm** is replaced with **strmqikm**, and **gsk7capicmd** is replaced with **runmqakm**. All the commands start the GSKit V8.0 certificate administration tools, and not the GSKit V7.0 tools.

IBM WebSphere MQ Version 7.1 does not use a machine-wide shared installation of GSKit: instead it uses a private GSKit installation in the IBM WebSphere MQ installation directory. Each IBM WebSphere MQ Version 7.1 installation may use a different GSKit version. To display the version number of GSKit embedded in a particular WebSphere MQ installation, run the **dspmqver** command from that installation as shown in the following table:

Table 4. Renamed GSKit commands

Platform	GSKit V7.0 command	GSKit V8.0 command
UNIX and Linux	`gsk7cmd`	`runmqckm`
UNIX and Linux	`gsk7ikm`	`strmqikm`
Windows,UNIX and Linux	`gsk7capicmd`	`runmqakm`
Windows,UNIX and Linux	`gsk7ver`	`dspmqver -p 64 -v`

Note: Do not use the **gsk8ver** command to display the GSKit version number: only the **dspmqver** command will show the correct GSKit version number for IBM WebSphere MQ Version 7.1.

Related concepts:

Using iKeyman, iKeycmd, runmqakm, and runmqckm (*WebSphere MQ V7.1 Administering Guide*)

Related reference:

runmqckm, and runmqakm commands (*WebSphere MQ V7.1 Reference*)

dspmqver (*WebSphere MQ V7.1 Reference*)

GSKit: The iKeyman command to create a certificate does not validate the certificate:

The **iKeyman** command in GSKit V8.0 does not validate a certificate when it is created. **iKeyman** in GSKit V7.0, validated a certificate before it added it to certificate store.

The implication is that if you create a certificate using the **iKeyman** in GSKit V8.0, all the necessary intermediate and root certificates might not be present, or they might have expired. When the certificate is checked it might fail, Necessary certificates might be missing, or have expired.

Related concepts:

Certificate validation and trust policy design on UNIX, Linux, and Windows systems (*WebSphere MQ V7.1 Administering Guide*)

GSKit: PKCS#11 and JRE addressing mode:

If you use **iKeyman** or **iKeycmd** to administer certificates and keys for PKCS#11 cryptographic hardware note that the addressing mode of the JRE for these tools has changed.

On the following platforms the JRE was 32 bit however in IBM WebSphere MQ Version 7.1 it is now 64 bit only. Where it has changed you might need to install additional PKCS#11 drivers appropriate for the addressing mode of the **iKeyman** and **iKeycmd** JRE. This is because the PKCS#11 driver must use the same addressing mode as the JRE. The following table shows the IBM WebSphere MQ Version 7.1 JRE addressing modes.

Table 5. IBM WebSphere MQ Version 7.1 JRE addressing modes

Platform	JRE Addressing Mode
Windows (32 bit or 64 bit)	32
Linux for System x 32 bit	32
Linux for System x 64 bit	64
Linux for System p	64
Linux for System z	64
HP-UX	64
Solaris Sparc	64

Table 5. IBM WebSphere MQ Version 7.1 JRE addressing modes (continued)

Platform	JRE Addressing Mode
Solaris x86-64	64
AIX	64
z/OS	n/a
IBM i	n/a

GSKit: Import of a duplicate PKCS#12 certificate:

In GSKit V8.0, the **iKeyman** command does not report an attempt to import a duplicate PKCS#12 certificate as an error. In GSKit V7.0, the **iKeyman** command reported an error. In neither version is a duplicate certificate imported.

For GSKIT V8.0, a duplicate certificate is a certificate with the same label and public key.

The implication is that if some of the issuer information is different, but the name and public key are the same, the changes are not imported. The correct way to update a certificate is to use the `-cert -receive` option, which replaces an existing certificate.

gskcapicmd does not allow or ignore duplicates on import in this way.

Related concepts:

Importing a personal certificate into a key repository on UNIX, Linux or Windows systems (*WebSphere MQ V7.1 Administering Guide*)

GSKit: Certificate stores created by iKeyman and iKeycmd no longer contain CA certificates:

The **iKeyman** and **iKeycmd** utilities in GSKit V8.0 create a certificate store without adding pre-defined CA certificates to the store. To create a working certificate store, you must now add all the certificates that you require and trust. In GSKit V7.0 **iKeyman** and **iKeycmd** created a certificate store that already contained CA certificates.

Existing data bases created by GSKit V7.0 are unaffected by this change.

Related concepts:

Adding default CA certificates into an empty key repository, on UNIX, Linux or Windows systems with GSKit version 8.0 (*WebSphere MQ V7.1 Administering Guide*)

GSKit: Password expiry to key database deprecated:

In GSKit V8.0, the password expiry function in **iKeyman** continues to work the same as in GSKit V7.0, but it might be withdrawn in future versions of GSKit.

Use the file system protection provided with the operating system to protect the key database and password stash file.

Linux: Recompile C++ applications and update run time libraries:

C++ IBM WebSphere MQ MQI client and server applications on Linux must be recompiled using a supported version of the GNU Compiler Collection (GCC). The C++ run time libraries for this version of GCC must be installed in `/usr/lib` or `/usr/lib64`.

For information about which versions of GCC are supported , see System Requirements for IBM WebSphere MQ, and follow the links for Linux. Older versions of the GCC that are not included in the System Requirements information are no longer supported.

If you are using one of the supported Linux distributions, the libraries are correctly installed; see System Requirements for IBM WebSphere MQ.

The GCC 4.x libraries support SSL and TLS connections from a IBM WebSphere MQ MQI client. SSL and TLS use GSKit version 8, which depends on `libstdc++.so.6`. `libstdc++.so.6` is included in GCC 4.x.

Related tasks:

Linux: Rebuilding a C++ application (*WebSphere MQ V7.1 Installing Guide*)

GSKit: Signature algorithm moved out of settings file:

In GSKit V8.0, the default signature algorithm used when creating self-signed certificates or certificate requests or selected in the creation dialogs is passed as a command-line parameter. In GSKit V7.0, the default signature algorithm was specified in the settings file.

The change has very little effect: it causes a different default signature algorithm to be selected. It does not alter the selection of a signature algorithm.

Related concepts:

Creating a self-signed personal certificate on UNIX, Linux, and Windows systems (*WebSphere MQ V7.1 Administering Guide*)

Related reference:

runmqckm and runmqakm options (*WebSphere MQ V7.1 Reference*)

GSKit: Signed certificate validity period not within signer validity:

In GSKit V8.0, the **`iKeyman`** command does not check whether the validity period of a resulting certificate is within the validity period of the signed certificate. In GSKit V7.0, **`iKeyman`** checked that the validity period of the resulting certificate was within the validity period of the signed certificate.

The IETF RFC standards for SSL/TLS allow a certificate whose validity dates extend beyond those of its signer. This change to GSKit brings it into line with those standards. The check is whether the certificate is issued within the validity period of the signer, and not whether it expires within the validity period of the signer.

Related concepts:

How SSL and TLS provide identification, authentication, confidentiality, and integrity (*WebSphere MQ V7.1 Administering Guide*)

GSKit: Stricter default file permissions:

The default file permissions set by **runmqckm** and **strmqikm** in IBM WebSphere MQ Version 7.1 on UNIX and Linux are stricter than the permissions that are set by **runmqckm**, **strmqikm**, **gsk7cmd**, and **gsk7ikm** in earlier releases of IBM WebSphere MQ.

The permissions set by **runmqckm** and **strmqikm** in IBM WebSphere MQ Version 7.1 permit only the creator to access the UNIX and Linux SSL/TLS key databases. The **runmqckm**, **strmqikm**, **gsk7cmd**, and **gsk7ikm** tools in earlier releases of IBM WebSphere MQ set world-readable permissions, making the files liable to theft and impersonation attacks.

The permissions set by **gsk7capicmd**, in earlier releases of IBM WebSphere MQ, and **runmqakm** in IBM WebSphere MQ Version 7.1, permit only the creator to access UNIX and Linux SSL/TLS key databases.

The migration of SSL/TLS key databases to Version 7.1 does not alter their access permissions. In many cases, administrators set more restrictive access permissions on these files to overcome the liability to theft and impersonation attacks; these permissions are retained.

The default file permissions set on Windows are unchanged. Continue to tighten up the access permissions on SSL/TLS key database files on Windows after creating the files with **runmqckm** or **strmqikm**.

Related concepts:

Accessing and securing your key database files on Windows (*WebSphere MQ V7.1 Administering Guide*)

Accessing and securing your key database files on UNIX and Linux systems (*WebSphere MQ V7.1 Administering Guide*)

IBM WebSphere MQ Explorer changes

IBM WebSphere Eclipse Platform is no longer shipped with IBM WebSphere MQ; it is not required to run MQ Explorer. The change makes no difference to administrators who run MQ Explorer. For developers who run MQ Explorer in an Eclipse development environment, a change is necessary. You must install and configure a separate Eclipse environment to be able to switch between MQ Explorer and other perspectives.

Packaging changes

To install IBM WebSphere MQ Explorer use the MS0T SupportPac, which provides IBM IBM WebSphere MQ Explorer as a standalone application.

In versions of IBM WebSphere MQ earlier than Version 7.1, you can select the Workbench mode preference in MQ Explorer. In workbench mode, you could switch to the other perspectives installed in the WebSphere Eclipse Platform. You can no longer set the Workbench mode preference, because the WebSphere Eclipse Platform is not shipped with MQ Explorer in Version 7.1.

To switch between MQ Explorer and other perspectives, you must install MQ Explorer into your own Eclipse environment or into an Eclipse-based product. You can then switch between perspectives. For example, you can develop applications using IBM WebSphere MQ classes for JMS or IBM WebSphere MQ Telemetry applications; see Creating your first MQ Telemetry Transport publisher application using Java (*WebSphere MQ V7.1 Programming Guide*)

If you installed extensions to previous versions of MQ Explorer, such as SupportPacs or IBM WebSphere Message Broker Explorer, you must reinstall compatible versions of the extensions after upgrading MQ Explorer to Version 7.1.

If you continue to run IBM WebSphere MQ Version 7.0.1 on the same server as IBM WebSphere MQ Version 7.1, and you use MQ Explorer, each installation uses its own installation of MQ Explorer. When you uninstall Version 7.0.1, its version of MQ Explorer is uninstalled. To remove IBM WebSphere Eclipse Platform, uninstall it separately. The workspace is not deleted.

Test result migration

Test results are not migrated from version to version. To view any test results, you must rerun the tests.

Java: Different message property data type returned

If the data type of a message property is set, the same data type is returned when the message is received. In some circumstances in IBM WebSphere MQ Version 7.0.1, properties set with a specific type were returned with the default type String.

The change affects Java applications that used the MQRFH2 class, and retrieved properties using the `getFieldValue` method.

You can write a message property in Java using a method such as `setIntFieldValue`. In IBM WebSphere MQ Version 7.0.1 the property is written into an `MQRFH2` header with a default type of String. When you retrieve the property with the `getFieldValue` method, a String object is returned. From IBM WebSphere MQ Version 7.1, the change is that the correct type of object is returned, in this example the type of object returned is Integer.

If your application retrieves the property with the `getIntFieldValue` method, there is no change in behavior; an Integer is returned. If property is written to the `MQRFH2` header by some other means, and the data type is set, then `getFieldValue` returns the correct type of object.

Related information:

Class MQRFH2

Java: Change in behavior of default value of MQEnvironment.userID

Change caused when using CLIENT transport for a channel that does not have a security exit defined.

If an IBM WebSphere MQ classes for Java application is connecting to a queue manager, using the CLIENT transport through a channel that does not have a security exit defined, and the MQEnvironment.userID field is left at its default value of the empty string (`""`), the IBM WebSphere MQ classes for Java application queries the value of the Java System Property `user.name` and passes this to the queue manager for authorization as part of the MQQueueManager constructor.

If the user specified by the Java System Property `user.name` is not authorized to access the queue manager, the MQQueueManager constructor throws an MQException containing Reason Code MQRC_NOT_AUTHORIZED.

JMS: Change in behavior of the default user identifier value

From IBM WebSphere MQ Version 7.1, a change to the behavior of the default user identifier value results in an exception if an application attempts to connect to a queue manager using the CLIENT transport through a channel that does not have a security exit defined.

If a IBM WebSphere MQ classes for JMS application is connecting to a queue manager, using the CLIENT transport through a channel that does not have a security exit defined, and no user identifier is specified, by calling `ConnectionFactory.createConnection()`, the IBM WebSphere MQ classes for JMS application queries the value of the Java System Property `user.name` and passes this to the queue manager for authorization as part of the call to create a connection from a connection factory object. This behaviour also occurs when calling `ConnectionFactory.createConnection(String, String)` and passing a blank or null value for the first parameter **`userID`**.

If the user specified by the Java System Property `user.name` is not authorized to access the queue manager, a JMSException containing Reason Code MQRC_NOT_AUTHORIZED is thrown.

JMS: some objects are no longer serializable

JMS objects such as JMSConnections and JMSSessions, which used to be serializable when using the IBM WebSphere MQ for IBM WebSphere MQ Version 7.0.1, are no longer serializable when using IBM WebSphere MQ Version 7.1 or later.

When a Java application serializes an object, state information about that object is written to an output stream, such as a file. The contents of output stream can then be read at a later date, to reconstruct (or deserialize) the Java object so that it can be reused.

The following interfaces provided by the IBM WebSphere MQ classes for JMS are implemented by objects that represent an active connection from an application to an IBM WebSphere MQ queue manager:

- JMSConnection
- JMSQueueConnection
- JMSTopicConnection
- JMSSession
- JMSQueueSession
- JMSTopicSession

There is an IBM WebSphere MQ connection handle (hconn) associated with every JMSConnection, JMSQueueConnection, JMSTopicConnection, JMSSession, JMSQueueSession and JMSTopicSession object that is created by an application.

Serializing one of these objects would result in state information about those objects being written to an output stream. This included information about the connection handle to IBM WebSphere MQ that was associated with the object.

However, there was no guarantee that the connection handle would still be valid when the object was deserialized and resused, which could lead to unexpected behavior. To prevent applications running into these issues, the IBM WebSphere MQ classes for JMS for IBM WebSphere MQ Version 7.1 (and later) will throw a NotSerializableException if an application tries to:

- Serialize a JMSConnection, JMSQueueConnection, JMSTopicConnection, JMSSession, JMSQueueSession or JMSTopicSession object using the method:
  ```
  writeObject(ObjectOutputStream)
  ```
- Deserialize a JMSConnection, JMSQueueConnection, JMSTopicConnection, JMSSession, JMSQueueSession or JMSTopicSession object using the method:
  ```
  readObject(ObjectInputStream)
  ```

JMS: Reason code changes

Some reason codes returned in JMS exceptions have changed. The changes affect `MQRC_Q_MGR_NOT_AVAILABLE` and `MQRC_SSL_INITIALIZATION_ERROR`.

In earlier releases of IBM WebSphere MQ, if a JMS application call fails to connect, it receives an exception with a reason code 2059 (080B) (RC2059): MQRC_Q_MGR_NOT_AVAILABLE (*WebSphere MQ V7.1 Administering Guide*). In Version 7.1, it can still receive `MQRC_Q_MGR_NOT_AVAILABLE`, or one of the following more specific reason codes.

- 2537 (09E9) (RC2537): MQRC_CHANNEL_NOT_AVAILABLE (*WebSphere MQ V7.1 Administering Guide*)
- 2538 (09EA) (RC2538): MQRC_HOST_NOT_AVAILABLE (*WebSphere MQ V7.1 Administering Guide*)
- 2539 (09EB) (RC2539): MQRC_CHANNEL_CONFIG_ERROR (*WebSphere MQ V7.1 Administering Guide*)
- 2540 (09EC) (RC2540): MQRC_UNKNOWN_CHANNEL_NAME (*WebSphere MQ V7.1 Administering Guide*)

Similarly, when trying to connect, a JMS application might have received 2393 (0959) (RC2393): MQRC_SSL_INITIALIZATION_ERROR (*WebSphere MQ V7.1 Administering Guide*). In Version 7.1, it can still receive `MQRC_SSL_INITIALIZATION_ERROR`, or a more specific reason code, such as 2400 (0960) (RC2400): MQRC_UNSUPPORTED_CIPHER_SUITE (*WebSphere MQ V7.1 Administering Guide*), that identifies the cause of the SSL initialization error.

JMS: ResourceAdapter object configuration

When a WebSphere Application Server connects to IBM WebSphere MQ it creates Message Driven Beans (MDBs) using JMS connections. These MDBs can no longer share one JMS connection. The configuration of ResourceAdapter object is migrated so that there is a single MDB for each JMS connection.

Changed ResourceAdapter properties

`connectionConcurrency`

The maximum number of MDBs to share a JMS connection. Sharing connections is not possible and this property always has the value `1`. Its previous default value was `5`.

`maxConnections`

This property is the number of JMS connections that the resource adapter can manage. In Version 7.1, it also determines the number of MDBs that can connect because each MDB requires one JMS connection. The default value of `maxConnections` is now `50`. Its previous default value was `10`.

If `connectionConcurrency` is set to a value greater than 1, the maximum number of connections supported by the resource adapter is scaled by the value of `connectionConcurrency`. For example, if `maxConnections` is set to `2` and `connectionConcurrency` is set to `4`, the maximum number of connections supported by the resource adapter is 8. As a result, `connectionConcurrency` is set to `1` and `maxConnections` is set to 8.

If `connectionConcurrency` is set to a value greater than `1`, it is adjusted automatically. To avoid automatic adjustment, set `connectionConcurrency` to `1`. You can then set `maxConnections` to the value you want.

The scaling mechanism ensures that sufficient connections are available for existing deployments whether you have changed them in your deployment, configuration, or programs.

If the adjusted `maxConnections` value exceeds the `MAXINST` or `MAXINSTC` attributes of any used channel, previously working deployments might fail.

The default value of both channel attributes equates to unlimited. If you changed them from the default value, you must ensure that the new maxConnections value does not exceed MAXINST or MAXINSTC.

Related concepts:

Configuration of the ResourceAdapter object (*WebSphere MQ V7.1 Programming Guide*)

Related reference:

Maximum instances (MAXINST) (*WebSphere MQ V7.1 Reference*)

Maximum instances per client (MAXINSTC) (*WebSphere MQ V7.1 Reference*)

MQI and PCF reason code changes

Reason codes that have changed and that affect some existing programs, are listed.

MQRC_NOT_OPEN_FOR_INPUT

In IBM WebSphere MQ Version 7.0 a queue opened with MQOO_OUTPUT, and then browsed, returned an error with the wrong reason-code, MQRC_NOT_OPEN_FOR_INPUT. The correct reason-code, MQRC_NOT_OPEN_FOR_BROWSE, was issued by Version 6.0 and earlier. Version 7.1 correctly returns an error with the same reason code as Version 6.0, MQRC_NOT_OPEN_FOR_BROWSE.

MQRC_DEF_XMIT_Q_USAGE_ERROR

The product documentation in previous versions of IBM WebSphere MQ warned about defining the default transmission queue as SYSTEM.CLUSTER.TRANSMIT.QUEUE. In Version 7.1, an attempt to open the default transmission queue, defined as SYSTEM.CLUSTER.TRANSMIT.QUEUE, results in the error MQRC_DEF_XMIT_Q_USAGE_ERROR.

MQRC_FASTPATH_NOT_AVAILABLE

An application that connects to multiple queue managers in the same process and uses MQCNO_FASTPATH_BINDING might fail with an error and reason code MQRC_FASTPATH_NOT_AVAILABLE; see “Connect to multiple queue managers and use MQCNO_FASTPATH_BINDING” on page 30 (*WebSphere MQ V7.1 Installing Guide*).

MQRCCF_DEF_XMIT_Q_CLUS_ERROR

The product documentation in previous versions of IBM WebSphere MQ warned about defining the default transmission queue as SYSTEM.CLUSTER.TRANSMIT.QUEUE. In Version 7.1, an attempt to alter the queue manager attribute **DEFXMITQ** to SYSTEM.CLUSTER.TRANSMIT.QUEUE results in an error.

The PCF reason code is 3269 (0CC5) (RC3269): MQRCCF_DEF_XMIT_Q_CLUS_ERROR (*WebSphere MQ V7.1 Administering Guide*).

Related reference:

“Connect to multiple queue managers and use MQCNO_FASTPATH_BINDING” on page 30 (*WebSphere MQ V7.1 Installing Guide*)

2036 (07F4) (RC2036): MQRC_NOT_OPEN_FOR_BROWSE (*WebSphere MQ V7.1 Administering Guide*)

2037 (07F5) (RC2037): MQRC_NOT_OPEN_FOR_INPUT (*WebSphere MQ V7.1 Administering Guide*)

2590 (0A1E) (RC2590): MQRC_FASTPATH_NOT_AVAILABLE (*WebSphere MQ V7.1 Administering Guide*)

3269 (0CC5) (RC3269): MQRCCF_DEF_XMIT_Q_CLUS_ERROR (*WebSphere MQ V7.1 Administering Guide*)

Related information:

AMQ8520

Publish/Subscribe: Delete temporary dynamic queue

If a subscription is associated with a temporary dynamic queue, when the queue is deleted, the subscription is deleted. This changes the behavior of incorrectly written publish/subscribe applications migrated from Version 6.0. Publish/subscribe applications migrated from IBM WebSphere Message Broker are unchanged. The change does not affect the behavior of integrated publish/subscribe applications, which are written using the MQI publish/subscribe interface.

- In IBM WebSphere MQ Version 6.0 the product documentation stated “The subscriber queue must not be a temporary dynamic queue”; see the *Queue name* in the IBM WebSphere MQ Version 6.0 product documentation. But the injunction was not enforced, although the configuration is not supported.
- In IBM WebSphere Message Broker and WebSphere Event Broker version 6.0, and IBM WebSphere Message Broker version 6.1, you could create a subscription that used a temporary dynamic queue as the subscriber queue. If the queue was deleted, the subscription was deleted with it.
- In IBM WebSphere MQ Version 7.0, if you migrate or create a queued publish/subscribe application that uses `MQRFH1`, it behaves the same as IBM WebSphere MQ Version 6.0. You can create a temporary dynamic queue for a subscription, and if the queue is deleted, the subscription is not deleted, as in IBM WebSphere MQ Version 6.0. The lack of a subscriber queue results in any matching publications ending up on the dead letter queue. This behavior is the same as IBM WebSphere MQ Version 6.0. `MQRFH1` publish/subscribe applications are typically migrated from IBM WebSphere MQ Version 6.0.
- In IBM WebSphere MQ Version 7.0.1 from fix pack 7.0.1.6 onwards, and in Version 7.1, in the same `MQRFH1` queued publish/subscribe case, if the temporary dynamic queue is deleted, the subscription is deleted. This change prevents a buildup of publications from a subscription without a subscriber queue ending up on the dead letter queue. It is a change from IBM WebSphere MQ Version 6.0.
- In Version 7.1, if you migrate or create a queued publish/subscribe application that uses `MQRFH2`, it behaves the same as WebSphere Event Broker and IBM WebSphere Message Broker. You can create a temporary dynamic queue for a subscription, and if the queue is deleted, the subscription is deleted, as in WebSphere Event Broker and IBM WebSphere Message Broker. `MQRFH2` publish/subscribe applications are typically migrated from WebSphere Event Broker or IBM WebSphere Message Broker.
- In IBM WebSphere MQ Version 7.1, if you create a durable subscription using integrated publish/subscribe, you cannot define a temporary dynamic queue as the destination for its matching publications.
- In IBM WebSphere MQ Version 7.1, you can create a managed, non-durable subscription using integrated publish/subscribe, which creates a temporary dynamic queue as the destination for matching publications. The subscription is deleted with the queue.

Summary

In Version 7.1, you cannot create a temporary dynamic queue as the destination for publications for a durable subscription using the integrated publish/subscribe interface.

In the current fix level of Version 7.1, if you use either of the queued publish/subscribe interfaces, `MQRFH1` or `MQRFH2`, the behavior is the same. You can create a temporary dynamic queue as the subscriber queue, and if the queue is deleted, the subscription is deleted with it. Deleting the subscription with the queue retains the same the supported behavior of IBM WebSphere MQ Version 6.0, WebSphere Event Broker, and IBM WebSphere Message Broker applications. It modifies the unsupported behavior of IBM WebSphere MQ Version 6.0 applications.

Queue manager log files and error logs: Default sizes increased

From IBM WebSphere MQ Version 7.1, the default size of queue manager log files is changed to 4096 pages, and the error log, `AMQERRnn.log`, increases in size from 256 KB to 2 MB (2 097 152 bytes) on IBM i, UNIX, Linux, and Windows. The changes affect both new and migrated queue managers.

Queue manager log

In IBM WebSphere MQ Version 7.1, the default queue manager log size is increased to 4096 pages. For more information about setting non-default values, see Log defaults for WebSphere MQ (*WebSphere MQ V7.1 Installing Guide*).

Queue manager error log

In IBM WebSphere MQ Version 7.1, the default queue manager error log size is increased to 2 MB (2 097 152 bytes).

Override the change by setting the environment variable `MQMAXERRORLOGSIZE`, or setting `ErrorLogSize` in the `QMErrorLog` stanza in the `qm.ini` file.

The change increases the number of error messages that are saved in the error logs.

Related reference:

Queue manager error logs (*WebSphere MQ V7.1 Installing Guide*)

Queue manager error logs on IBM i (*WebSphere MQ V7.1 Installing Guide*)

`SSLPEER` and `SSLCERTI` changes

IBM WebSphere MQ Version 7.1 obtains the Distinguished Encoding Rules (DER) encoding of the certificate and uses it to determine the subject and issuer distinguished names. The subject and issuer distinguished names are used in the `SSLPEER` and `SSLCERTI` fields. A `SERIALNUMBER` attribute is also included in the subject distinguished name and contains the serial number for the certificate of the remote partner. Some attributes of subject and issuer distinguished names are returned in a different sequence from previous releases.

The change to subject and issuer distinguished names affects channel security exits. It also affects aplications which depend upon the subject and issuer distinguished names that are returned by the PCF programming interface. Channel security exits and applications that set or query `SSLPEER` and `SSLCERTI` must be examined, and possibly changed. The fields that are affected are listed in Table 6 and Table 7 on page 46.

Table 6. Channel status fields affected by changes to subject and issuer distinguished names

Channel status attribute	**PCF channel parameter type**
SSL Peer (SSLPEER) (*WebSphere MQ V7.1 Reference*)	`MQCACH_SSL_SHORT_PEER_NAME`
`SSLCERTI`	`MQCACH_SSL_CERT_ISSUER_NAME`

Table 7. Channel data structures affected by changes to subject and issuer distinguished names

Channel data structure	Field
MQCD – Channel definition (*WebSphere MQ V7.1 Reference*)	SSLPeerNamePtr (MQPTR) (*WebSphere MQ V7.1 Reference*)
MQCXP – Channel exit parameter (*WebSphere MQ V7.1 Reference*)	SSLRemCertIssNamePtr (PMQVOID) (*WebSphere MQ V7.1 Reference*)

Existing peer name filters specified in the `SSLPEER` field of a channel definition are not affected. They continue to operate in the same manner as in earlier releases. The peer name matching algorithm has been updated to process existing `SSLPEER` filters. It is not necessary to alter any channel definitions.

Related concepts:

Channel security exit programs (*WebSphere MQ V7.1 Programming Guide*)

Telemetry: Installer integrated with IBM WebSphere MQ

IBM WebSphere MQ Telemetry is no longer installed separately from IBM WebSphere MQ. It is installed as a component of IBM WebSphere MQ. If you installed IBM WebSphere MQ Telemetry with Version 7.0.1, you must uninstall it before installing Version 7.1.

You can install IBM WebSphere MQ Telemetry at the same time as IBM WebSphere MQ, or you can rerun the installer and install IBM WebSphere MQ Telemetry at a later time.

Related tasks:

Installing IBM WebSphere MQ Telemetry (*WebSphere MQ V7.1 Installing Guide*)

Uninstalling IBM WebSphere MQ Telemetry components (*WebSphere MQ V7.1 Installing Guide*)

AIX: Shared objects

In IBM WebSphere MQ for AIX Version 7.1 the `.a` shared objects in the `lib64` directory contains both the 32 bit and 64 bit objects. A symlink to the `.a` file is also placed in the `lib` directory. The AIX loader can then correctly pick up the correct object for the type of application being run.

This means that IBM WebSphere MQ applications can run with the `LIBPATH` containing either the `lib` or `lib64` directory, or both.

AIX: `/usr/lpp/mqm` symbolic link removed

Before Version 6.0, IBM WebSphere MQ placed a symbolic link in `/usr/lpp/mqm` on AIX. The link ensured queue managers and applications migrated from IBM WebSphere MQ versions before Version 5.3 continued to work, without change. The link is not created in Version 7.1.

In version 5.0, IBM WebSphere MQ for AIX was installed into `/usr/lpp/mqm`. That changed in Version 5.3 to `/usr/mqm`. A symbolic link was placed in `/usr/lpp/mqm`, linking to `/usr/mqm`. Existing programs and scripts that relied on the installation into `/usr/lpp/mqm` continued to work unchanged. That symbolic link has been removed in Version 7.1, because you can now install IBM WebSphere MQ in any directory. Applications and command scripts are affected by the change.

The effect on applications is no different to the effect of migrating on other UNIX and Linux platforms. If the installation is made primary, then symbolic links to the IBM WebSphere MQ link libraries are placed in `/usr/lib`. Most applications migrated from earlier IBM WebSphere MQ versions search the default search path, which normally includes `/usr/lib`. The applications find the symbolic link to the IBM WebSphere MQ load libraries in `/usr/lib`.

If the installation is not primary, then you must configure the correct search path to load the IBM WebSphere MQ link libraries. If you choose to run **setmqenv**, IBM WebSphere MQ places the IBM WebSphere MQ link library path into `LIBPATH`. Unless the application is configured not to search the `LIBPATH`, if for example it is a `setuid` or `setgid` application, then the IBM WebSphere MQ library is loaded successfully; see UNIX and Linux: Migrating IBM WebSphere MQ library loading from Version 7.0.1 to Version 7.1 (*WebSphere MQ V7.1 Installing Guide*).

If you have written command scripts that run IBM WebSphere MQ commands, you might have coded explicit paths to the directory tree where IBM WebSphere MQ was installed. You must modify these command scripts. You can run **setmqenv** to create the correct environment to run the command scripts. If you have set the installation as primary, you do not have to specify the path to the command.

Related tasks:

UNIX and Linux: Migrating IBM WebSphere MQ library loading from Version 7.0.1 to Version 7.1 (*WebSphere MQ V7.1 Installing Guide*)

Related information:

Loading IBM WebSphere MQ libraries

AIX, HP-UX, and Solaris: Building applications for TXSeries

You must rebuild IBM WebSphere MQ applications that link to TXSeries.

Before Version 7.1, IBM WebSphere MQ applications that used the TXSeries CICS support loaded the IBM WebSphere MQ library, `mqz_r`. From Version 7.1 those applications must load the IBM WebSphere MQ library, `mqzi_r` instead. You must change your build scripts accordingly.

Version 7.1 of `mqz_r` includes code to load a different version of the IBM WebSphere MQ library. IBM WebSphere MQ loads a different version of the IBM WebSphere MQ library, if it detects that the queue manager the application is connected to is associated with a different installation to the one from which the library was loaded. `mqzi_r` does not include the additional code. When using TXSeries, the application must run with the IBM WebSphere MQ library it loaded, and not a different library loaded by IBM WebSphere MQ. For this reason, IBM WebSphere MQ applications that use the IBM WebSphere MQ TXSeries support must load the `mqzi_r` library, and not the `mqz_r` library.

An implication of applications loading `mqzi_r` is that the application must load the correct version of `mqzi_r`. It must load the one from the installation that is associated with the queue manager that the application is connected to.

Related concepts:

AIX: TXSeries CICS support (*WebSphere MQ V7.1 Programming Guide*)

Solaris: TXSeries CICS support (*WebSphere MQ V7.1 Programming Guide*)

Related reference:

Building libraries for use with TXSeries for Multiplatforms version 5 (*WebSphere MQ V7.1 Programming Guide*)

HP-UX: TXSeries CICS support (*WebSphere MQ V7.1 Programming Guide*)

Linux: Recompile C++ applications and update run time libraries

C++ IBM WebSphere MQ MQI client and server applications on Linux must be recompiled using a supported version of the GNU Compiler Collection (GCC). The C++ run time libraries for this version of GCC must be installed in `/usr/lib` or `/usr/lib64`.

For information about which versions of GCC are supported , see System Requirements for IBM WebSphere MQ, and follow the links for Linux. Older versions of the GCC that are not included in the System Requirements information are no longer supported.

If you are using one of the supported Linux distributions, the libraries are correctly installed; see System Requirements for IBM WebSphere MQ.

The GCC 4.x libraries support SSL and TLS connections from a IBM WebSphere MQ MQI client. SSL and TLS use GSKit version 8, which depends on `libstdc++.so.6`. `libstdc++.so.6` is included in GCC 4.x.

Related tasks:

Linux: Rebuilding a C++ application (*WebSphere MQ V7.1 Installing Guide*)

Linux: Increased shared memory allocation required

The default system allocation for the maximum amount of shared memory (`SHMMAX`) to allocate on Linux systems is 32 MB. IBM WebSphere MQ starts by allocating 64 MB and increases its allocation on demand by doubling its previous allocation. On a production system set `SHMMAX` to at least 256 MB to accommodate additional allocations.

Related concepts:

Additional settings for installing on Linux systems (*WebSphere MQ V7.1 Installing Guide*)

UNIX and Linux: `crtmqlnk` and `dltmqlnk` removed

The **`crtmqlnk`** and **`dltmqlnk`** commands are not present in this version of the product. In versions prior to Version 7.1, the commands created symbolic links in subdirectories of `/usr`. From Version 7.1 onwards, you must use the **`setmqinst`** command instead.

Related tasks:

Changing the primary installation (*WebSphere MQ V7.1 Installing Guide*)

Related reference:

"UNIX and Linux: `/usr` symbolic links removed" on page 50 (*WebSphere MQ V7.1 Installing Guide*)

setmqinst (*WebSphere MQ V7.1 Reference*)

UNIX and Linux: Message catalogs moved

IBM WebSphere MQ message catalogs are no longer stored in the system directories in Version 7.1. To support multiple installations, copies of the message catalogs are stored with each installation. If you want messages only in the locale of your system, the change has no affect on your system. If you have customized the way the search procedure selects a message catalog, then the customization might no longer work correctly.

Set the `LANG` environment variable to load a message catalog for a different language from the system locale.

Related tasks:

Displaying messages in your national language on UNIX and Linux systems (*WebSphere MQ V7.1 Installing Guide*)

UNIX and Linux: MQ services and triggered applications

IBM WebSphere MQ Version 7.1 has been configured to allow both `LD_LIBRARY_PATH` and `$ORIGIN` to work for MQ services and triggered applications. For this reason MQ Services and triggered applications have been changed so that they run under the user ID who started the queue manager and not `setuid` or `setgid`.

If any files used by the service were previously restricted to certain users, then they might not be accessible by the user ID who started the queue manager. Resources used by MQ services or triggered applications must be adjusted as appropriate.

Note:

- On AIX, `LD_LIBRARY_PATH` is also known as `LIBPATH` and `$ORIGIN` is not supported.
- On HP-UX, `LD_LIBRARY_PATH` is also known as `SHLIB_PATH`.

Related concepts:

Working with services (*WebSphere MQ V7.1 Administering Guide*)

Starting WebSphere MQ applications using triggers (*WebSphere MQ V7.1 Programming Guide*)

UNIX and Linux: `ps -ef | grep amq` interpretation

The interpretation of the list of IBM WebSphere MQ processes that results from filtering a scan of UNIX or Linux processes has changed. The results can show IBM WebSphere MQ processes running for multiple installations on a server. Before Version 7.1, the search identified IBM WebSphere MQ processes running on only a single installation of IBM WebSphere MQ on a UNIXor Linux server.

The implications of this change depend on how the results are qualified and interpreted, and how the list of processes is used. The change affects you, only if you start to run multiple installations on a single server. If you have incorporated the list of IBM WebSphere MQ processes into administrative scripts or manual procedures, you must review the usage.

Examples

The following two examples, which are drawn from the product documentation, illustrate the point.

1. In the product documentation, before Version 7.1, the scan was used as a step in tasks to change the installation of IBM WebSphere MQ. The purpose was to detect when all queue managers had ended. In Version 7.1, the tasks use the **`dspmq`** command to detect when all queue managers associated with a specific installation have ended.
2. In the product documentation, a process scan is used to monitor starting a queue manager in a high availability cluster. Another script is used to stop a queue manager. In the script to stop a queue manager, if the queue manager does not end within a period of time, the list of processes is piped into a **`kill -9`** command. In both these cases, the scan filters on the queue manager name, and is unaffected by the change to multiple installations.

Related concepts:

Stopping a queue manager under the control of a high availability (HA) cluster (*WebSphere MQ V7.1 Installing Guide*)

Monitoring a queue manager (*WebSphere MQ V7.1 Installing Guide*)

UNIX and Linux: `/usr` symbolic links removed

On all UNIX and Linux platforms, the links from the `/usr` file system are no longer made automatically. In order to take advantage of these links, you must set an installation as the primary installation or set the links up manually.

In previous releases the installation of IBM WebSphere MQ on UNIX and Linux created the symbolic links shown in Table 8. In Version 7.1, these links are not created. You must run **`setmqinst`** to create a primary installation containing symbolic links. No symbolic links are created in other installations.

Table 8. Default symbolic links created in releases before Version 7.1

Symbolic link from	To
`/usr/bin/amq...`	`/opt/mqm/bin/amq...`
`/usr/lib/amq...`	`/opt/mqm/lib/amq...`
`/usr/include/cmq...`	`/opt/mqm/inc/cmq...`
`/usr/share/man/...`	`/opt/mqm/man/...`

Only a subset of those links created with previous releases are now made, see External library and control command links to primary installation on UNIX and Linux (*WebSphere MQ V7.1 Installing Guide*)

For more information about choosing whether to have a primary installation, see Choosing a primary installation (*WebSphere MQ V7.1 Installing Guide*)

Related tasks:

Changing the primary installation (*WebSphere MQ V7.1 Installing Guide*)

UNIX and Linux: Migrating IBM WebSphere MQ library loading from Version 7.0.1 to Version 7.1 (*WebSphere MQ V7.1 Installing Guide*)

Related reference:

setmqinst (*WebSphere MQ V7.1 Reference*)

Windows: `amqmsrvn.exe` process removed

The `amqmsrvn.exe` DCOM process is replaced by a Windows service, `amqsvc.exe`, in Version 7.1. This change is unlikely to cause any problems. However, you might have to make some changes. You might have configured the user that runs the IBM WebSphere MQ Windows service `MQSeriesServices` without the user right to "Log on as a service". Alternatively, the user might not have "List Folder" authority on all the subdirectories from the root of the drive to the location of the service `amqsvc.exe`.

If you omitted the "Log on as a service" user right, or one of the subdirectories under which IBM WebSphere MQ is installed does not grant the "List Folder" authority to the user, the `MQ_`*`InstallationName`* IBM WebSphere MQ Windows services in Version 7.1 fails to start.

Diagnosing the problem

If the service fails to start, Windows event messages are generated:

- If you did not give the user the "Log on as a service" user right, the Windows Service Control Manager adds an event: `7038: The user has not been granted the requested logon type`. The **`strmqsvc`** command reports error 1069.
- If you did not give the user the "List Folder" authority, the Windows Service Control Manager adds an event: `7009: Timed out waiting for the service to connect`. The **`strmqsvc`** command reports error 1053.

If the "Prepare WebSphere MQ wizard" encounters a failure when validating the security credentials of the user performing an installation, an error is returned: `WebSphere MQ is not correctly configured for Windows domain users`. This error indicates that the service failed to start.

Resolution

To resolve this problem:

- Check that the user has the "Log on as a service" user right
- Check that the user is a member of the local mqm group
- Check that the local mqm group has "List Folder" authority on each subdirectory in the path to the service `amqsvc.exe`.

Related reference:

"Windows: "Logon as a service" required" on page 52 (*WebSphere MQ V7.1 Installing Guide*)

Windows: IgnoredErrorCodes registry key

The registry key used to specify error codes that you do not want written to the Windows Application Event Log has changed.

The contents of this registry key are not automatically migrated. If you want to continue to ignore specific error codes, you must manually migrate the registry key.

Previously, the key was in the following location:

```
HKLM\Software\IBM\MQSeries\CurrentVersion\IgnoredErrorCodes
```

The key is now in the following location:

```
HKLM\Software\IBM\WebSphere MQ\Installation\MQ_INSTALLATION_NAME\IgnoredErrorCodes
```

where *`MQ_INSTALLATION_NAME`* is the installation name associated with a particular installation of IBM WebSphere MQ.

Related information:

Ignoring error codes under Windows systems

Windows: Installation and infrastructure information

The location of Windows installation and infrastructure information has changed.

A top-level string value, `WorkPath`, in the `HKLM\SOFTWARE\IBM\WebSphere MQ` key, stores the location of the product data directory which is shared between all installations. The first installation on a machine specifies it, subsequent installations pick up the same location from this key.

Other information previously stored in the registry on Windows is now stored in .ini files.

Related concepts:

Changing configuration information on Windows, UNIX and Linux systems (*WebSphere MQ V7.1 Installing Guide*)

Windows: Local queue performance monitoring

In IBM WebSphere MQ for Windows Version 7.1 it is no longer possible to monitor local queues using the Windows performance monitor.

Use the performance monitoring commands, which are common to all platforms, provided by IBM WebSphere MQ.

Related concepts:

Real-time monitoring (*WebSphere MQ V7.1 Administering Guide*)

Windows: “Logon as a service” required

The user ID that runs the IBM WebSphere MQ Windows service must have the user right to “Logon as a service”. If the user ID does not have the right to run the service, the service does not start and it returns an error in the Windows system event log. Typically you will have run the Prepare IBM WebSphere MQ wizard, and set up the user ID correctly. Only if you have configured the user ID manually is it possible that you might have a problem in Version 7.1.

You have always been required to give the user ID that you configure to run IBM WebSphere MQ the user right to “Logon as a service”. If you run the Prepare IBM WebSphere MQ wizard, it creates a user ID with this right. Alternatively, it ensures that a user ID you provide has this right.

It is possible that you ran IBM WebSphere MQ in earlier releases with a user ID that did not have the “Logon as a service” right. You might have used it to configure the IBM WebSphere MQ Windows service MQSeriesServices, without any problems. If you run a IBM WebSphere MQ Windows service in Version 7.1 with the same user ID that does not have the “Logon as a service” right, the service does not start.

The IBM WebSphere MQ Windows service MQSeriesServices, with the display name `IBM MQSeries`, has changed in Version 7.1. A single IBM WebSphere MQ Windows service per server is no longer sufficient. Version 7.1 requires a IBM WebSphere MQ Windows service per installation. Each service is named `MQ_`*`InstallationName`*, and has a display name `IBM WebSphere MQ (`*`InstallationName`*`)`. The change, which is necessary to run multiple installations of IBM WebSphere MQ, has prevented IBM WebSphere MQ running the service under a single specific user ID. In Version 7.1, a `MQ_`*`InstallationName`* service must run as a service.

The consequence is, from Version 7.1 onwards, a user ID that is configured to run the Windows service `MQ_`*`InstallationName`* must be configured to “Logon as a service”. If the user ID is not configured correctly, errors are returned in the Windows system event log.

Many installations on earlier releases, and new installations from Version 7.1 onwards, configure IBM WebSphere MQ with the Prepare IBM WebSphere MQ wizard. The wizard sets up the user ID with the “Logon as a service” right and configures the IBM WebSphere MQ Windows service with this user ID. Only if, in previous releases, you have configured MQSeriesServices with another user ID that you configured manually, might you have this migration problem to fix.

Windows: MSCS restriction with multiple installations

When you install or upgrade to IBM WebSphere MQ Version 7.1, the first IBM WebSphere MQ installation on the server is the only one that can be used with Microsoft Cluster Server (MSCS). No other installations on the server can be used with MSCS. This restriction limits the use of MSCS with multiple IBM WebSphere MQ installations.

When you run the **haregtyp** command it defines the first IBM WebSphere MQ to be installed as an MSCS resource type; see WebSphere MQ MSCS support utility programs (*WebSphere MQ V7.1 Installing Guide*). The implications are as follows:

1. You must associate queue managers that are participating in an MSCS cluster with the first installation on the server.
2. Setting the primary installation has no effect on which installation is associated with the MSCS cluster.
3. If you are upgrading from Version 7.0.1 to Version 7.1, you must follow the single-stage migration scenario; see UNIX, Linux, and Windows: Single-stage migration from Version 7.0.1 to Version 7.1 (*WebSphere MQ V7.1 Installing Guide*).

Windows: Migration of registry information

Before Version 7.1 all IBM WebSphere MQ configuration information, and most queue manager configuration information, was stored in the Windows registry. From Version 7.1 onwards all configuration information is stored in files. If Version 7.0.1 is uninstalled from server that has other IBM WebSphere MQ installations, an additional migration step must be performed. The additional step completes transferring configuration information from the registry to the `mqs.ini` file.

The change does not affect the operation of existing applications or queue managers, but it does affect any administrative procedures and scripts that reference the registry.

Before Version 7.0.1 all IBM WebSphere MQ configuration information was stored in the Windows registry. In Version 7.0.1, to support multi-instance queue managers, the queue manager configuration information of some queue managers is stored in `qm.ini` and `qmstatus.ini` rather than in the registry.

In Version 7.1, all IBM WebSphere MQ configuration information about Windows is stored in files; the same files as in UNIX and Linux. If you are migrating an existing Windows system to Version 7.1, the transfer of configuration data from the registry to files is automatic. It takes place when the installation is upgraded to Version 7.1.

If Version 7.0.1 is uninstalled rather than upgraded, and there are other IBM WebSphere MQ Version 7.1 installations on the same server, the migration requires extra steps.

The Version 7.0.1 configuration information is accessed from other installations. You must stop all the queue managers and IBM WebSphere MQ applications running on the server to release any locks.

The IBM WebSphere MQ configuration information in the registry is automatically migrated to `qm.ini` and `qmstatus.ini` when Version 7.0.1 is uninstalled from a server that has a Version 7.1, or later, installation. See step 2 in UNIX, Linux, and Windows: Side-by-side migration from Version 7.0.1 to Version 7.1 (*WebSphere MQ V7.1 Installing Guide*) and step 5 in UNIX, Linux, and Windows: Multi-stage migration from Version 7.0.1 to Version 7.1 (*WebSphere MQ V7.1 Installing Guide*). As a consequence, after uninstallation of Version 7.0.1 on a multi-installation server, it is difficult to restore a Version 7.0.1 installation to run any queue managers that you want to restore to the `701` command level:

1. You cannot reinstall Version 7.0.1 on the server. You must run the queue managers on a different server.

2. When you transfer the queue manager data to another server, with Version 7.0.1 installed, you must create the correct registry configuration entries. The entries are not available to copy from the registry on the multi-installation server. Back up the registry entries before uninstalling Version 7.0.1.

Related concepts:

Changing configuration information on Windows, UNIX and Linux systems (*WebSphere MQ V7.1 Installing Guide*)

Related tasks:

UNIX, Linux, and Windows: Side-by-side migration from Version 7.0.1 to Version 7.1 (*WebSphere MQ V7.1 Installing Guide*)

UNIX, Linux, and Windows: Multi-stage migration from Version 7.0.1 to Version 7.1 (*WebSphere MQ V7.1 Installing Guide*)

Windows: Relocation of the mqclient.ini file

In IBM WebSphere MQ for Windows Version 7.1 the `mqclient.ini` file has moved from FilePath to WorkPath. This is similar to the model already used on UNIX and Linux systems.

For users who supply separate file and work paths you will see a change in behavior. You have an additional step to perform when you choose to uninstall IBM WebSphere MQ Version 7.0 before installing IBM WebSphere MQ Version 7.1. Before uninstalling IBM WebSphere MQ Version 7.0, you must copy `mqclient.ini` directly to the Config directory in your data path so that it can be picked up by the IBM WebSphere MQ Version 7.1 installation.

Windows: Task manager interpretation

The interpretation of the processes listed by the Windows Task Manager has changed. The results can show IBM WebSphere MQ processes running for multiple installations on a server. Before Version 7.1, the process list identified IBM WebSphere MQ processes running on only a single installation of IBM WebSphere MQ on a Windows server.

The implications of this change depend on how the results are qualified and interpreted, and how the list of processes is used. The change affects you, only if you start to run multiple installations on a single server. If you have incorporated the list of IBM WebSphere MQ processes into administrative scripts or manual procedures, you must review the usage.

Related concepts:

Stopping queue managers in WebSphere MQ for Windows (*WebSphere MQ V7.1 Administering Guide*)

Windows: IBM WebSphere MQ Active Directory Services Interface

The IBM WebSphere MQ Active Directory Services Interface is no longer available.

If your application uses the IBM WebSphere MQ Active Directory Services Interface, you must rewrite your application to use Programmable Command Formats.

Related concepts:

Introduction to Programmable Command Formats (*WebSphere MQ V7.1 Administering Guide*)

z/OS: Clustered QALIAS MQOO_BND_AS_Q_DEF

Prior to IBM WebSphere MQ version 7.1, a clustered QALIAS defined with DEFBIND(OPEN), and an application opening the alias with an MQOO_BND_AS_Q_DEF command resolved to MQOO_BND_NOT_FIXED. This behaviour is changed for this scenario in IBM WebSphere MQ version 7.1.

Note: DEFBIND(OPEN) and MQOO_BND_AS_Q_DEF are default settings in IBM WebSphere MQ version 7.1.

To maintain IBM WebSphere MQ version 7.0.1 behaviour, the DEFBIND option on clustered QALIAS objects should be changed from OPEN to NOTFIXED.

What's changed in IBM WebSphere MQ Version 7.1 Fix Packs

Changes to functions and resources in IBM WebSphere MQ Version 7.1 Fix Packs are described in this section.

- "IBM WebSphere MQ Version 7.1.0, Fix Pack 1: NSA Suite B Support"
- "IBM WebSphere MQ Version 7.1.0, Fix Pack 2: Certificate validation policies"
- "IBM WebSphere MQ Version 7.1.0, Fix Pack 6: IBM WebSphere MQ classes for JMS in a CICS OSGi JVM server" on page 56
- "IBM WebSphere MQ Version 7.1.0, Fix Pack 8: Restriction on the use of topic alias queues in distribution lists" on page 56
- "IBM WebSphere MQ Version 7.1.0, Fix Pack 8: GSKit version updated" on page 56
- "IBM WebSphere MQ Version 7.1.0, Fix Pack 9: CipherSpec deprecation" on page 56
- "IBM WebSphere MQ Version 7.1.0, Fix Pack 9: Support for class name whitelisting in IBM WebSphere MQ classes for JMS ObjectMessage" on page 56
- "IBM WebSphere MQ Version 7.1.0, Fix Pack 9: New constant JMS_IBM_SUBSCRIPTION_USER_DATA added to the JmsConstants interface" on page 57
- "IBM WebSphere MQ Version 7.1.0, Fix Pack 9 plus interim fix for APAR IT26482: Change to authorities needed for IBM WebSphere MQ classes for JMS" on page 57

IBM WebSphere MQ Version 7.1.0, Fix Pack 1: NSA Suite B Support

From Version 7.1.0, Fix Pack 1, IBM WebSphere MQ implements the updated RFC 6460 Suite B standard instead of the original RFC 5430. There are two changes in the new version of the standard:

- The CBC-based CipherSpecs `ECDHE_ECDSA_WITH_AES_128_CBC_SHA256` and `ECDHE_ECDSA_WITH_AES_256_CBC_SHA384` are no longer Suite B compliant and cannot be used unless Suite B compliance is disabled.
- The security levels are less restrictive. The configured security levels now determine the minimum acceptable Suite B cryptographic strength. Any CipherSpec or digital certificate which exceeds the minimum is also permitted.

  This means that the 128-bit security level now permits anything that offers either 128-bit Suite B security, or 192-bit Suite B security. Therefore, there is no longer a distinction between configuring the 128-bit security level on its own and configuring both security levels.

For more information about IBM WebSphere MQ and Suite B support, see NSA Suite B Cryptography in IBM WebSphere MQ (*WebSphere MQ V7.1 Administering Guide*).

IBM WebSphere MQ Version 7.1.0, Fix Pack 2: Certificate validation policies

From Version 7.1.0, Fix Pack 2, on UNIX, Linux, and Windows, you can specify how strictly the certificate chain validation conforms to the RFC 5280 industry security standard. For more information, see Certificate validation policies in WebSphere MQ (*WebSphere MQ V7.1 Administering Guide*).

The certificate validation policy can be configured only on queue managers at command level 711, or higher. For more information about command levels and enabling fix pack function, see New function in maintenance level upgrades (*WebSphere MQ V7.1 Installing Guide*).

IBM WebSphere MQ Version 7.1.0, Fix Pack 6: IBM WebSphere MQ classes for JMS in a CICS OSGi JVM server

From Version 7.1.0, Fix Pack 6, IBM WebSphere MQ adds support for using the IBM WebSphere MQ classes for JMS in certain versions of the CICS Open Services Gateway initiative (OSGi) Java Virtual Machine (JVM) server.

For more information, see Using IBM WebSphere MQ classes for JMS in a CICS OSGi JVM server.

IBM WebSphere MQ Version 7.1.0, Fix Pack 8: 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 7.1.0, Fix Pack 8, if an alias queue points to a topic object in a distribution list, IBM WebSphere MQ returns `MQRC_ALIAS_BASE_Q_TYPE_ERROR`.

IBM WebSphere MQ Version 7.1.0, Fix Pack 8: GSKit version updated

From Version 7.1.0, Fix Pack 8, the GSKit version has been updated. 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 7.1.0, Fix Pack 8, or later, are compatible with your applications and other IBM WebSphere MQ installations, you must update to a version of IBM WebSphere 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 WebSphere 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.

IBM WebSphere MQ Version 7.1.0, Fix Pack 9: CipherSpec deprecation

From Version 7.1.0, Fix Pack 9, the following CipherSpecs are deprecated:

- TLS_RSA_WITH_3DES_EDE_CBC_SHA
- ECDHE_ECDSA_3DES_EDE_CBC_SHA256
- ECDHE_RSA_3DES_EDE_CBC_SHA256

IBM WebSphere MQ Version 7.1.0, Fix Pack 9: Support for class name whitelisting in IBM WebSphere MQ classes for JMS ObjectMessage

From Version 7.1.0, Fix Pack 9, IBM WebSphere 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 ClassName whitelisting in JMS ObjectMessage and Running WebSphere MQ classes for JMS applications under the Java security manager.

IBM WebSphere MQ Version 7.1.0, Fix Pack 9: New constant JMS_IBM_SUBSCRIPTION_USER_DATA added to the JmsConstants interface

From Version 7.1.0, Fix Pack 9, the IBM WebSphere 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 WebSphere 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 subscription user data and DEFINE SUB.

IBM WebSphere MQ Version 7.1.0, Fix Pack 9 plus interim fix for APAR IT26482: Change to authorities needed for IBM WebSphere MQ classes for JMS

For Version 7.1.0, Fix Pack 9 plus an interim fix for APAR IT26482, the IBM WebSphere MQ classes for JMS have been updated so that only inquire access is required to query the **BackoutThreshold** and **BackoutRequeueQName** of a cluster queue. For all other versions, browse and get access are also required.

For more information, see Handling poison messages in IBM WebSphere MQ classes for JMS.

Related concepts:

"What's new in IBM WebSphere MQ Version 7.1" on page 8

"What's changed in IBM WebSphere MQ Version 7.1" on page 17

Related information:

IBM WebSphere MQ requirements

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

Recommended Fixes for WebSphere MQ

WebSphere MQ planned maintenance release dates

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 WebSphere MQ Version 7.1.

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.

For IBM WebSphere MQ Version 7.0.1 and earlier, the documentation is provided outside of the online IBM Knowledge Center. For more information, see Documentation for older versions of IBM MQ.

Related concepts:

"What's new in IBM WebSphere MQ Version 7.1" on page 8

"What's changed in IBM WebSphere MQ Version 7.1" on page 17

"What's changed in IBM WebSphere MQ Version 7.1 Fix Packs" on page 55

Mappings from the old IBM WebSphere MQ books to the new product documentation

This section provides a mapping from the old IBM WebSphere MQ books to the new product documentation structure:

- “Quick beginnings for AIX” on page 59
- “Quick beginnings for HP-UX” on page 59
- “Quick beginnings for IBM i” on page 60
- “Quick beginnings for Linux” on page 60
- “Quick beginnings for Solaris” on page 61
- “Quick beginnings for Windows” on page 61
- “Application programming guide” on page 62
- “Application programming reference” on page 62
- “Clients” on page 63
- “Constants” on page 64
- “Intercommunication” on page 64
- “Messages and codes” on page 65
- “Migration” on page 66
- “Monitoring” on page 66
- “Programmable Command Formats and Administration Interface” on page 66
- “Publish/Subscribe User's Guide” on page 66
- “Queue manager clusters” on page 67
- “Script (MQSC) Command Reference” on page 67
- “Security” on page 67
- “System Administration Guide” on page 67
- “Using .NET” on page 68
- “Using C++” on page 68
- “Using Java” on page 69
- “Web services” on page 69
- “Using the Component Object Model Interface” on page 69
- “z/OS Concepts and Planning Guide” on page 69
- “z/OS Messages and Codes” on page 69
- “z/OS Problem Determination Guide” on page 70
- “z/OS System Administration Guide” on page 70
- “z/OS System Setup Guide” on page 71
- “IBM i System Administration Guide” on page 71
- “IBM i Application Programming Reference (ILE RPG)” on page 72
- “Glossary” on page 306

Related concepts:

"WebSphere MQ Version 7.1 PDF documentation" on page 8

"IBM WebSphere MQ information roadmap" on page 5

Quick beginnings for AIX

This section provides a mapping from the old Quick beginnings for AIX book to the new product documentation structure:

- AIX: Planning for migration from IBM WebSphere MQ Version 7.0.1 to IBM WebSphere MQ Version 7.1 (*WebSphere MQ V7.1 Installing Guide*)
- Checking requirements (*WebSphere MQ V7.1 Installing Guide*)
- Preparing the system (*WebSphere MQ V7.1 Installing Guide*)
- Installing IBM WebSphere MQ server on AIX (*WebSphere MQ V7.1 Installing Guide*)
- Verifying a server installation (*WebSphere MQ V7.1 Installing Guide*)
- Installing a IBM WebSphere MQ client on AIX systems (*WebSphere MQ V7.1 Installing Guide*)
- Verifying a client installation (*WebSphere MQ V7.1 Installing Guide*)
- AIX: Applying maintenance level upgrades on IBM WebSphere MQ Version 7.1 (*WebSphere MQ V7.1 Installing Guide*)
- Uninstalling IBM WebSphere MQ on AIX (*WebSphere MQ V7.1 Installing Guide*)

Quick beginnings for HP-UX

This section provides a mapping from the old Quick beginnings for HP-UX book to the new product documentation structure:

- HP-UX: Planning for migration from IBM WebSphere MQ Version 7.0.1 to IBM WebSphere MQ Version 7.1 (*WebSphere MQ V7.1 Installing Guide*)
- Checking requirements (*WebSphere MQ V7.1 Installing Guide*)
- Preparing the system (*WebSphere MQ V7.1 Installing Guide*)
- Installing IBM WebSphere MQ server on HP-UX (*WebSphere MQ V7.1 Installing Guide*)
- Verifying a server installation (*WebSphere MQ V7.1 Installing Guide*)
- Installing IBM WebSphere MQ client on HP-UX (*WebSphere MQ V7.1 Installing Guide*)
- Verifying a client installation (*WebSphere MQ V7.1 Installing Guide*)
- HP-UX: Applying maintenance level updates on IBM WebSphere MQ Version 7.1 (*WebSphere MQ V7.1 Installing Guide*)
- Uninstalling IBM WebSphere MQ on HP-UX (*WebSphere MQ V7.1 Installing Guide*)

Quick beginnings for IBM i

This section provides a mapping from the old Quick beginnings for IBM i book to the new product documentation structure:

- Checking requirements (*WebSphere MQ V7.1 Installing Guide*)
- Preparing the system (*WebSphere MQ V7.1 Installing Guide*)
- Installing IBM WebSphere MQ server on IBM i (*WebSphere MQ V7.1 Installing Guide*)
- Verifying a server installation (*WebSphere MQ V7.1 Installing Guide*)
- Installing IBM WebSphere MQ client on IBM i (*WebSphere MQ V7.1 Installing Guide*)
- Installing IBM WebSphere MQ Java messaging and web services for IBM i (*WebSphere MQ V7.1 Installing Guide*)
- Verifying a client installation (*WebSphere MQ V7.1 Installing Guide*)
- IBM i: Applying maintenance level updates on IBM WebSphere MQ Version 7.1 (*WebSphere MQ V7.1 Installing Guide*)
- IBM i: Applying maintenance level updates on IBM WebSphere MQ Version 7.1 (*WebSphere MQ V7.1 Installing Guide*)
- Uninstalling IBM WebSphere MQ for IBM i, Version 7.1 (*WebSphere MQ V7.1 Installing Guide*)

Quick beginnings for Linux

This section provides a mapping from the old Quick beginnings for Linux book to the new product documentation structure:

- Linux: Planning for migration from IBM WebSphere MQ Version 7.0.1 to IBM WebSphere MQ Version 7.1 (*WebSphere MQ V7.1 Installing Guide*)
- Checking requirements (*WebSphere MQ V7.1 Installing Guide*)
- Preparing the system (*WebSphere MQ V7.1 Installing Guide*)
- Installing IBM WebSphere MQ server on Linux (*WebSphere MQ V7.1 Installing Guide*)
- Verifying a server installation (*WebSphere MQ V7.1 Installing Guide*)
- Installing WebSphere MQ client on Linux (*WebSphere MQ V7.1 Installing Guide*)
- Verifying a client installation (*WebSphere MQ V7.1 Installing Guide*)
- Linux: Applying maintenance level updates on IBM WebSphere MQ Version 7.1 (*WebSphere MQ V7.1 Installing Guide*)
- Uninstalling IBM WebSphere MQ on Linux (*WebSphere MQ V7.1 Installing Guide*)

Quick beginnings for Solaris

This section provides a mapping from the old Quick beginnings for Solaris book to the new product documentation structure:

- Solaris: Planning for migration from IBM WebSphere MQ Version 7.0.1 to IBM WebSphere MQ Version 7.1 (*WebSphere MQ V7.1 Installing Guide*)
- Checking requirements (*WebSphere MQ V7.1 Installing Guide*)
- Preparing the system (*WebSphere MQ V7.1 Installing Guide*)
- Installing IBM WebSphere MQ server on Solaris (*WebSphere MQ V7.1 Installing Guide*)
- Verifying a server installation (*WebSphere MQ V7.1 Installing Guide*)
- Installing IBM WebSphere MQ client on Solaris (*WebSphere MQ V7.1 Installing Guide*)
- Verifying a client installation (*WebSphere MQ V7.1 Installing Guide*)
- Solaris: Applying maintenance level updates on IBM WebSphere MQ Version 7.1 (*WebSphere MQ V7.1 Installing Guide*)
- Uninstalling IBM WebSphere MQ on Solaris (*WebSphere MQ V7.1 Installing Guide*)

Quick beginnings for Windows

This section provides a mapping from the old Quick beginnings for Windows book to the new product documentation structure:

- Windows: Planning for migration from IBM WebSphere MQ Version 7.0.1 to IBM WebSphere MQ Version 7.1 (*WebSphere MQ V7.1 Installing Guide*)
- Checking requirements (*WebSphere MQ V7.1 Installing Guide*)
- Preparing the system (*WebSphere MQ V7.1 Installing Guide*)
- Installing IBM WebSphere MQ server on Windows (*WebSphere MQ V7.1 Installing Guide*)
- Verifying a server installation (*WebSphere MQ V7.1 Installing Guide*)
- Installing IBM WebSphere MQ client on Windows systems (*WebSphere MQ V7.1 Installing Guide*)
- Verifying a client installation (*WebSphere MQ V7.1 Installing Guide*)
- Windows: Applying maintenance level upgrades on IBM WebSphere MQ Version 7.1 (*WebSphere MQ V7.1 Installing Guide*)
- Uninstalling IBM WebSphere MQ on Solaris (*WebSphere MQ V7.1 Installing Guide*)

Application programming guide

This section provides a mapping from the old Application programming guide book to the new product documentation structure:

- Application development concepts (*WebSphere MQ V7.1 Programming Guide*)
- Designing IBM WebSphere MQ applications (*WebSphere MQ V7.1 Programming Guide*)
- Writing a queuing application (*WebSphere MQ V7.1 Programming Guide*)
- Sample WebSphere MQ programs (*WebSphere MQ V7.1 Programming Guide*)
- C language examples (*WebSphere MQ V7.1 Reference*)
- COBOL examples (*WebSphere MQ V7.1 Reference*)
- System/390 assembler-language examples (*WebSphere MQ V7.1 Reference*)
- PL/I examples (*WebSphere MQ V7.1 Reference*)
- WebSphere MQ data definition files (*WebSphere MQ V7.1 Programming Guide*)
- Coding standards on 64-bit platforms (*WebSphere MQ V7.1 Reference*)

Application programming reference

This section provides a mapping from the old Application programming reference book to the new product documentation structure:

- Data types used in the MQI (*WebSphere MQ V7.1 Reference*)
- Function calls (*WebSphere MQ V7.1 Reference*)
- Attributes of objects (*WebSphere MQ V7.1 Reference*)
- Return codes (*WebSphere MQ V7.1 Reference*)
- Rules for validating MQI options (*WebSphere MQ V7.1 Reference*)
- Machine encodings (*WebSphere MQ V7.1 Reference*)
- Report options and message flags (*WebSphere MQ V7.1 Reference*)
- Data conversion (*WebSphere MQ V7.1 Reference*)
- Properties specified as MQRFH2 elements (*WebSphere MQ V7.1 Reference*)
- Code page conversion (*WebSphere MQ V7.1 Reference*)

Clients

This section provides a mapping from the old Clients book to the new product documentation structure:

- Overview of WebSphere MQ MQI clients
- Platform support for WebSphere MQ clients
- Installing a WebSphere MQ client (*WebSphere MQ V7.1 Installing Guide*)
- Configuring connections between the server and client (*WebSphere MQ V7.1 Installing Guide*)
- Configuring an extended transactional client (*WebSphere MQ V7.1 Installing Guide*)
- Verifying a client installation (*WebSphere MQ V7.1 Installing Guide*)
- Setting up WebSphere MQ MQI client security (*WebSphere MQ V7.1 Administering Guide*)
- MQI client: Default behavior of client-connection and server-connectionChannels
- Creating server-connection and client-connection definitions on different platforms (*WebSphere MQ V7.1 Installing Guide*)
- Creating server-connection and client-connection definitions on the server (*WebSphere MQ V7.1 Installing Guide*)
- Channel-exit programs for MQI channels (*WebSphere MQ V7.1 Installing Guide*)
- Connecting a client to a queue-sharing group (*WebSphere MQ V7.1 Installing Guide*)
- Configuring a client using a configuration file (*WebSphere MQ V7.1 Installing Guide*)
- Using WebSphere MQ environment variables (*WebSphere MQ V7.1 Installing Guide*)
- Using the message queue interface (MQI) in a client application (*WebSphere MQ V7.1 Installing Guide*)
- Building applications for WebSphere MQ MQI clients (*WebSphere MQ V7.1 Programming Guide*)
- Running applications in the WebSphere MQ MQI client environment (*WebSphere MQ V7.1 Programming Guide*)
- Preparing and running CICS and Tuxedo applications (*WebSphere MQ V7.1 Programming Guide*)
- Preparing and running Microsoft Transaction Server applications (*WebSphere MQ V7.1 Programming Guide*)
- Preparing and running WebSphere MQ JMS applications (*WebSphere MQ V7.1 Programming Guide*)

  Preparing and running WebSphere MQ JMS applications (*WebSphere MQ V7.1 Programming Guide*)
- Resolving problems with WebSphere MQ MQI clients (*WebSphere MQ V7.1 Administering Guide*)
- Referencing connection definitions using a pre-connect exit from a repository (*WebSphere MQ V7.1 Programming Guide*)

Constants

This section provides a mapping from the old Constants book to the new product documentation structure:

- WebSphere MQ COPY, header, include, and module files (*WebSphere MQ V7.1 Reference*)
- Constants (*WebSphere MQ V7.1 Reference*)

Intercommunication

This section provides a mapping from the old Intercommunication book to the new product documentation structure:

Introduction

- Concepts of intercommunication
- Connecting applications using distributed queuing (*WebSphere MQ V7.1 Installing Guide*)
- Networks and Network Planning (*WebSphere MQ V7.1 Installing Guide*)
- WebSphere MQ distributed-messaging techniques (*WebSphere MQ V7.1 Installing Guide*)
- Introduction to distributed queue management (*WebSphere MQ V7.1 Installing Guide*)
- Channel attributes (*WebSphere MQ V7.1 Reference*)
- Example configuration information (*WebSphere MQ V7.1 Reference*)

Distributed queue management in WebSphere MQ for Windows and UNIX platforms

- Monitoring and controlling channels on Windows, UNIX and Linux platforms (*WebSphere MQ V7.1 Installing Guide*)
- Creating a transmission queue (*WebSphere MQ V7.1 Installing Guide*)
- Triggering channels (*WebSphere MQ V7.1 Installing Guide*)
- Channel programs (*WebSphere MQ V7.1 Reference*)
- Security for remote messaging (*WebSphere MQ V7.1 Administering Guide*)
- Other things to consider for distributed queue management (*WebSphere MQ V7.1 Installing Guide*)
- Setting up communication for Windows (*WebSphere MQ V7.1 Installing Guide*)
- Example configuration - IBM WebSphere MQ for Windows (*WebSphere MQ V7.1 Reference*)
- Example configuration - IBM WebSphere MQ for AIX (*WebSphere MQ V7.1 Reference*)
- Example configuration - IBM WebSphere MQ for HP-UX (*WebSphere MQ V7.1 Reference*)
- Example configuration - IBM WebSphere MQ for Solaris (*WebSphere MQ V7.1 Reference*)
- Example configuration - IBM WebSphere MQ for Linux (*WebSphere MQ V7.1 Reference*)
- Message channel planning example for distributed platforms (*WebSphere MQ V7.1 Reference*)

Distributed queue management in WebSphere MQ for z/OS

- Monitoring and controlling channels on z/OS (*WebSphere MQ V7.1 Installing Guide*)
- Preparing WebSphere MQ for z/OS for distributed queuing (*WebSphere MQ V7.1 Installing Guide*)
- Setting up communication for z/OS (*WebSphere MQ V7.1 Installing Guide*)
- Example configuration - IBM WebSphere MQ for z/OS (*WebSphere MQ V7.1 Reference*)
- Message channel planning example for z/OS (*WebSphere MQ V7.1 Reference*)
- Preparing WebSphere MQ for z/OS for DQM with queue-sharing groups (*WebSphere MQ V7.1 Installing Guide*)
- Setting up communication for WebSphere MQ for z/OS using queue-sharing groups (*WebSphere MQ V7.1 Installing Guide*)
- Example configuration - IBM WebSphere MQ for z/OS using queue-sharing groups (*WebSphere MQ V7.1 Reference*)
- Message channel planning example for z/OS using queue-sharing groups (*WebSphere MQ V7.1 Reference*)
- Intra-group queuing
- Example configuration — WebSphere MQ for z/OS using intra-group queuing (*WebSphere MQ V7.1 Reference*)

Distributed queue management in WebSphere MQ for IBM i

- Monitoring and controlling channels on IBM i (*WebSphere MQ V7.1 Installing Guide*)
- Setting up communication for WebSphere MQ for IBM i (*WebSphere MQ V7.1 Installing Guide*)
- Example configuration - IBM WebSphere MQ for IBM i (*WebSphere MQ V7.1 Reference*)
- Message channel planning example for WebSphere MQ for IBM i (*WebSphere MQ V7.1 Reference*)

Further intercommunication considerations

- Channel-exit programs for messaging channels (*WebSphere MQ V7.1 Programming Guide*)
- Channel-exit calls and data structures (*WebSphere MQ V7.1 Reference*)
- Resolving problems with channels and DQM (*WebSphere MQ V7.1 Administering Guide*)
- Queue name resolution (*WebSphere MQ V7.1 Reference*)

Messages and codes

This section provides a mapping from the old Messages and codes book to the new product documentation structure:

- Diagnostic messages: AMQ4000-9999 (*WebSphere MQ V7.1 Reference*)
- API completion and reason codes (*WebSphere MQ V7.1 Administering Guide*)
- PCF reason codes (*WebSphere MQ V7.1 Administering Guide*)
- Secure Sockets Layer (SSL) and Transport Layer Security (TLS) return codes (*WebSphere MQ V7.1 Administering Guide*)

- WCF custom channel exceptions (*WebSphere MQ V7.1 Administering Guide*)

Migration

The structure of the migration section in Version 7.1 remains the same as in Version 7.0.1.

Topics have been added for migration to Version 7.1, and removed for migration to Version 7.0.1. To refer to migration to Version 7.0.1 and earlier releases, refer to Where to find a topic about a specific migration path (*WebSphere MQ V7.1 Installing Guide*).

Monitoring

This section provides a mapping from the old Monitoring book to the new product documentation structure:

- Event monitoring (*WebSphere MQ V7.1 Administering Guide*)
- Message monitoring (*WebSphere MQ V7.1 Administering Guide*)
- Accounting and statistics messages (*WebSphere MQ V7.1 Administering Guide*)
- Real-time monitoring (*WebSphere MQ V7.1 Administering Guide*)
- Structure data types (*WebSphere MQ V7.1 Reference*)
- Object attributes for event data (*WebSphere MQ V7.1 Reference*)

Programmable Command Formats and Administration Interface

This section provides a mapping from the old Programmable Command Formats and Administration Interface book to the new product documentation structure:

- Introduction to Programmable Command Formats (*WebSphere MQ V7.1 Administering Guide*)
- Introduction to the WebSphere MQ Administration Interface (MQAI) (*WebSphere MQ V7.1 Administering Guide*)

Publish/Subscribe User's Guide

This section provides a mapping from the old Publish/Subscribe User's Guide book to the new product documentation structure:

- Introduction to WebSphere MQ publish/subscribe messaging (*WebSphere MQ V7.1 Installing Guide*)
- Distributed publish/subscribe (*WebSphere MQ V7.1 Installing Guide*)
- Writing publish/subscribe applications (*WebSphere MQ V7.1 Programming Guide*)
- Publish/subscribe security (*WebSphere MQ V7.1 Administering Guide*)
- Publish/Subscribe migration from version 6.0 (*WebSphere MQ V7.1 Installing Guide*)
- Migration of the publish/subscribe broker in WebSphere Event Broker and WebSphere Message Broker (*WebSphere MQ V7.1 Installing Guide*)

Queue manager clusters

This section provides a mapping from the old Queue manager clusters book to the new product documentation structure:

- "How clusters work" on page 96
- Configuring a queue manager cluster (*WebSphere MQ V7.1 Installing Guide*)
- Managing IBM WebSphere MQ clusters (*WebSphere MQ V7.1 Installing Guide*)
- Routing messages to and from clusters (*WebSphere MQ V7.1 Installing Guide*)
- Using clusters for workload management (*WebSphere MQ V7.1 Installing Guide*)
- Keeping clusters secure (*WebSphere MQ V7.1 Administering Guide*)
- Working with the MQI and clusters (*WebSphere MQ V7.1 Programming Guide*)
- WebSphere MQ cluster commands (*WebSphere MQ V7.1 Reference*)
- Resolving problems with queue manager clusters (*WebSphere MQ V7.1 Administering Guide*)

Script (MQSC) Command Reference

This section provides a mapping from the old Script (MQSC) Command Reference book to the new product documentation structure:

- Script (MQSC) Commands (*WebSphere MQ V7.1 Administering Guide*)
- Generic values and characters with special meanings (*WebSphere MQ V7.1 Reference*)
- Building command scripts (*WebSphere MQ V7.1 Reference*)
- Using commands on z/OS (*WebSphere MQ V7.1 Reference*)
- "Rules for naming IBM WebSphere MQ objects" on page 137
- Syntax diagrams (*WebSphere MQ V7.1 Reference*)
- The MQSC commands (*WebSphere MQ V7.1 Reference*)

Security

This section provides a mapping from the old Security book to the new product documentation structure:

- Security (*WebSphere MQ V7.1 Administering Guide*)

System Administration Guide

This section provides a mapping from the old System Administration Guide book to the new product documentation structure:

- "IBM WebSphere MQ Technical overview" on page 72
- Administering IBM WebSphere MQ (*WebSphere MQ V7.1 Administering Guide*)
- Administering local WebSphere MQ objects (*WebSphere MQ V7.1 Administering Guide*)
- Administration using the IBM WebSphere MQ Explorer (*WebSphere MQ V7.1 Administering Guide*)
- Using the WebSphere MQ Taskbar application (Windows only) (*WebSphere MQ V7.1 Administering Guide*)

- WebSphere MQ Control commands (*WebSphere MQ V7.1 Reference*)

Configuration and management

- Changing IBM WebSphere MQ and queue manager configuration information (*WebSphere MQ V7.1 Installing Guide*)
- Planning file system support (*WebSphere MQ V7.1 Installing Guide*)
- Setting up security on Windows, UNIX and Linux systems (*WebSphere MQ V7.1 Administering Guide*)
- Transactional support (*WebSphere MQ V7.1 Programming Guide*)
- Handling undelivered messages with the WebSphere MQ dead-letter queue handler (*WebSphere MQ V7.1 Installing Guide*)
- Availability, recovery and restart (*WebSphere MQ V7.1 Installing Guide*)
- Troubleshooting and support (*WebSphere MQ V7.1 Administering Guide*)
- WebSphere MQ and UNIX System V IPC resources (*WebSphere MQ V7.1 Installing Guide*)
- WebSphere MQ and UNIX Process Priority (*WebSphere MQ V7.1 Installing Guide*)
- User exits, API exits, and WebSphere MQ installable services (*WebSphere MQ V7.1 Programming Guide*)

Using .NET

This section provides a mapping from the old Using .NET book to the new product documentation structure:

- Using .NET (*WebSphere MQ V7.1 Programming Guide*)
- Writing and deploying WebSphere MQ .NET programs (*WebSphere MQ V7.1 Programming Guide*)
- The WebSphere MQ .NET classes and interfaces (*WebSphere MQ V7.1 Reference*)
- IBM WebSphere MQ custom channel for Microsoft Windows Communication Foundation (WCF) (*WebSphere MQ V7.1 Programming Guide*)

Using C++

This section provides a mapping from the old Using C++ book to the new product documentation structure:

- Using C++ (*WebSphere MQ V7.1 Programming Guide*)
- WebSphere MQ C++ classes (*WebSphere MQ V7.1 Reference*)

Using Java

This section provides a mapping from the old Using Java book to the new product documentation structure:

- Should I use IBM WebSphere MQ classes for Java or IBM WebSphere MQ classes for JMS? (*WebSphere MQ V7.1 Programming Guide*)
- Using WebSphere MQ classes for Java (*WebSphere MQ V7.1 Programming Guide*)
- WebSphere MQ classes for JMS
- Using WebSphere MQ classes for JMS (*WebSphere MQ V7.1 Programming Guide*)
- WebSphere MQ classes for Java

Web services

This section provides a mapping from the old Web services to the new product documentation structure:

- WebSphere MQ transport for SOAP (*WebSphere MQ V7.1 Programming Guide*)
- WebSphere MQ bridge for HTTP (*WebSphere MQ V7.1 Programming Guide*)

Using the Component Object Model Interface

This section provides a mapping from the old Using the Component Object Model Interface book to the new product documentation structure:

- Using the Component Object Model Interface (WebSphere MQ Automation Classes for ActiveX) (*WebSphere MQ V7.1 Programming Guide*)

z/OS Concepts and Planning Guide

This section provides a mapping from the old z/OS Concepts and Planning Guide book to the new product documentation structure:

- “WebSphere MQ for z/OS concepts” on page 178
- “IBM WebSphere MQ and other z/OS products” on page 299
- Planning your IBM WebSphere MQ environment on z/OS (*WebSphere MQ V7.1 Installing Guide*)
- Planning to install WebSphere MQ (*WebSphere MQ V7.1 Installing Guide*)
- Macros intended for customer use (*WebSphere MQ V7.1 Installing Guide*)
- Sub-capacity license charges with WebSphere MQ for z/OS (*WebSphere MQ V7.1 Installing Guide*)

z/OS Messages and Codes

This section provides a mapping from the old z/OS Messages and Codes book to the new product documentation structure:

- Messages for WebSphere MQ for z/OS (*WebSphere MQ V7.1 Reference*)
- WebSphere MQ for z/OS codes (*WebSphere MQ V7.1 Reference*)
- WebSphere MQ CICS abend codes (*WebSphere MQ V7.1 Reference*)
- API completion and reason codes (*WebSphere MQ V7.1 Administering Guide*)
- PCF reason codes (*WebSphere MQ V7.1 Administering Guide*)

- WebSphere MQ component identifiers (*WebSphere MQ V7.1 Reference*)
- Communications protocol return codes (*WebSphere MQ V7.1 Reference*)
- Secure Sockets Layer (SSL) and Transport Layer Security (TLS) return codes for z/OS (*WebSphere MQ V7.1 Reference*)
- Distributed queuing message codes (*WebSphere MQ V7.1 Reference*)
- Publish/subscribe reason codes (*WebSphere MQ V7.1 Reference*)
- Messages from other products (*WebSphere MQ V7.1 Reference*)

z/OS Problem Determination Guide

This section provides a mapping from the old z/OS Problem Determination Guide book to the new product documentation structure:

- Troubleshooting overview (*WebSphere MQ V7.1 Administering Guide*)
- Dealing with program abends (*WebSphere MQ V7.1 Administering Guide*)
- Dealing with applications that are running slowly or have stopped on z/OS (*WebSphere MQ V7.1 Administering Guide*)
- Dealing with performance problems on z/OS (*WebSphere MQ V7.1 Administering Guide*)
- Dealing with incorrect output (*WebSphere MQ V7.1 Administering Guide*)
- Problem determination on z/OS (*WebSphere MQ V7.1 Administering Guide*)
- WebSphere MQ dumps (*WebSphere MQ V7.1 Administering Guide*)
- Using trace for problem determination on z/OS (*WebSphere MQ V7.1 Administering Guide*)
- Searching the IBM database for similar problems, and solutions (*WebSphere MQ V7.1 Administering Guide*)
- Contacting IBM Software Support (*WebSphere MQ V7.1 Administering Guide*)
- Resolving a problem (*WebSphere MQ V7.1 Administering Guide*)

z/OS System Administration Guide

This section provides a mapping from the old z/OS System Administration Guide book to the new product documentation structure:

- Operating WebSphere MQ for z/OS (*WebSphere MQ V7.1 Administering Guide*)
- "WebSphere MQ and CICS" on page 299
- WebSphere MQ and IMS (*WebSphere MQ V7.1 Administering Guide*)
- Managing WebSphere MQ resources on z/OS (*WebSphere MQ V7.1 Administering Guide*)
- Recovery and restart (*WebSphere MQ V7.1 Administering Guide*)
- Using the WebSphere MQ Utilities (*WebSphere MQ V7.1 Reference*)
- Using the WebSphere MQ Utilities (*WebSphere MQ V7.1 Reference*)

z/OS System Setup Guide

This section provides a mapping from the old z/OS System Setup Guide book to the new product documentation structure:

- Preparing to customize your queue managers (*WebSphere MQ V7.1 Installing Guide*)
- Customizing your queue managers (*WebSphere MQ V7.1 Installing Guide*)
- Migration paths: IBM WebSphere MQ for z/OS
- Testing your queue manager on z/OS (*WebSphere MQ V7.1 Installing Guide*)
- Using IBM WebSphere MQ with CICS (*WebSphere MQ V7.1 Installing Guide*)
- Using WebSphere MQ with IMS (*WebSphere MQ V7.1 Installing Guide*)
- Monitoring performance and resource usage (*WebSphere MQ V7.1 Administering Guide*)
- Setting up security on z/OS (*WebSphere MQ V7.1 Administering Guide*)
- Upgrading and applying service to Language Environment or z/OS Callable Services (*WebSphere MQ V7.1 Installing Guide*)
- Using OTMA exits in IMS (*WebSphere MQ V7.1 Installing Guide*)

IBM i System Administration Guide

This section provides a mapping from the old IBM i System Administration Guide book to the new product documentation structure:

- Managing WebSphere MQ for IBM i using CL commands (*WebSphere MQ V7.1 Administering Guide*)
- Alternative ways of administering WebSphere MQ for IBM i (*WebSphere MQ V7.1 Administering Guide*)
- Work management (*WebSphere MQ V7.1 Administering Guide*)
- Setting up security on IBM i (*WebSphere MQ V7.1 Administering Guide*)
- The WebSphere MQ for IBM i dead-letter queue handler (*WebSphere MQ V7.1 Installing Guide*)
- Availability, backup, recovery, and restart (*WebSphere MQ V7.1 Administering Guide*)
- Determining problems with IBM WebSphere MQ for IBM i applications (*WebSphere MQ V7.1 Administering Guide*)
- Changing configuration information on IBM i (*WebSphere MQ V7.1 Installing Guide*)
- Installable services and components for IBM i (*WebSphere MQ V7.1 Programming Guide*)
- “Understanding WebSphere MQ for IBM i queue manager library names” on page 141
- IBM WebSphere MQ for IBM i system and default objects (*WebSphere MQ V7.1 Reference*)
- Quiescing WebSphere MQ for IBM i (*WebSphere MQ V7.1 Administering Guide*)

IBM i Application Programming Reference (ILE RPG)

This section provides a mapping from the old IBM i Application Programming Reference (ILE RPG) book to the new product documentation structure:

- Data type descriptions (*WebSphere MQ V7.1 Reference*)
- Function calls (*WebSphere MQ V7.1 Reference*)
- Attributes of objects (*WebSphere MQ V7.1 Reference*)
- Applications (*WebSphere MQ V7.1 Reference*)
- Return codes for IBM i (ILE RPG) (*WebSphere MQ V7.1 Reference*)
- Rules for validating MQI options for IBM i (ILE RPG) (*WebSphere MQ V7.1 Reference*)
- Machine encodings on IBM i (*WebSphere MQ V7.1 Reference*)
- Report options and message flags (*WebSphere MQ V7.1 Reference*)
- Data conversion (*WebSphere MQ V7.1 Reference*)

IBM WebSphere MQ Technical overview

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

IBM WebSphere 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 links to find out about message queuing and other features provided by IBM WebSphere MQ.

- "Introduction to message queuing" on page 73
- "Concepts of intercommunication" on page 81
- "IBM WebSphere MQ Telemetry" on page 100
- "Objects" on page 136
- "WebSphere MQ Multicast" on page 164
- "Security" on page 165
- "Clients and servers" on page 166
- "Transaction management and support" on page 174
- "Extending queue manager facilities" on page 175
- "WebSphere MQ for z/OS concepts" on page 178
- "IBM WebSphere MQ and other z/OS products" on page 299

Related concepts:

"Introduction to IBM WebSphere MQ" on page 1

Designing a IBM WebSphere MQ architecture (*WebSphere MQ V7.1 Installing Guide*)

Related reference:

"Main features and benefits of message queuing" on page 75

Introduction to message queuing

The WebSphere 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	Programs communicate by sending each other data in messages rather than calling each other directly.
Queuing	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 77). 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 an application must know information about that application before it can send a message to that application. For example, the 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 about publish/subscribe messaging, see Introduction to WebSphere MQ publish/subscribe messaging (*WebSphere MQ V7.1 Installing Guide*)

Benefits of message queuing to the application designer and developer

WebSphere 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, HP-UX and z/OS applications can communicate through WebSphere MQ for HP-UX and WebSphere 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.

WebSphere 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 (*WebSphere MQ V7.1 Programming Guide*).

Related concepts:

"Messages and queues" on page 80

"Objects" on page 136

Related reference:

"Main features and benefits of message queuing"

"Message queuing terminology" on page 77

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 5 on page 76).

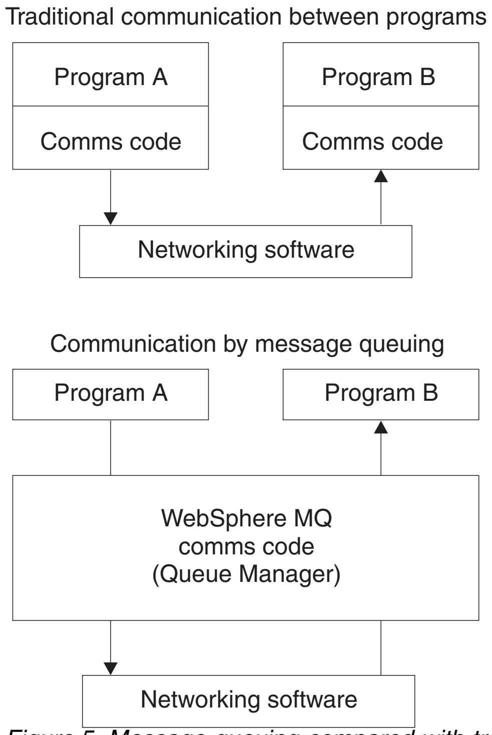


Figure 5. Message queuing compared with traditional communication

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

Authorization checks are carried out on each resource, using the tables that are set up and maintained by the WebSphere MQ administrator.

- Use Security Server (formerly known as RACF) or other external security managers on WebSphere MQ for z/OS.
- On WebSphere MQ on UNIX systems, Linux systems, Windows systems, and IBM i, a security manager called the object authority manager (OAM) is provided as an installable service. By default, the OAM is active.

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 WebSphere MQ depending on the form of sync point coordination selected for the application.

Recovery support

For recovery to be possible, all persistent WebSphere 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 (*WebSphere MQ V7.1 Programming Guide*).

Note: When considering WebSphere MQ clients and servers, you do not have to change a server application to support additional WebSphere MQ MQI clients on new platforms. Similarly, the WebSphere 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
- WebSphere MQ MQI client
- Point-to-point
- Publish/subscribe
- Subscription

Message

In message queuing, a message is a collection of data sent by one program and intended for another program. See WebSphere MQ messages (*WebSphere MQ V7.1 Programming Guide*) for more information. Refer to Types of message (*WebSphere MQ V7.1 Programming Guide*) for information about message types.

Message descriptor

A WebSphere 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 WebSphere 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 WebSphere 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. WebSphere 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 moves messages from one queue manager to another.

References are made to them when dealing with report messages and you need to consider them when designing your application. See Writing your own message channel agents (*WebSphere MQ V7.1 Installing Guide*) for more information.

Cluster

A *cluster* is a network of queue managers that are logically associated in some way. Clustering is available to queue managers on all WebSphere MQ V7.0 platforms.

In a WebSphere 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 “How clusters work” on page 96.

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 WebSphere MQ for 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 WebSphere MQ for z/OS. See “Shared queues and queue-sharing groups” on page 183 for more information about queue-sharing groups.

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 WebSphere MQ for z/OS.

WebSphere MQ MQI client

WebSphere MQ MQI *clients* are independently installable components of WebSphere MQ products. A client allows you to run WebSphere 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 WebSphere MQ MQI client components, see

Installing a IBM WebSphere MQ client (*WebSphere MQ V7.1 Installing Guide*) and

Configuring connections between the server and client (*WebSphere MQ V7.1 Installing Guide*).

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

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 about publish/subscribe,

see Introduction to WebSphere MQ publish/subscribe messaging (*WebSphere MQ V7.1 Installing Guide*).

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.

IBM WebSphere MQ messages have two parts:

- *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 WebSphere MQ. For a complete description of the message descriptor, see MQMD – Message descriptor (*WebSphere MQ V7.1 Reference*).

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 (*WebSphere MQ V7.1 Programming Guide*).

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 WebSphere 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.

Concepts of intercommunication

In WebSphere MQ, intercommunication 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 WebSphere MQ. This is called a *distributed* environment. WebSphere MQ handles communication in a distributed environment such as this using Distributed Queue Management (DQM).

The local queue manager is sometimes called the *source queue manager* and the remote queue manager is sometimes called the *target queue manager* or the *partner queue manager*.

How does distributed queuing work?

Distributed Queuing enables sending messages from one Queue Manager to another. The receiving Queue Manager could be on the same machine or a remote one. Queue Managers, Queues, Channels and associated Definitions are outlined, along with Clustering (a network of logically associated Queue Managers).

Figure 6 on page 82 shows an overview of the components of distributed queuing.

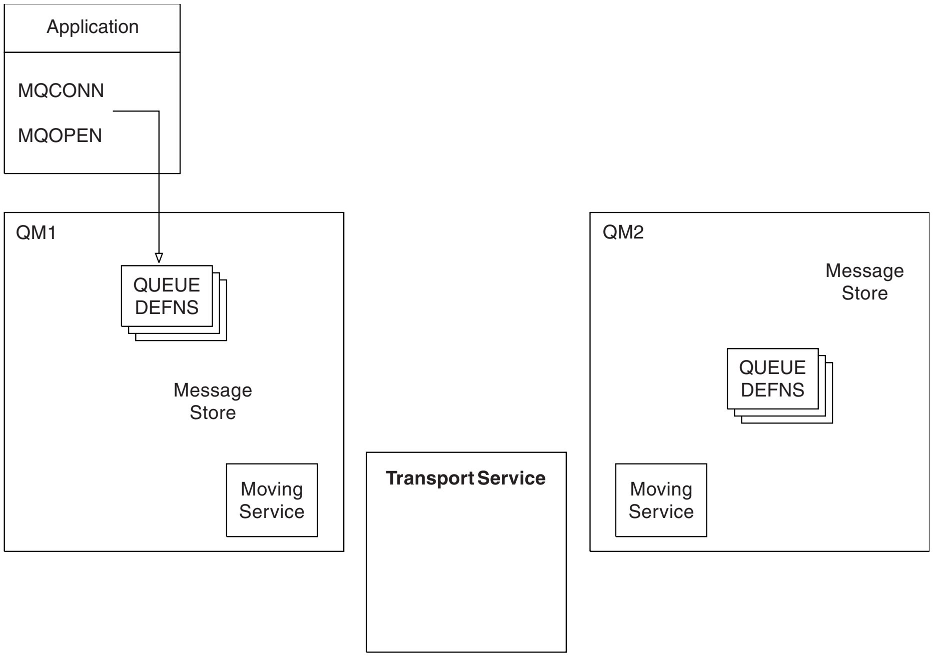


Figure 6. Overview of the components of distributed queuing

1. An application uses the MQCONN call to connect to a queue manager.
2. The application then uses the MQOPEN call to open a queue so that it can put messages on it.
3. A queue manager has a definition for each of its queues, specifying information such as the maximum number of messages allowed on the queue. It can also have definitions of queues located on remote queue managers.
4. If the messages are destined for a queue on a remote system, the local queue manager holds them in a message store until it is ready to forward them to the remote queue manager. This has no effect on the application.
5. Each queue manager contains communications software called the *moving service* component; through this, the queue manager can communicate with other queue managers.
6. 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)

What are the components of distributed queuing?

WebSphere MQ applications can put messages onto a local queue, that is, a queue on the queue manager the application is connected to.

A queue manager has a definition for each of its queues. It can also have definitions for queues that are owned by other queue managers. These are called *remote queue definitions*. WebSphere MQ applications can also put messages targeted at these remote queues.

If the messages are destined for a remote queue manager, the local queue manager stores them on a *transmission queue* until it is ready to send them to the remote queue manager. A transmission queue is a special type of local queue on which messages are stored until they can be successfully transmitted and stored at the remote queue manager.

The software that handles the sending and receiving of messages is called the *Message Channel Agent* (MCA).

Messages are transmitted between queue managers on 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.

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.

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 constitute one channel.

There is also 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.

Figure 7 shows the relationship between queue managers, transmission queues, channels, and MCAs.

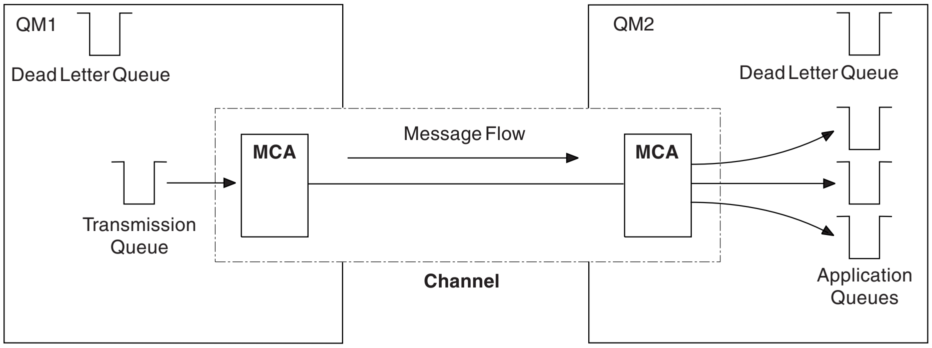


Figure 7. Sending messages

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 Figure 8 on page 84.

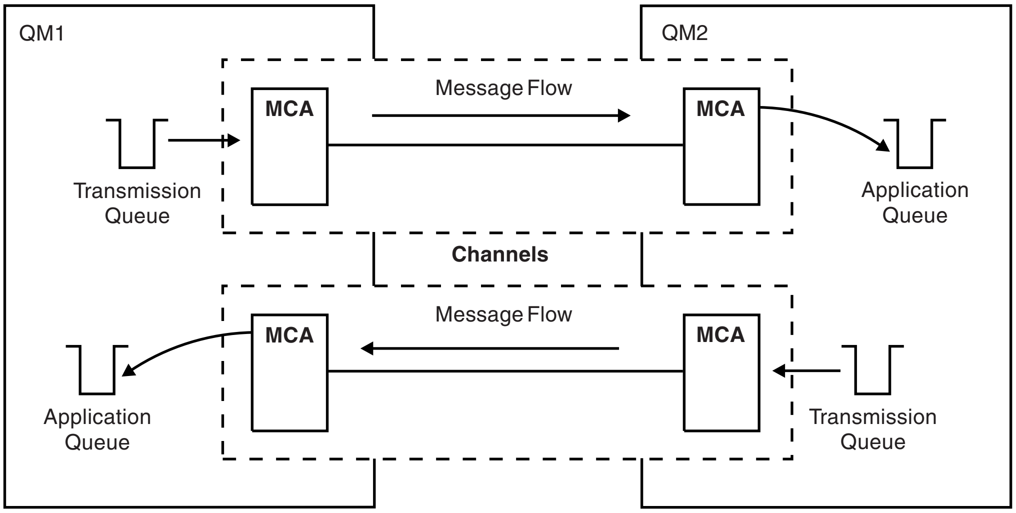


Figure 8. Sending messages in both directions

For more information about Distributed Queue Management, see Introduction to distributed queue management (*WebSphere MQ V7.1 Installing Guide*).

Cluster components

An alternative to the traditional WebSphere MQ network that is interconnected through manually defining channels is the use of clusters.

A cluster is a network of queue managers that are logically associated in some way. You can group queue managers in a cluster so that queue managers can make the queues that they host available to every other queue manager in the cluster. Assuming that you have the necessary network infrastructure in place, any queue manager can send a message to any other queue manager in the same 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. Each queue manager needs to define only one cluster-receiver channel and one cluster-sender channel, any additional channels are automatically managed by the cluster.

Figure 9 on page 85 shows the components of a cluster called CLUSTER:

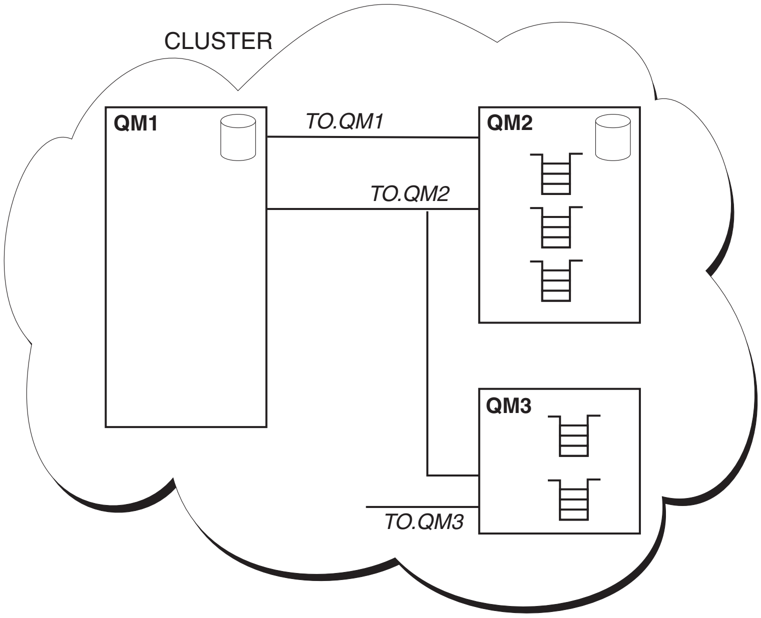


Figure 9. A cluster of queue managers

- 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.

As with 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.

For more information about clusters, see “Queue manager clusters” on page 67.

Related concepts:
“Distributed queuing components”
“Dead-letter queues” on page 89
“Remote queue definitions” on page 89
“How to get to the remote queue manager” on page 89
“Addressing information” on page 91
“What are aliases?” on page 92
“Queue manager alias definitions” on page 93
“Reply-to queue alias definitions” on page 94

Distributed queuing components

These are objects you need for enabling intercommunication.

The components of distributed queuing are:

- Message channels

- Message channel agents
- Transmission queues
- Channel initiators and listeners
- Channel-exit programs

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 about MQI channels, 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 (*WebSphere MQ V7.1 Installing Guide*) (not applicable to z/OS). For examples of the parameters needed to set up sender-receiver channels, see Example configuration information (*WebSphere MQ V7.1 Reference*) applicable to your platform. For the parameters needed to define a channel of any type, see DEFINE CHANNEL (*WebSphere MQ V7.1 Reference*).

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 10 on page 87 illustrates this.

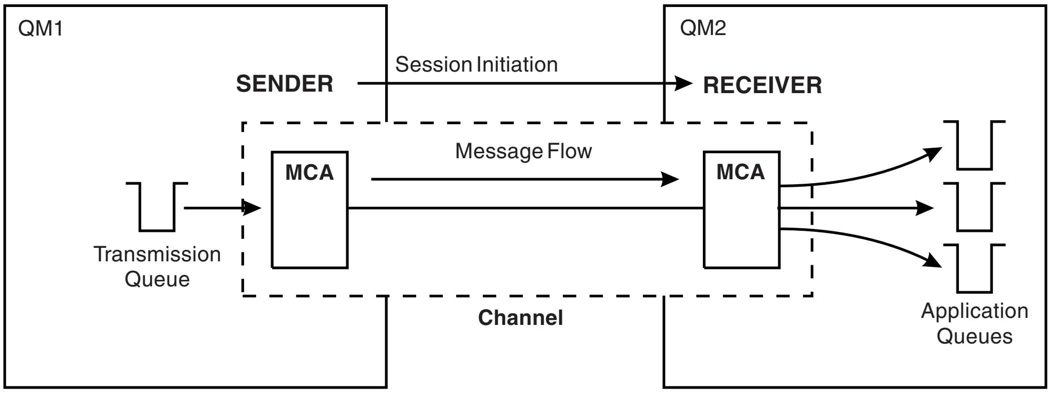


Figure 10. A sender-receiver channel

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.

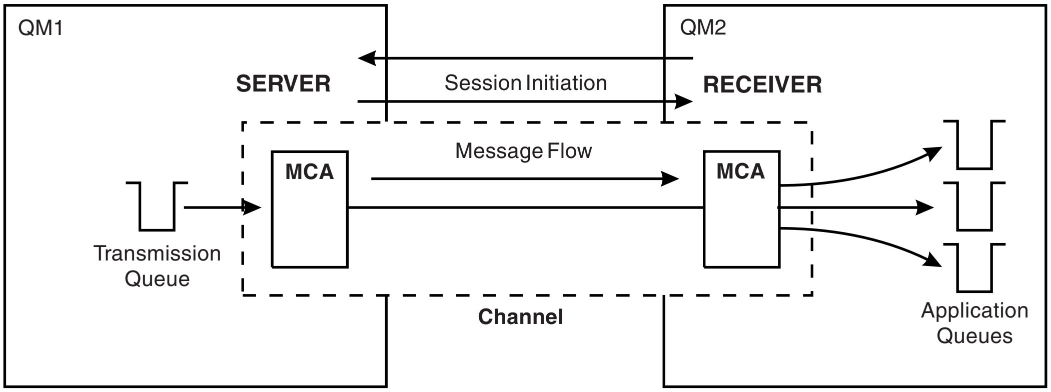


Figure 11. A requester-server channel

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.

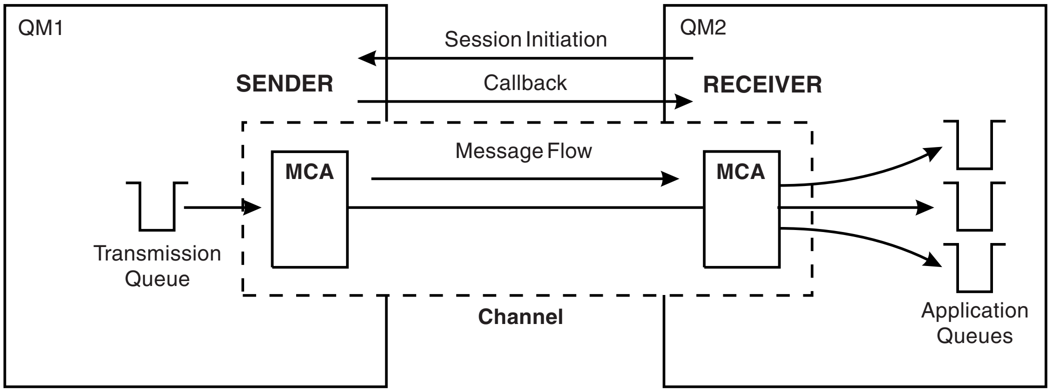


Figure 12. A requester-sender channel

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 10 on page 87.

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.

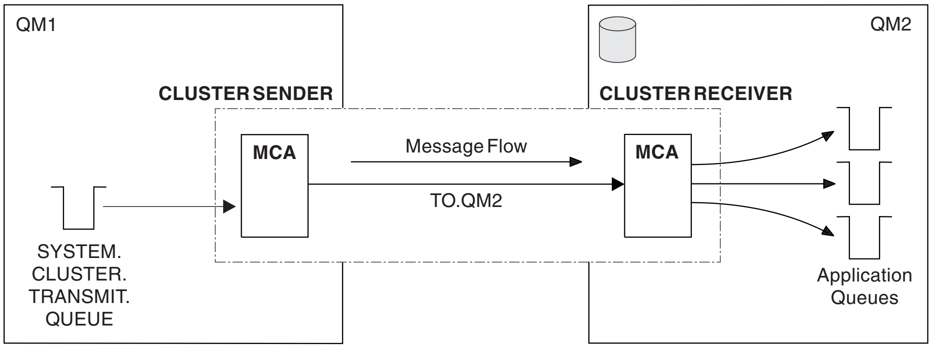


Figure 13. A cluster-sender channel

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 13.

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.

Messages are put on this queue when they cannot be put on the destination queue. 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.

Consider defining 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.

You can use the USEDLQ channel attribute to determine whether the dead-letter queue is used when messages cannot be delivered. This attribute can be configured so that some functions of the queue manager use the dead-letter queue, while other functions do not. For more information about the use of the USEDLQ channel attribute on different MQSC commands, see DEFINE CHANNEL (*WebSphere MQ V7.1 Reference*), DISPLAY CHANNEL (*WebSphere MQ V7.1 Reference*), ALTER CHANNEL (*WebSphere MQ V7.1 Reference*), and DISPLAY CLUSQMGR (*WebSphere MQ V7.1 Reference*).

Related reference:

Use Dead-Letter Queue (USEDLQ) (*WebSphere MQ V7.1 Reference*)

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 14 on page 90.

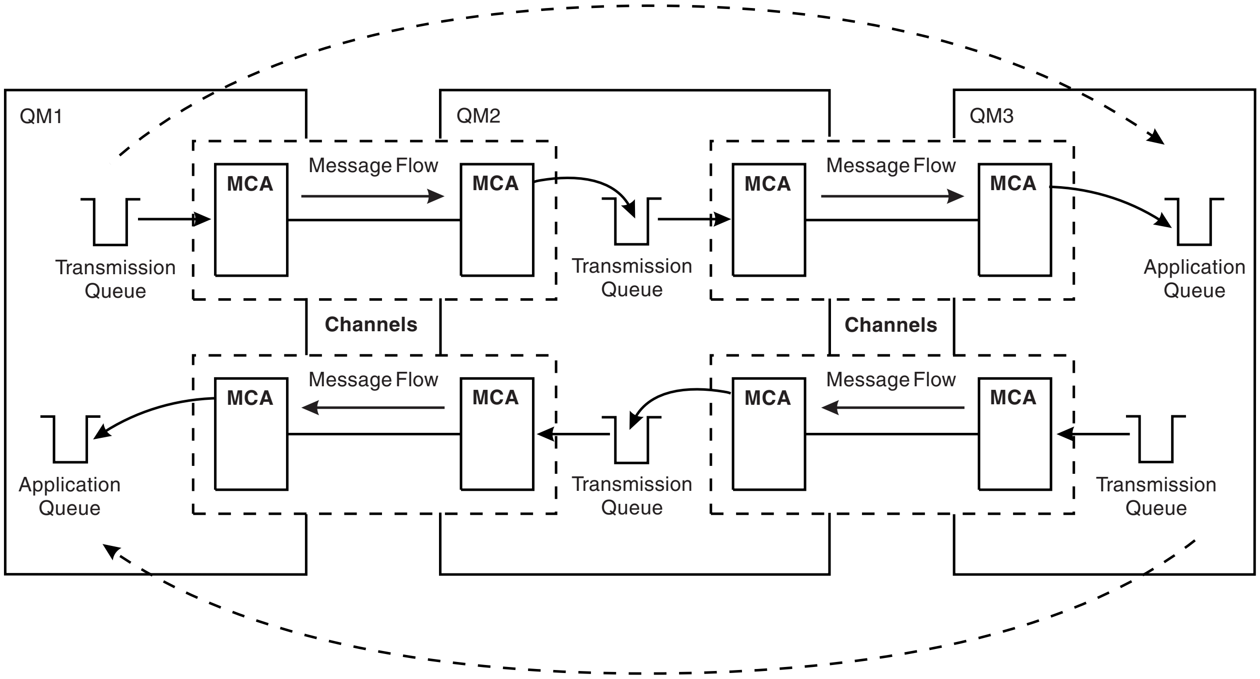


Figure 14. Passing through intermediate queue managers

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 15.

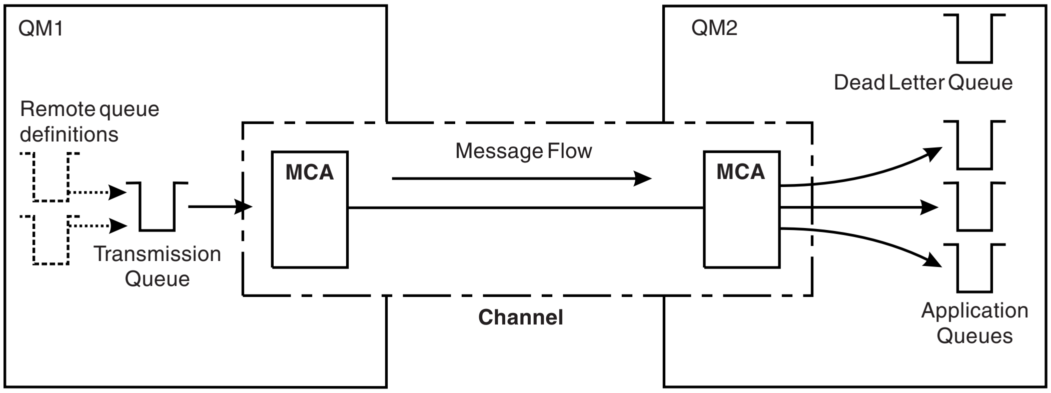


Figure 15. Sharing a transmission queue

Figure 15 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 16.

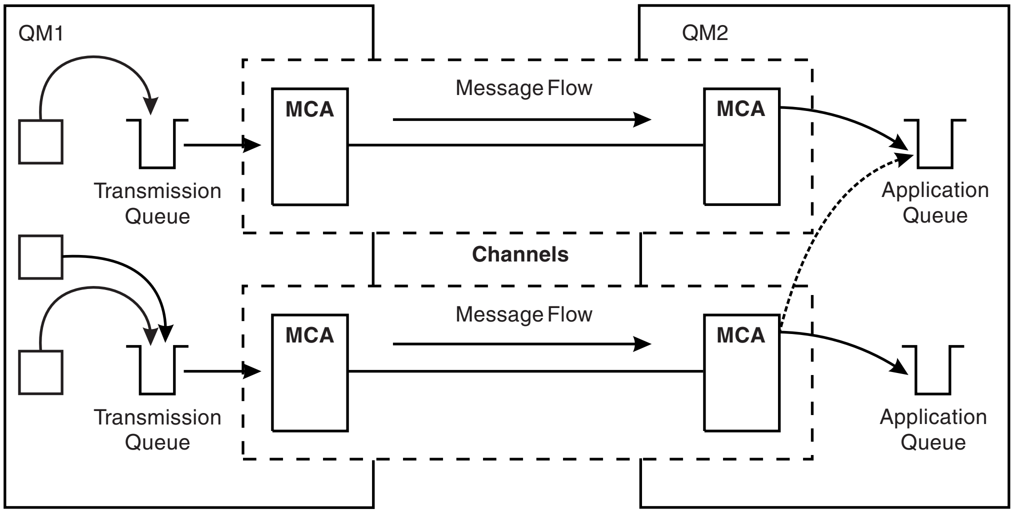


Figure 16. Using multiple channels

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. WebSphere MQ uses a workload management algorithm that uses a round-robin routine to select an available queue manager to route a message to. For more information see “Clusters” on page 156.

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 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 (*WebSphere MQ V7.1 Reference*).

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.

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 17 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.

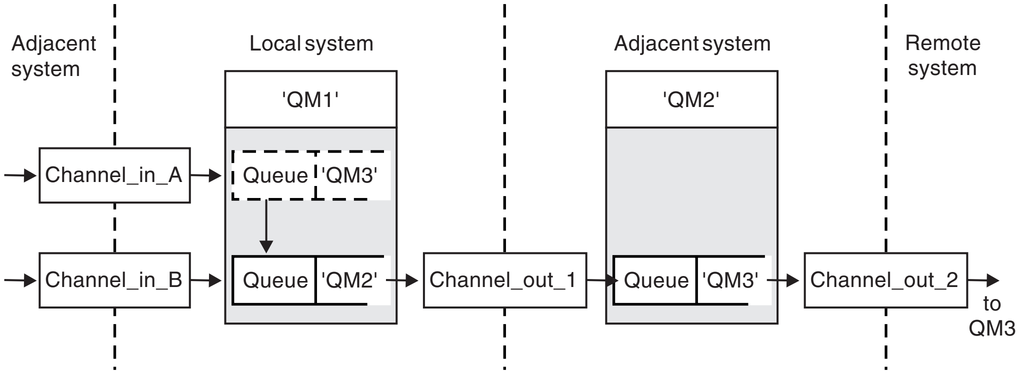


Figure 17. Queue manager alias

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. This instance is an unusual use of queue name resolution. It is the only situation in which name resolution takes place at a time when a queue is not being opened. Queue name resolution therefore occurs before interaction with the remote application that the message is being sent to.

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.

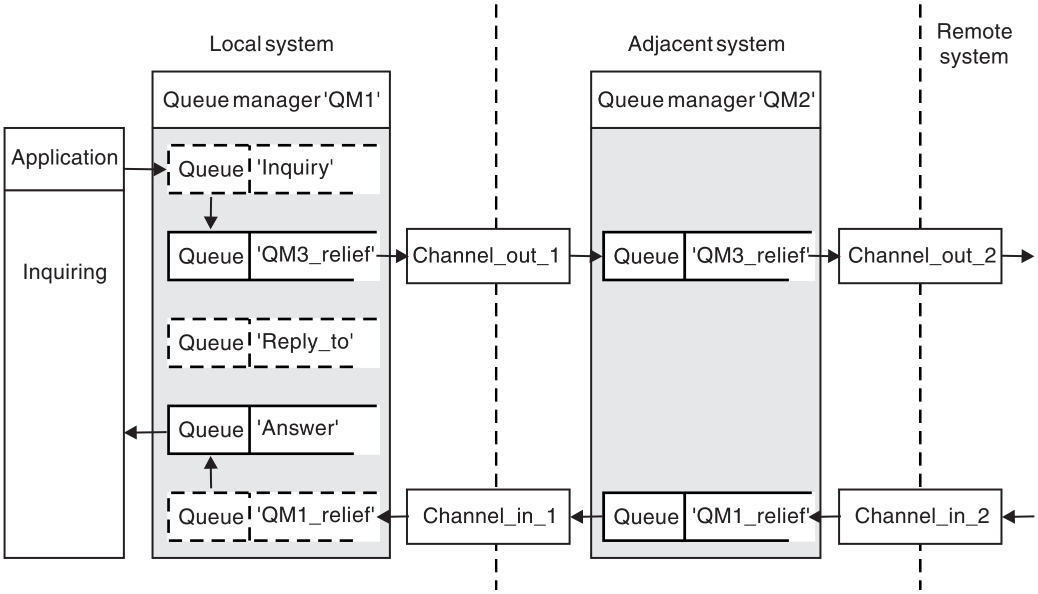


Figure 18. Reply-to queue alias used for changing reply location

In the example in Figure 18:

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.

How clusters work

Understand what clusters are and how they work.

A cluster is a network of queue managers that are logically associated in some way. The queue managers in a cluster might be physically remote. For example, they might represent the branches of an international chain store and be physically located in different countries. Each cluster within an enterprise must have a unique name.

Typically a cluster contains queue managers that are logically related in some way and need to share some data or applications. For example you might have one queue manager for each department in your company, managing data and applications specific to that department. You could group all these queue managers into a cluster so that they all feed into the payroll application. Or you might have one queue manager for each branch of your chain store, managing the stock levels and other information for that branch. If you group these queue managers into a cluster, they can all access the same set of sales and purchases applications. The sales and purchases application might be held centrally on the head-office queue manager.

Once you set up a cluster, the queue managers within it can communicate with each other, without you defining extra channel definitions or remote-queue definitions.

You can convert an existing network of queue managers into a cluster or you can establish a cluster as part of setting up a new network.

A IBM WebSphere MQ client can connect to a queue manager that is part of a cluster, just as it can connect to any other queue manager.

Clusters can also be used for workload management. For more information, see Using clusters for workload management (*WebSphere MQ V7.1 Installing Guide*).

How messages are routed in a cluster

If you are familiar with IBM WebSphere MQ and distributed queuing, 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 WebSphere 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.

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.

This information is stored in repositories. Most queue managers retain only the information that they need, that is, information about queues and queue managers with which they need to communicate. Each queue manager keeps the information in a partial repository. Some designated queue managers retain a full repository of all the information about all queue managers in the cluster.

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 might then be needed for communication between this queue manager and any other member of the cluster is created automatically

Queue managers on platforms that support clusters do not have to be part of a cluster. You can continue to use distributed queuing techniques as well as, or instead of, using clusters.

Example of a cluster

Figure 19 on page 98 shows the components of a cluster called `CLSTR1`.

- In this cluster, there are three queue managers, `QM1`, `QM2`, and `QM3`.
- `QM1` and `QM2` host repositories of information about all the queue managers and cluster-related objects in the cluster. They are referred to as *full repository queue managers*. The repositories are represented in the diagram by the shaded cylinders.
- `QM2` and `QM3` host some queues that are accessible to any other queue manager in the cluster. Queues that are accessible to any other queue manager in the cluster are called *cluster queues*. The cluster queues are represented in the diagram by the shaded queues. Cluster queues are accessible from anywhere in the cluster. IBM WebSphere MQ clustering code ensures that remote queue definitions for cluster queues are created on any queue manager that refers to them.

  As with distributed queuing, an application uses the MQPUT call to put a message on a cluster queue at any queue manager in the cluster. An application uses the MQGET call to retrieve messages from a cluster queue only on the queue manager where the queue resides.
- Each queue manager has a manually created definition for the receiving end of a channel called *`cluster-name.queue-manager`* on which it can receive messages. On the receiving queue manager, *`cluster-name.queue-manager`* is a cluster-receiver channel. A cluster-receiver channel is like a receiver channel used in distributed queuing; it receives messages for the queue manager. In addition, it also receives information about the cluster.
-

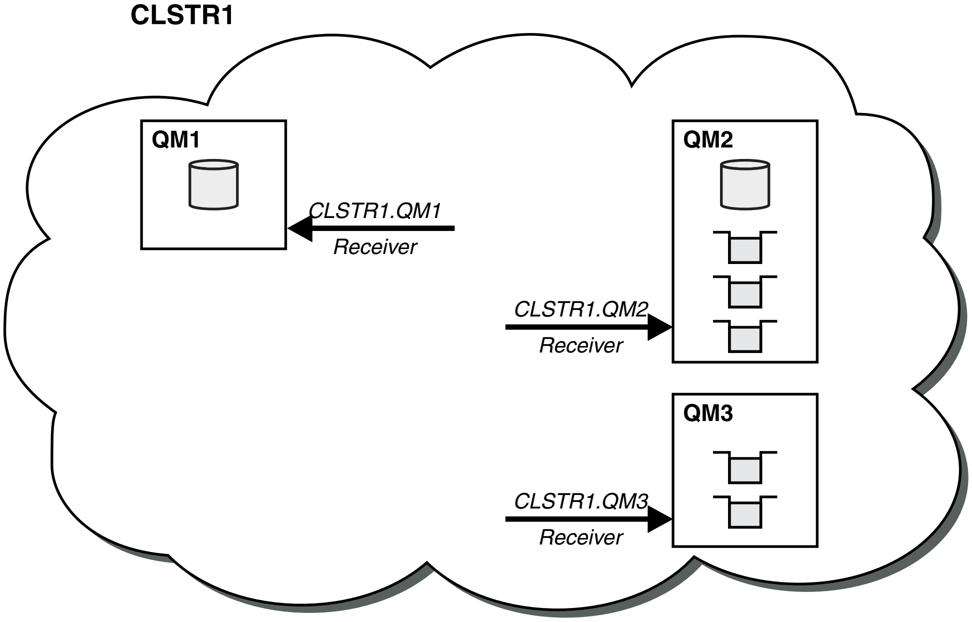


Figure 19. A cluster of queue managers

- In Figure 20 on page 99 each queue manager also has a definition for the sending end of a channel. It connects to the cluster-receiver channel of one of the full repository queue managers. On the sending queue manager, `*cluster-name*.*queue-manager*` is a cluster-sender channel. `QM1` and `QM3` have cluster-sender channels connecting to `CLSTR1.QM2`, see dotted line "2".

  `QM2` has a cluster-sender channel connecting to `CLSTR1.QM1`, see dotted line "3". A cluster-sender channel is like a sender-channel used in distributed queuing; it sends messages to the receiving queue manager. In addition, it also sends information about the cluster.

  Once both the cluster-receiver end and the cluster-sender end of a channel are defined, the channel starts automatically.

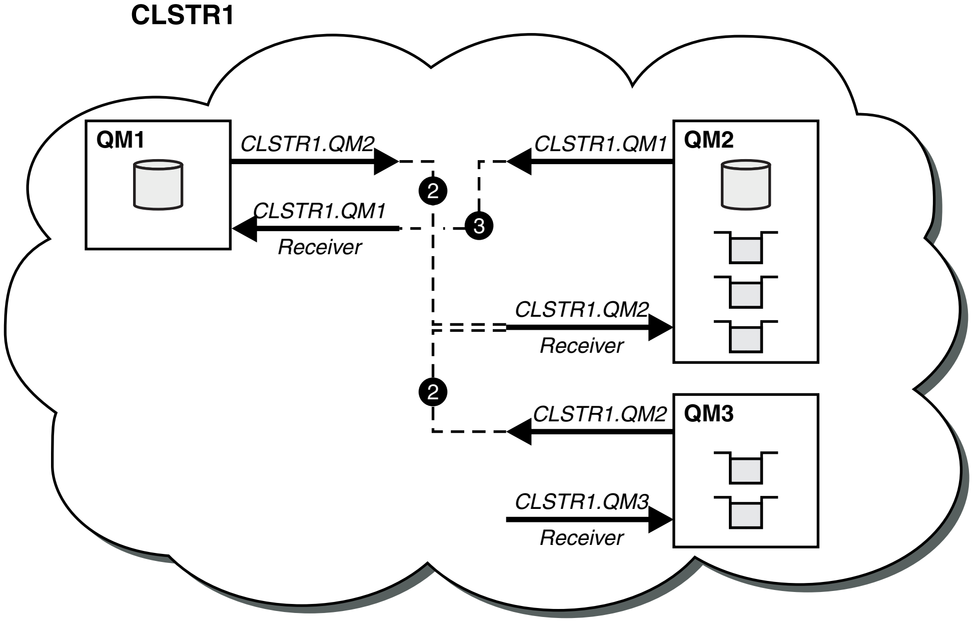


Figure 20. A cluster of queue managers with sender channels

What makes clustering work?

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

Look again at Figure 19 on page 98. Suppose that an application connected to queue manager QM3 wants to send some messages to the queues at QM2. The first time that QM3 must access those queues, it discovers them by consulting a full repository. The full repository in this case is QM2, which is accessed using the sender channel CLSTR1.QM2. With the information from the repository, it can automatically create remote definitions for those queues. If the queues are on QM1, this mechanism still works, because QM2 is a full repository. A full repository has a complete record of all the objects in the cluster. In this latter case, QM3 would also automatically create a cluster-sender channel corresponding to the cluster-receiver channel on QM1, allowing direct communication between the two.

Figure 21 on page 100 shows the same cluster, with the two cluster-sender channels that were created automatically. The cluster-sender channels are represented by the two dashed lines that join with the cluster-receiver channel CLSTR1.QM3. It also shows the cluster transmission queue, SYSTEM.CLUSTER.TRANSMIT.QUEUE, which QM1 uses to send its messages. All queue managers in the cluster have a cluster transmission queue, from which they can send messages to any other queue manager in the same cluster.

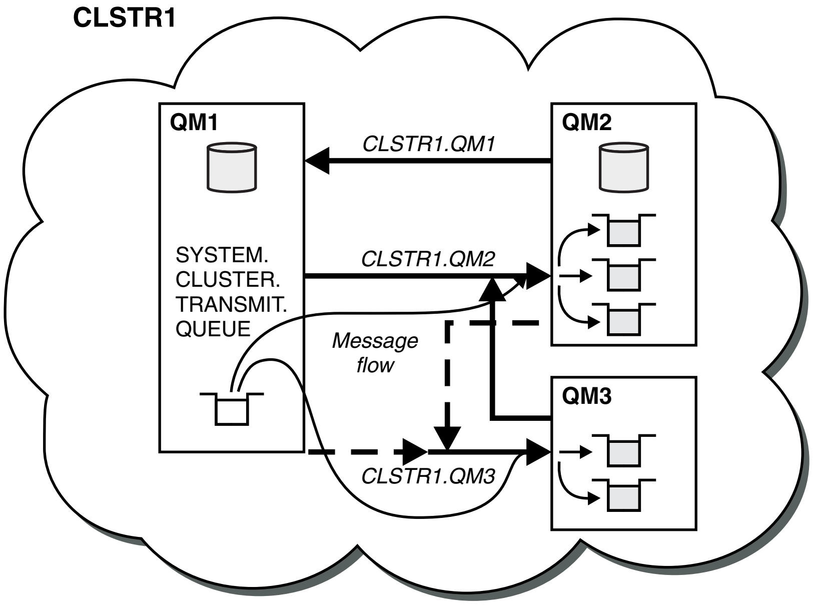


Figure 21. A cluster of queue managers, showing auto-defined channels

Note: Other diagrams show only the receiving ends of channels for which you make manual definitions. The sending ends are omitted because they are mostly defined automatically when needed. The auto-definition of most cluster-sender channels is crucial to the function and efficiency of clusters.

Related concepts:

Clusters

Comparison of clustering and distributed queuing (*WebSphere MQ V7.1 Installing Guide*)

Components of a cluster (*WebSphere MQ V7.1 Installing Guide*)

Related tasks:

Configuring a queue manager cluster (*WebSphere MQ V7.1 Installing Guide*)

Setting up a new cluster (*WebSphere MQ V7.1 Installing Guide*)

Managing IBM WebSphere MQ clusters (*WebSphere MQ V7.1 Installing Guide*)

IBM WebSphere MQ Telemetry

People, businesses, and governments increasingly want to use telemetry to interact more smartly with the environment we live and work in. Telemetry connects all kinds of devices to the internet and to the enterprise, and reduces the costs of building applications for smart devices.

MQTT support was previously available with either WebSphere Message Broker or WebSphere MQ Version 7.0.1, where WebSphere MQ Telemetry was a separate feature. Because WebSphere MQ Telemetry is a component of WebSphere MQ Version 7.1 and later, upgrading is essentially uninstalling WebSphere MQ Telemetry version 7.0.1 and installing WebSphere MQ Version 7.1. WebSphere 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 IBM WebSphere MQ Telemetry from Version 7.0.1 to Version 7.1 (*WebSphere MQ V7.1 Installing Guide*) or Migration of telemetry applications from using IBM

WebSphere Message Broker version 6 to use IBM WebSphere MQ Telemetry and IBM WebSphere Message Broker version 7.0 (*WebSphere MQ V7.1 Installing Guide*).

Included in IBM WebSphere MQ Telemetry are the following components:

Installer

IBM WebSphere MQ Telemetry is installed using either a GUI or command-line installer.

The Java components of the SDK are installed as WebSphere Eclipse Platform features. The C components of the SDK are supplied as compressed files.

Telemetry channels

Use telemetry channels to manage the connection of MQTT clients to IBM WebSphere MQ. Telemetry channels use new IBM WebSphere MQ objects, such as the `SYSTEM.MQTT.TRANSMIT.QUEUE`, to interact with IBM WebSphere MQ.

Telemetry service

MQTT clients use the `SYSTEM.MQXR.SERVICE` telemetry service to connect to telemetry channels.

IBM WebSphere MQ Explorer support for IBM WebSphere MQ Telemetry

IBM WebSphere MQ Telemetry can be administered using IBM WebSphere MQ Explorer.

Client Software Development Kit (SDK)

The client SDK has four parts:

1. MQTT v3 client libraries for Java SE and Java ME. Use the Java libraries to write Java clients for devices that support Java SE or Java ME.
2. MQTT v3 libraries for C. Use the C libraries to write C clients for a number of platforms.
3. IBM WebSphere MQ Telemetry daemon for devices, which is an advanced client written in C that runs on a number of platforms.
4. MQTT v3 protocol. The MQTT v3 protocol is published and licensed for reuse. Use the protocol, and reference MQTT client implementations, to write MQTT clients for different platforms and languages.

Documentation

IBM WebSphere MQ Telemetry documentation is included in the standard IBM WebSphere 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, red-ferries. 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 concepts and scenarios for monitoring and control” on page 103.

Related concepts:

Installing IBM WebSphere MQ Telemetry (*WebSphere MQ V7.1 Installing Guide*)

Administering IBM WebSphere MQ Telemetry (*WebSphere MQ V7.1 Administering Guide*)

Related tasks:

Migration of telemetry applications from using IBM WebSphere Message Broker version 6 to use IBM WebSphere MQ Telemetry and IBM WebSphere Message Broker version 7.0 (*WebSphere MQ V7.1 Installing Guide*)

Migrating IBM WebSphere MQ Telemetry from Version 7.0.1 to Version 7.1 (*WebSphere MQ V7.1 Installing Guide*)

Developing applications for IBM WebSphere MQ Telemetry (*WebSphere MQ V7.1 Programming Guide*)

Troubleshooting for IBM WebSphere MQ Telemetry (*WebSphere MQ V7.1 Administering Guide*)

Related reference:

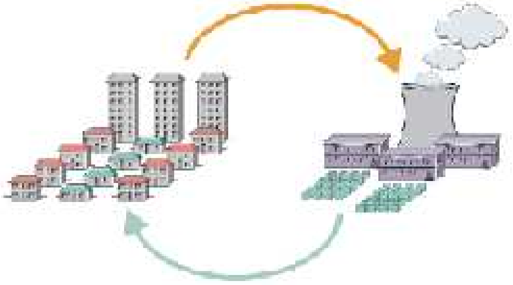
IBM WebSphere MQ Telemetry Reference (*WebSphere MQ V7.1 Reference*)

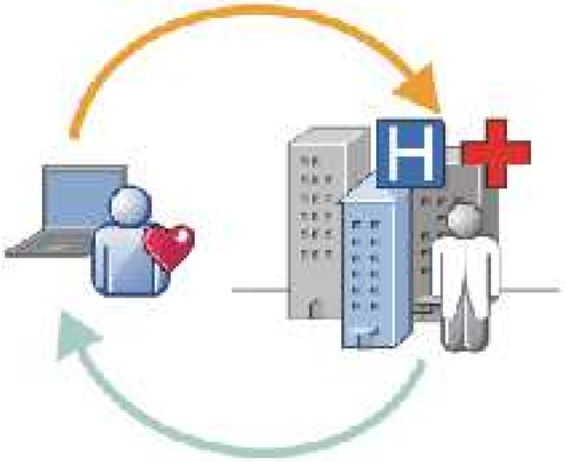
Telemetry concepts and scenarios for monitoring and control

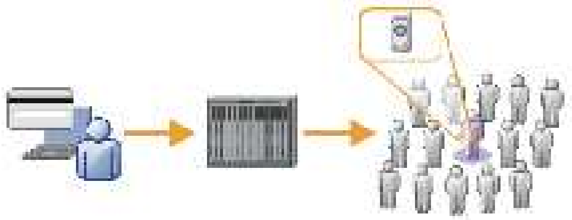
Telemetry is the automated sensing and measurement of data from a remote device. The emphasis is on the transmission of data from the device to a central control point. Telemetry also includes sending configuration and control information to the device.

IBM WebSphere MQ Telemetry connects small devices by using the MQTT protocol, and connects the devices to other applications by using IBM WebSphere MQ. IBM WebSphere 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 IBM WebSphere MQ Telemetry:

Telemetry: Smart Electricity	
	• MQTT message containing energy usage data sent to service provider. • IBM WebSphere MQ Telemetry sends CONTROL COMMANDS based on analysis of energy usage data. • For more information, see the following scenario: "Telemetry scenario: Home energy monitoring and control" on page 106

Telemetry: Smart Health Services	
• IBM WebSphere 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 scenario: "Telemetry scenario: Home patient monitoring" on page 104	

Telemetry: One in a Crowd	
	• A simple card transaction is sent to the bank’s server. • IBM WebSphere MQ Telemetry identifies the one person from the thousands, alerting the customer that their card has been used. • IBM WebSphere MQ Telemetry can use the simplest input of information, and locate that individual.

The subsequent scenarios, drawn from actual examples, illustrate some ways of using telemetry and some of the common problems that telemetry technology must resolve.

Related concepts:

“Telemetry scenario: Home patient monitoring”

“Telemetry scenario: Home energy monitoring and control” on page 106

“Telemetry scenarios: Radio Frequency Identification (RFID)” on page 107

“Telemetry scenarios: Environment sensing” on page 108

“Telemetry scenarios: Mobile applications” on page 109

Telemetry scenario: 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 scenarios:

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 Java SE andJava ME 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 patent 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 scenarios it is sufficient to encrypt the connection, using VPN or SSL. In other scenarios, 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 SSL, and the server authenticated to the device.

The telemetry channel between the device and the queue manager supports JAAS for user authentication and SSL for communication encryption, and device authentication. Access to a publication is controlled by the object authority manager in WebSphere 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 WebSphere 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 scenario 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, WebSphere MQ Telemetry has an initial target of 50,000 open connections per queue manager.

The connections are managed using WebSphere MQ Explorer. The 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 scenario: 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 deployed widely. The UK government, for instance, is in consultation about deployment of smart meters to every UK home by 2020.

Home metering scenarios 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 applicances, 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. WebSphere 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 SSL. To ensure authorization of control actions, the MQTT connection between the meter and the remote control point can be mutually authenticated using SSL.

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 scenarios 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 scenarios: Radio Frequency Identification (RFID):

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 scenarios vary enormously. RFID scenarios, and home patient monitoring and home energy monitoring and control scenarios, have some similarities and differences.

Invisibility

In many scenarios, 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 scenarios, the reader has a standard computing environment, such as 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 scenario, 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 loosing 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 scenarios: 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 SE or 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 scenario, 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 scenarios: Mobile applications:

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 scenarios, 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. SSL 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 scenarios.

Connecting telemetry devices to a queue manager

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 service.

As an alternative to connecting telemetry devices directly to the telemetry service, you can connect the devices to the WebSphere MQ Telemetry daemon for devices. The daemon is itself an MQTT v3 client. It pools the device connections, and makes a single connection to the telemetry service. You can connect daemons in a hierarchy, increasing the number of devices that can be indirectly connected to WebSphere MQ by many orders of magnitude.

The MQTT client initiates a TCP/IP connection using the MqttClient.connect method. Like WebSphere 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 WebSphere MQ Telemetry, called the telemetry service. Each queue manager runs a maximum of one telemetry service.

The telemetry 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 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, and create a first-failure data capture (FDC) file.

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 WebSphere MQ to connect MQTT clients using the wizards provided in the WebSphere MQ Telemetry supplement for WebSphere MQ Explorer. Alternatively, follow the instructions in Configuring a queue manager for telemetry on Linux and AIX (*WebSphere MQ V7.1 Administering Guide*) and Configuring a queue manager for telemetry on Windows (*WebSphere MQ V7.1 Administering Guide*) to configure telemetry manually.

Telemetry connection protocols

WebSphere MQ Telemetry supports TCP/IP IPv4 and IPv6, and SSL.

Telemetry service

The telemetry service is a TCP/IP listener, that is managed as a IBM WebSphere MQ service. Create the service using a IBM WebSphere MQ Explorer wizard, or with a **runmqsc** command.

The IBM WebSphere MQ Telemetry service is called SYSTEM.MQXR.SERVICE.

The **Telemetry sample configuration** wizard, provided in the IBM WebSphere MQ Telemetry supplement for IBM WebSphere MQ Explorer, creates the telemetry service and a sample telemetry channel; see Verifying the installation of IBM WebSphere MQ Telemetry by using IBM WebSphere MQ Explorer. Create the sample configuration from the command line; see Verifying the installation of IBM WebSphere MQ Telemetry using the command line (*WebSphere MQ V7.1 Installing Guide*).

The telemetry service starts and stops automatically with the queue manager. Control the service using the services folder in IBM WebSphere MQ Explorer. To see the service, you must click the icon to stop the 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 SSL authentication, or to manage groups of clients.

Create Telemetry channels using the **New Telemetry Channel** wizard, supplied in the WebSphere MQ Telemetry supplement for WebSphere MQ Explorer. Configure a channel, using the wizard, to accept connections from MQTT clients on a particular TCP/IP port.

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 SSL to encrypt messages and authenticate the telemetry channel and client; see SSL configuration of MQTT clients and telemetry channels (*WebSphere MQ V7.1 Administering Guide*). Specify the user ID to simplify authorizing access to WebSphere MQ objects. Specify a JAAS configuration to authenticate the MQTT user with JAAS; see MQTT client identification, authorization, and authentication (*WebSphere MQ V7.1 Administering Guide*).

MQTT protocol

The 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 MQ Telemetry Transport format and protocol (*WebSphere MQ V7.1 Reference*). 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 MQ Telemetry Transport client (*WebSphere MQ V7.1 Programming Guide*). "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.

The protocol is one of a family of MQTT protocols that are used in other products.

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 WebSphere MQ client applications, MQTT client apps are not IBM WebSphere MQ applications. They do not specify a queue manager to connect to. They are not limited to using specific IBM WebSphere 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 MQ Telemetry Transport format and protocol (*WebSphere MQ V7.1 Reference*).

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.

Two copies of the `com.ibm.micro.client.mqttv3.jar` JAR file are installed. One copy has a version number as part of the file name. For example: `com.ibm.micro.client.mqttv3_3.0.2.0-20100723.jar`. Use the versioned copy in OSGi applications. The content of the JAR files is the same.

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 a IBM WebSphere MQ application can start an exchange of messages.

MQTT client apps and IBM WebSphere MQ applications publish and subscribe to the same set of topics. A IBM WebSphere MQ application can also send a message directly to an MQTT client app without the client app first creating a subscription.

MQTT client apps are connected to IBM WebSphere MQ using a telemetry channel. The telemetry channel acts as a bridge between the different types of message used by MQTT and IBM WebSphere 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.

Send a message to an MQTT client

WebSphere 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 WebSphere MQ

Do the task, “Publishing a message to the MQTT client utility from WebSphere MQ Explorer” on page 115 to send a publication from WebSphere 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 22 on page 113, the MQTT client subscribes using the topic string `"MQTT Examples"`. A WebSphere MQ C application, Figure 23 on page 114, publishes to the topic using the topic string `"MQTT Examples"`. In the code snippet Figure 24 on page 114, the MQTT client receives the publication in the callback method, messageArrived.

For further information about how to configure WebSphere MQ to send publications in response to subscriptions from MQTT clients, see Publishing a message in response to an MQTT client subscription.

A WebSphere MQ application sends a message directly to an MQTT client

Do the task, “Sending a message to an MQTT client using WebSphere MQ Explorer” on page 119 to send a message directly from WebSphere 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 service sets the topic name to the remote queue name.

You cannot send unsolicited messages to the WebSphere MQ daemon for devices: the daemon might shut down if it receives an unsolicited message. An MQTT v3 client cannot send an unsolicited message to another MQTT v3 client, nor to a WebSphere MQ queue.

For further information about how to configure WebSphere 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 25 on page 115 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, "Publish a message to a specific MQTT v3 client" on page 120, 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 22 shows how an MQTT client written in Java creates a subscription. It also needs a callback method, messageArrived to receive publications for the subscription. The code snippet is extracted from the task, Creating a subscriber for MQ Telemetry Transport using Java (*WebSphere MQ V7.1 Programming Guide*).

```
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 22. MQTT v3 client subscriber

The code snippet in Figure 23 on page 114 shows how an WebSphere MQ application written in C sends a publication. The code snippet is extracted from the task, Create a publisher to a variable topic (*WebSphere MQ V7.1 Programming Guide*)

```
/* 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 23. WebSphere MQ publisher

When the publication arrives, the MQTT client calls the messageArrived method of the MQTT application client MqttCallback class. The code snippet is extracted from the task, Creating a subscriber for MQ Telemetry Transport using Java (*WebSphere MQ V7.1 Programming Guide*).

```
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 24. messageArrived method

Figure 25 on page 115 shows an MQTT v3 publishing a message to the subscription created in Figure 22 on page 113. The code snippet is extracted from the task, Creating your first MQ Telemetry Transport publisher application using Java (*WebSphere MQ V7.1 Programming Guide*).

```
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(Example.topicString);
String     publication  = "Hello world";
MqttMessage     message = new MqttMessage(publication.getBytes());
MqttDeliveryToken token = topic.publish(message);
```

Figure 25. MQTT v3 client publisher

Publishing a message to the MQTT client utility from WebSphere MQ Explorer:

Follow the steps in this task to publish a message using WebSphere 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 WebSphere MQ and the WebSphere MQ Explorer, and that WebSphere MQ and WebSphere 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 WebSphere MQ Explorer user ID is assumed to be member of the `mqm` group.

About this task

In the task, you create a topic in WebSphere MQ and subscribe to the topic using the MQTT client utility. When you publish to the topic using WebSphere MQ Explorer, the MQTT client receives the publication.

Procedure

Do one of the following tasks:

- You have installed WebSphere MQ Telemetry, but you have not started it yet. Do the task: “Start task with no telemetry service yet defined” on page 116.
- You have run WebSphere MQ telemetry before, but want to use a new queue manager to do the demonstration. Do the task: “Start task with no telemetry service yet defined” on page 116.
- 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 (*WebSphere MQ V7.1 Administering Guide*)
     - Configuring a queue manager for telemetry on Windows (*WebSphere MQ V7.1 Administering Guide*)
  2. Do the task: “Start task with a running telemetry service” on page 117
- 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 service” on page 117.

What to do next

Do "Sending a message to an MQTT client using WebSphere MQ Explorer" on page 119 to send a message directly to the client utility.

Start task with no telemetry 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 WebSphere 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 118. In this task, you create a queue manager to avoid the possibility of interfering with any existing default transmission queue.

Procedure

1. Using WebSphere 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 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 WebSphere MQ.
   a. Right-click the `Queue Managers\`*`QmgrName`*`\Topics` folder in the WebSphere 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 WebSphere MQ Explorer.
   a. Click the `Queue Managers\`*`QmgrName`*`\Topics` folder in the WebSphere 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 service:

Create a telemetry channel and a topic. Authorize the user to use the topic and the telemetry transmit queue. Publish a message using WebSphere 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 service is defined and running. The telemetry 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 118.

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 WebSphere MQ, and start the MQTT client utility.
   a. Right-click the Queue Managers\QmgrName\Telemetry\Channels folder in the WebSphere 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 WebSphere MQ.
   a. Right-click the Queue Managers\QmgrName\Topics folder in the WebSphere 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 WebSphere MQ Explorer.
   a. Click the Queue Managers*QmgrName*\Topics folder in the WebSphere 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 WebSphere 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 service yet defined” on page 116 or the task, “Start task with a running telemetry service” on page 117.

About this task

When an MQTT client creates a subscription, WebSphere 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 WebSphere MQ Explorer.

   a. Click the `Queue Managers\`*QmgrName*`\Topics` folder in the WebSphere 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 WebSphere MQ Explorer:

Send a message to the MQTT client utility by putting a message onto a WebSphere MQ queue using WebSphere 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 WebSphere MQ Explorer" on page 115. 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 MQTT clean sessions (*WebSphere MQ V7.1 Programming Guide*).

   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 Identifer** 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 WebSphere 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.

Publish 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 WebSphere MQ as the publish/subscribe broker. Repeat the task using WebSphere MQ Telemetry daemon for devices as the publish/subscribe broker.

Before you begin

Do the task, "Publishing a message to the MQTT client utility from WebSphere MQ Explorer" on page 115. 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.

An additional task, "Using the WebSphere MQ Telemetry daemon for devices as the publish/subscribe broker" on page 122, uses the WebSphere MQ Telemetry daemon for devices as the publish/subscribe broker, rather than WebSphere MQ.

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 MQTT clean sessions (*WebSphere MQ V7.1 Programming Guide*).

   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 Identifer** 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 only 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 service.

What to do next

Do the next task, "Using the WebSphere MQ Telemetry daemon for devices as the publish/subscribe broker" on page 122. The MQTT client utility connects to the WebSphere MQ Telemetry daemon for devices rather than to a telemetry channel.

Using the WebSphere MQ Telemetry daemon for devices as the publish/subscribe broker:

Use the WebSphere MQ Telemetry daemon for devices as the publish/subscribe broker instead of WebSphere MQ. Publish a message with one instance of the MQTT client utility to send to another instance, by subscribing using its `ClientIdentifier` as a topic string.

Before you begin

Install the daemon, if you have not done so already..

Do not run the verification; it uses port 1883, which is already in use by the `PlainText` telemetry channel,

About this task

In the task, you connect the MQTT client utility to the WebSphere MQ Telemetry daemon for devices using a non-default TCP/IP port. One client subscribes using its `ClientIdentifier` as a topic string, and the other client publishes to `ClientIdentifier`, exactly in the same way as in the previous task, see Procedure.

Note: The task is documented for running the daemon on Windows. To run the daemon on Linux, modify the path and the permissions for `amqtdd`.

Procedure

1. Open a command window in the directory containing the WebSphere MQ Telemetry daemon for devices.

   The directory path for Windows is, *`WebSphere MQ installation directory`*`\mqxr\SDK\advanced\DeviceDaemon\windows_ia32`
2. Run the daemon on a different TCP/IP port.
   1. Create a file called `amqtdd.cfg` in the same directory as the daemon.
   2. Add a line to the file to configure a different default port for the daemon.

      ```
      port 1884
      ```
   3. Save the file.
3. Start the daemon.

   ```
   amqtdd
   ```

   The daemon writes its console log to the command window:

   ```
   20100712 123133.857 CWNAN9999I IBM WebSphere MQ Telemetry daemon for devices
   20100712 123133.857 CWNAN9997I Licensed Materials - Property of IBM
   20100712 123133.857 CWNAN9996I Copyright IBM Corp. 2007, 2019 All Rights Reserved
   20100712 123133.857 CWNAN9995I US Government Users Restricted Rights ...
   20100712 123133.857 CWNAN0049I Configuration file name is .\amqtdd.cfg
   20100712 123133.873 CWNAN0054I Features included: bridge
   20100712 123134.060 CWNAN0014I MQTT protocol starting, listening on port 1884
   ```
4. Start an instance of the MQTT client utility.

   Start the MQTT client utility only from a telemetry channel, and then you can connect to the daemon. Alternatively you can install the WebSphere MQ SupportPac, IA92. The SupportPac is available from IA92: WBI Brokers - Java implementation of WebSphere MQ Telemetry transport.
   1. Open the `Queue Managers\`*`QmgrName`*`\Telemetry\channels` folder.
   2. Right-click the **PlainText** channel > **Run MQTT Client Utility...**
   3. Change the **Port** to `1884`.
   4. Change the **Client Identifer** to `MyClient`.

   e. Click **Connect**.

   The **Client History** records a `Connected` event

5. 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.

6. 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. Change the **Port** to `1884`.

   d. Click **Connect**.

   The **Client History** records a `Connected` event

7. 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.

You can also monitor the connection and disconnection events to the WebSphere MQ Telemetry daemon for devices in the command window.

Send a message to a WebSphere MQ application from an MQTT client

A WebSphere MQ application can received a message from an MQTT v3 client by subscribing to a topic. The MQTT client connects to WebSphere MQ using telemetry channel, and sends a message to the WebSphere MQ application by publishing to the same topic.

Do the task, “Publishing a message to WebSphere MQ from an MQTT client,” to learn how to send a publication from an MQTT client to a subscription defined in WebSphere 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 WebSphere MQ from an MQTT client:

Create a subscription to a topic using WebSphere MQ Explorer and publish to the topic using WebSphere MQTT client utility.

Before you begin

Do the task, “Publishing a message to the MQTT client utility from WebSphere MQ Explorer” on page 115. 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 WebSphere MQ Explorer.

Procedure

1. Create a durable subscription to the topic string `MQTT Example`. Do either of the following procedures:
   - Run the command script described in Results
   - Do the following steps to create the queue, and subscription using WebSphere MQ Explorer.

   a. Right-click the `Queue Managers\`*QmgrName*`\Queues` folder in the WebSphere MQ Explorer > **New** > **Local queue...**.

   b. Type `MQTTExampleQueue` as the queue name > **Finish**.

   c. Right-click the `Queue Managers\`*QmgrName*`\Subscriptions` folder in the WebSphere 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 116 of "Publishing a message to the MQTT client utility from WebSphere MQ Explorer" on page 115.

   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 WebSphere MQ Explorer" on page 115.

   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 WebSphere 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 WebSphere 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.

Transfer messages between the WebSphere MQ Telemetry daemon for devices and WebSphere MQ

Do this task to learn how to send commands to the WebSphere MQ Telemetry daemon for devices. The commands you write create a bridge that transfers messages from WebSphere MQ to the daemon, and messages from the daemon to WebSphere MQ.

Before you begin

Do the tasks "Publish a message to a specific MQTT v3 client" on page 120 and "Using the WebSphere MQ Telemetry daemon for devices as the publish/subscribe broker" on page 122 to become familiar with using the MQTT client utility. When you have finished the tasks, leave one instance of the MQTT client utility connected to the telemetry daemon for devices. Leave another instance connected to the telemetry channel.

The task presumes you have defined a channel to the telemetry service listening to port 1883 on address `127.0.0.1`. Likewise, the default daemon listener is configured to listen to port 1884 on address `127.0.0.1`. A single line in the file `amqtdd.cfg`, which is stored in the same directory as the daemon, `amqtdd`, configures the default daemon listener port.

```
port 1884
```

About this task

In this task, you update a running daemon to create a connection bridge to the WebSphere MQ telemetry service, and then exchange messages with the daemon.

Tip: The update file, amqtdd.upd, is deleted by the daemon after it is used. To keep the commands you create for later use, you might want to create the commands in a different file and then transfer them to amqtdd.upd.

Procedure

1. Make sure that you have two instances of the MQTT client utility running. One is connected to the daemon on port 1884, and one is connected to the telemetry channel running on port 1883.
2. Create the file, amqtdd.upd, in the same directory as the daemon, amqtdd, with the following commands in the file.

   ```
   connection daemon1
   address 127.0.0.1:1883
   topic # in  import/ export/
   topic # out export/ import/
   try_private false
   ```

   - The bridge is called daemon1, and it connects to the channel configured for the telemetry service running at the socket address, 127.0.0.1:1883. The try_private command is optional; true is the default. Without this line, the bridge first tries to connect using a private protocol that is understood by WebSphere MQ Telemetry daemon for devices. Including try_private false in the commands avoids this step, and speeds up the time to finish a successful connection.
   - The line, topic # in import/ export/, instructs daemon1 to subscribe to all topics matching the topic string export/# created in the queue manager. It transfers the matching publications from the queue manager to the daemon, changing the start of the topic string from export/ to import/. The line, topic # out export/ import/, creates a subscription at the local daemon. The bridge subscribes to all topics matching the topic string export/# created in the daemon. It transfers publications from the daemon to the queue manager, changing the start of the topic string from export/ to import/.

   Figure 26 shows the resulting console log.

```
CWNAN0124I Starting bridge connection daemon1
CWNAN0133I Bridge connection daemon1 to 127.0.0.1:1883 now established
```

Figure 26. Console log from starting connection bridge

3. In each instance of the MQTT client utility, type import/# in the **Subscription/Topic:** input field > **Subscribe**.
4. In each instance of the MQTT client utility, type export/# in the **Publication/Topic:** input field.
   a. In the MQTT client utility connected to port 1883, the telemetry channel, type From the queue manager in the **Publication/Message:** input field > **Publish**.
   b. In the MQTT client utility connected to port 1884, the telemetry daemon, type From the daemon in the **Publication/Message:** input field > **Publish**.

   The client history in each MQTT client utility shows the publication that has been transferred from one broker to the other.

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 WebSphere 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 WebSphere 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 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 (*WebSphere MQ V7.1 Administering Guide*).

An MQTT client can create a persistent session; see “MQTT stateless and stateful sessions” on page 130. 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 (*WebSphere MQ V7.1 Programming Guide*). 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 WebSphere MQ or WebSphere Message Broker message flows.

Use JMS, MQI, or other WebSphere MQ programming interfaces to program telemetry applications in WebSphere MQ.

The telemetry service converts between MQTT v3 messages and WebSphere 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 a WebSphere MQ message; see “Integration of WebSphere MQ Telemetry with queue managers” on page 127.

Use the Publication, MQInput, and JMSInput nodes to send and receive publications between WebSphere Message Broker and MQTT clients.

Using message flows you can integrate telemetry with Web sites using HTTP, and with other applications using WebSphere MQ and WebSphere Adapters.

WebSphere MQ Telemetry replaces the SCADA nodes in WebSphere Message Broker version 7. See Migration of telemetry applications from using IBM WebSphere Message Broker version 6 to use IBM WebSphere MQ Telemetry and IBM WebSphere Message Broker version 7.0 (*WebSphere MQ V7.1 Installing Guide*) for information about how to migrate version 6 WebSphere Message Broker message flows using the SCADAInput and SCADAOutput nodes to version 7.

Integration of WebSphere MQ Telemetry with queue managers

The MQTT client is integrated with WebSphere MQ as a publish/subscribe application. It can either publish or subscribe to topics in WebSphere MQ, creating new topics, or using existing topics. It receives publications from WebSphere MQ as a result of MQTT clients, including itself, or other WebSphere 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 WebSphere MQ, are not supported. "MQTT client to WebSphere MQ publish/subscribe broker" and "WebSphere MQ to an MQTT client" on page 128 describe how attributes of publications are set. The settings depend on whether the publication is going to or from the WebSphere MQ publish/subscribe broker.

In WebSphere 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 WebSphere 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 WebSphere 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 a WebSphere MQ application, or an administrator creates a subscription, the subscription is named. List subscriptions using WebSphere 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 WebSphere MQ publish/subscribe broker

An MQTT client has sent a publication to WebSphere MQ. The telemetry service converts the publication to a WebSphere MQ message. The WebSphere MQ message contains three parts:

1. MQMD
2. RFH2
3. Message

MQMD properties are set to their default values, except where noted in Table 9.

Table 9. 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 WebSphere MQ administrator for the telemetry channel.
Priority	MQLONG	MQPRI_PRIORITY_AS_Q_DEF (Default for WebSphere MQ, which is different to JMS that has a default of 4.)

Persistence	MQLONG	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 service creates the WebSphere 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 (*WebSphere MQ V7.1 Programming Guide*).

RFH2 values are set as shown in Table 10. The Format property is set in the RFH2 fixed header and the other values are set in RFH2 folders.

Table 10. 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	mqps/Ret	Set if the original MQTT publication was sent with the RETAIN property set and the message is received as a retained publication.
MQTopicString	mqps/Top	The topic to which the MQTT message was published.

The payload in an MQTT publication is mapped to the contents of a WebSphere MQ message:

Table 11. Message contents

Message contents	**Type**	**Contents**
Buffer	MQBYTE*n*	Copy of bytes from incoming MQTT message. The length can be zero.

WebSphere MQ to an MQTT client

A client has subscribed to a publication topic. A WebSphere MQ application has published to the topic, resulting in a publication being sent to the MQTT subscriber by the WebSphere MQ publish/subscribe broker. Alternatively, a WebSphere MQ application has sent an unsolicited message directly to an MQTT client. Table 12 describes how the fixed message headers are set in the message that is sent to the MQTT client. Any other data in the WebSphere MQ message header, or any other headers, are discarded. The message data in the WebSphere 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 service.

Table 12. 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	The way the value of QoS in an outgoing publication from the publish/subscribe broker in WebSphere MQ is set depends on the incoming publication. It depends on whether the incoming publication was sent from an MQTT client, or from a WebSphere MQ application. **MQTT** Lower value of the QoS in the incoming publication, and in the QoS requested by the subscriber. **WebSphere MQ** Lower value of the QoS derived from the incoming publication: MQPER_NOT_PERSISTENT→QoS=0 MQPER_PERSISTENT→QoS=2 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.

Table 12. MQTT fixed header properties (continued)

MQTT field	Type	Value
RETAIN	boolean	Set if the incoming publication has the retained property set.

Table 13 describes how the variable message headers are set in the MQTT message that is sent to the MQTT client.

Table 13. 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.

Telemetry daemon for devices

The WebSphere MQ Telemetry daemon for devices is an advanced MQTT V3 client application. Use it to store and forward messages from other MQTT clients. It connects to WebSphere MQ like an MQTT client, but you can also connect other MQTT clients to it. You can connect it to other telemetry daemons too.

It serves four basic purposes:

1. **Connect local MQTT clients together in a publish/subscribe network.**

   You might connect the sensor and an actuator of a device as separate MQTT clients to the daemon. The sensor publishes its gauge readings, and the actuator subscribes to the readings, modifying its behavior based on their values. The readings are acted on locally.

2. **Filter which subscriptions, and which messages are published to the queue manager, and to the device.**

   In the previous example, a WebSphere Message Broker message flow might subscribe to the topic that the daemon publishes readings to. The flow updates a Web page and shows the state of the device.

   The daemon might also forward the subscription that the actuator created to the queue manager. A WebSphere Message Broker flow publishes a message to the topic the MQTT client servicing the actuator subscribed to. The MQTT client modifies the device settings.

   The message flow might start from a Web page using a WebSphere Message Broker HTTPInput node.

3. **Multiplex multiple MQTT clients onto a single TCP/IP connection to a telemetry channel.**

   Rather than each device connecting separately to the telemetry server, the daemon forwards publications and subscriptions on a single TCP/IP connection. The daemon reduces the number of TCP/IP connections managed by the telemetry service.

   Individual MQTT clients connect to the daemon. The individual clients are invisible to the queue manager. The daemon makes one connection to the queue manager on behalf of all the clients that connect to it.

4. **Store and forward messages between devices and the queue manager**

The daemon takes the responsibility for protecting telemetry devices from short-lived connection failures of the connection to the queue manager.

A device might only support "fire and forget" messaging. If the connection to the queue manager is only available intermittently, or is unreliable, the device has no way to transfer information predictably or reliably.

A solution is to attach the device to the daemon using a local connection that is always available. The daemon can buffer the messages that flow to and from the queue manager in its memory. It can use a reliable quality of service to send the messages to and from the queue manager on an unreliable connection.

Note: The daemon does not have persistent storage for “inflight” messages. Messages are buffered in memory.

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 MQTT clean sessions (*WebSphere MQ V7.1 Programming Guide*).

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 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 service.

Session state information is saved by the queue manager in the `SYSTEM.MQTT.PERSISTENT.STATE` queue.

The WebSphere 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 The MQTT client last will and testament publication (*WebSphere MQ V7.1 Programming Guide*). It is sent by the telemetry 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 (*WebSphere MQ V7.1 Programming Guide*). 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 130. Its subscriptions are active until they are deleted, or until the client creates a clean session.

Loose coupling between MQTT clients and WebSphere MQ applications

The flow of publications between MQTT clients and WebSphere MQ applications is loosely coupled. Publications might originate from either an MQTT client or a WebSphere 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 a WebSphere MQ application.

MQTT clients and WebSphere 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 MQTT delivery tokens (*WebSphere MQ V7.1 Programming Guide*).

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 WebSphere 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. A WebSphere 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.

WebSphere 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 service. WebSphere MQ Telemetry provides two other security mechanisms, SSL and JAAS.

SSL 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 SSL (*WebSphere MQ V7.1 Administering Guide*). You can also use SSL to check that the client device is permitted to connect to the server; see MQTT client authentication using SSL (*WebSphere MQ V7.1 Administering Guide*).

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 (*WebSphere MQ V7.1 Administering Guide*). JAAS can be used with LDAP to check a password using a single sign-on directory.

SSL and JAAS can be used in conjunction to provide two factor authentication. You can restrict the ciphers used by SSL 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 WebSphere 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 (*WebSphere MQ V7.1 Administering Guide*).

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.

WebSphere 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 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 WebSphere MQ Telemetry with queue managers" on page 127. 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 WebSphere MQ Telemetry

Consider the following factors when managing large numbers of clients and improving scalability of WebSphere MQ Telemetry.

Capacity Planning

For information about performance reports for WebSphere MQ Telemetry, select the WebSphere MQ Telemetry Performance Evaluations report from the website WebSphere 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 an SSL 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 good.

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 (*WebSphere MQ V7.1 Programming Guide*) (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 WebSphere MQ Explorer, and helps with problem determination; see The MQTT client identifier (*WebSphere MQ V7.1 Programming Guide*).

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 WebSphere MQ gives it a name of the form:

```
 ClientIdentifier:TopicName
```

To view all of the subscriptions for an MQTT client, you can use the WebSphere MQ MQSC **DISPLAY** command:

```
DISPLAY SUB('ClientID1:*')
```

Defining resources in WebSphere MQ for use by MQTT clients

An MQTT client connects to WebSphere MQ a remote queue manager. There are two basic methods for a WebSphere 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 (*WebSphere MQ V7.1 Administering Guide*).

Improving scalability by avoiding subscriptions.

When an MQTT V3 client subscribes to a topic, a subscription is created by the telemetry service in WebSphere 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 WebSphere 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 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 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 service in the java.properties file; see Passing JVM parameters to the telemetry 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 27. Alter maximum number of handles on Windows

```
echo "ALTER QMGR MAXHANDS(999999999)" | runmqsc qMgrName
```

Figure 28. 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 MQTT clean sessions (*WebSphere MQ V7.1 Programming Guide*). Alternatively, publish and subscribe using the best effort Quality of Service, `0`, in an MQTT client; see Quality of service (*WebSphere MQ V7.1 Programming Guide*). Use `non-persistent` messages when sending messages from WebSphere MQ. Messages with these qualities of service are not recovered when the system or connection restarts.

Devices supported by WebSphere MQ Telemetry

MQTT clients can connect to 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.

WebSphere MQ Telemetry provides two clients, which both implement the MQTT v3 protocol:

- A Java client that 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.
- 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 XP, 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 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 reference implementation support pack can be ported to devices that might not be on one of these platforms. Source code is provided for reference implementation (plus some binary files).

Objects

Many of the administration tasks involve manipulating IBM WebSphere MQ **objects**. The object types are queue managers, queues, process definitions, namelists, channels, client connection channels, listeners, services, and authentication information objects.

Queue managers define the properties (known as attributes) of these objects. The values of these attributes affect the way in which WebSphere MQ processes these objects. From your applications, you use the Message Queue Interface (MQI) to control these objects. Objects are identified by an *object descriptor* (MQOD) when addressed from a program.

When you use WebSphere MQ commands to define, alter, or delete objects, for example, the queue manager checks that you have the required level of authority to perform these operations. 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.

The manipulation or *administration* of objects includes:

- Starting and stopping queue managers.
- Creating objects, particularly queues, for applications.
- 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.

For information about naming IBM WebSphere MQ objects, see “Naming IBM WebSphere MQ objects” on page 137.

For information about the default objects created on a queue manager, see “System default objects” on page 163.

For an overview of methods about how to create and manage IBM WebSphere MQ objects, see “Managing objects” on page 143.

For information about the different types of IBM WebSphere MQ objects, see the following topics:

- “Queues” on page 144
- “WebSphere MQ queue managers” on page 154
- “Process definitions” on page 156
- “Namelists” on page 156
- “Authentication information objects” on page 157
- “Communication information objects” on page 157
- “Channels” on page 158
- “Client connection channels” on page 161
- “Listeners” on page 162
- “Services” on page 162
- “Topics” on page 162

For information about other administrative concepts, see the following topics:

- “Clusters” on page 156
- “Queue-sharing groups” on page 154

Related concepts:

"Introduction to message queuing" on page 73

"Object attributes" on page 144

Related reference:

The MQSC commands (*WebSphere MQ V7.1 Reference*)

Naming IBM WebSphere MQ objects

The naming convention adopted for WebSphere MQ objects depends on the object. The name of the machines and the user IDs that you use with IBM WebSphere 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 WebSphere 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 WebSphere MQ objects, see "Rules for naming IBM WebSphere MQ objects."

The name of the machines and the user IDs that you use with IBM WebSphere MQ are also subject to some naming restrictions:

- Ensure that the machine name does not contain any spaces. IBM WebSphere MQ does not support machine names that include spaces. If you install IBM WebSphere MQ on such a machine, you cannot create any queue managers.
- For IBM WebSphere MQ authorizations, names of user IDs and groups must be no longer than 20 characters (spaces are not allowed).
- A WebSphere 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:

"Understanding WebSphere MQ file names" on page 140

Related reference:

"Rules for naming IBM WebSphere MQ objects"

Rules for naming IBM WebSphere MQ objects:

IBM WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 (*WebSphere MQ V7.1 Installing Guide*) for more information about channels.
3. Topic strings can be a maximum of 10240 bytes. All IBM WebSphere MQ object names are case-sensitive.
4. The maximum length of storage class names is 8 characters.
5. The maximum length of CF structure names is 12 characters.

Characters in object names

The valid characters for IBM WebSphere MQ object names are:

Characters	Restrictions
Uppercase A - Z	None

Characters	Restrictions
Lowercase a - z	• In MQSC scripts, names with lowercase characters must be enclosed in single quotes. 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 quotes. This prevents the lowercase characters being folded into uppercase.
Numerics 0 - 9	None
Period (.)	None.
Underscore (_)	• Avoid using names with leading or trailing underscores because they cannot be handled by the IBM WebSphere MQ for z/OS operations and control panels.
Forward slash (/)	• 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 quotes.
Percent sign (%)	• None • If you are using RACF as the external security manager for IBM WebSphere 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 quotes.

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 WebSphere MQ contain the name of the queue manager.

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 WebSphere 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

On WebSphere MQ for z/OS, the coupling facility application structure name CSQSYSAPPL is reserved.

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 WebSphere MQ are listed in full in Queue names (*WebSphere MQ V7.1 Reference*).

On IBM WebSphere MQ for z/OS, the coupling facility application structure name CSQSYSAPPL is reserved.

Understanding WebSphere MQ file names:

Each WebSphere 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 WebSphere MQ configuration information:
  - On Windows 32-bit systems the default prefix is `C:\Program Files\IBM\WebSphere MQ`. On Windows 64-bit systems the default prefix is, `C:\Program Files\IBM\WebSphere MQ (x86)\`. This is configured in the `DefaultPrefix` stanza of the mqs.ini configuration file.
  - On UNIX and Linux systems 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 WebSphere MQ properties page in the WebSphere 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 WebSphere 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`:

- In WebSphere MQ for Windows with NTFS or FAT32, the queue manager name becomes:
  ```
  c:\Program Files\IBM\WebSphere MQ\qmgrs\queue!manager
  ```
- In WebSphere MQ for Windows with FAT, the queue manager name becomes:
  ```
  c:\Program Files\IBM\WebSphere MQ\qmgrs\queue!ma
  ```
- In WebSphere MQ for UNIX and Linux systems, the queue manager name becomes:
  ```
  /var/mqm/qmgrs/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 **dspmqfls** command to convert between real and transformed object names.

Understanding WebSphere MQ for IBM i queue manager library names:

Use this information to understand library names, IFS directories, and name transformation.

When a queue manager is created, WebSphere 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.

Understanding WebSphere MQ IFS directories and files

The IBM i Integrated File System (IFS) is used extensively by WebSphere MQ to store data. For more information about the IFS see the *Integrated File System Introduction*.

Each WebSphere MQ object (for example: channels or queue managers) 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 WebSphere 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 WebSphere 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 WebSphere MQ objects.

Managing objects

An overview of how to create, alter, display, and delete objects.

For further information about these objects, and queue-sharing groups (which are only supported on WebSphere MQ for z/OS and which are not strictly objects), see "Objects" on page 136.

With the exception of dynamic queues, these objects must be defined to the queue manager before you can work with them.

You define and manage objects using:

- The PCF commands described in Programmable command formats reference (*WebSphere MQ V7.1 Reference*)and Automating administration tasks (*WebSphere MQ V7.1 Administering Guide*)
- The MQSC commands described in The MQSC commands (*WebSphere MQ V7.1 Reference*)
- The WebSphere MQ for z/OS operations and control panels, described in Operating WebSphere MQ for z/OS (*WebSphere MQ V7.1 Administering Guide*)
- The WebSphere MQ Explorer (Windows, UNIX, and Linux for Intel systems only)

You can manage objects also by using the following methods:

- Control commands, which are typed in from a keyboard. See The control commands (*WebSphere MQ V7.1 Reference*).
- IBM WebSphere MQ Administration Interface (MQAI) calls in a program. See WebSphere MQ Administration Interface (MQAI) (*WebSphere MQ V7.1 Administering Guide*).
- IBM WebSphere MQ for Windows only:
  - MQAI Component Object Model (COM) calls in a program
  - The Windows Default Configuration Application

You can also display or alter the attributes of objects, or delete the objects.

Alternatively, for sequences of WebSphere 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 WebSphere MQ (*WebSphere MQ V7.1 Administering Guide*), and demonstrated in the Mail Manager sample application delivered with WebSphere MQ for z/OS. For a description of this sample, see Sample programs for WebSphere MQ for z/OS (*WebSphere MQ V7.1 Programming Guide*).

For sequences of WebSphere MQ for IBMi commands that you use regularly you can write CL programs.

For sequences of WebSphere MQ commands on Windows, UNIX and Linux systems, you can use the MQSC facility to run a series of commands held in a file.

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 WebSphere 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.

This guide mainly describes how to specify attributes using MQSC commands, and so it refers to most attributes using their MQSC command names, rather than their PCF names.

Queues

Introduction to WebSphere MQ queues and queue attributes.

A WebSphere 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 (*WebSphere MQ V7.1 Programming Guide*).

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. WebSphere 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 WebSphere MQ commands (MQSC), PCF commands, or platform-specific interfaces such as the WebSphere MQ for z/OS operations and control panels.

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 150.

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 (*WebSphere MQ V7.1 Reference*).

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 (although 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 (not supported on z/OS)
Controls whether an entry for this queue also exists in a name service

For a full description of these attributes, see Attributes for queues (*WebSphere MQ V7.1 Reference*).

Related concepts:

Developing applications reference (*WebSphere MQ V7.1 Reference*)
"Remote queues" on page 147
"Alias queues" on page 148
"Defining queues" on page 152
"Queues used by IBM WebSphere MQ" on page 152

Related reference:

The MQSC commands (*WebSphere MQ V7.1 Reference*)
"Local queues"
"Shared and cluster queues" on page 148
"Dynamic and Model queues" on page 150

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. If you are using clustering, all messages to cluster queues or cluster topics are placed on a single cluster transmission queue, `SYSTEM.CLUSTER.TRANSMIT.QUEUE`, at each cluster queue manager.

Transmission queues can trigger a message channel agent to send messages onward; see Starting WebSphere MQ applications using triggers (*WebSphere MQ V7.1 Programming Guide*).

On WebSphere 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 WebSphere 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 WebSphere MQ applications using triggers (*WebSphere MQ V7.1 Programming Guide*).

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 (*WebSphere MQ V7.1 Programming Guide*).

System command queue

The *system command queue* is a queue to which suitably authorized applications can send WebSphere 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.

On WebSphere 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

WebSphere MQ Programmable Command Formats and Administration Interface (*WebSphere MQ V7.1 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.

Channel queues

Channel queues are used for distributed queue management.

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:

- The name of the local definition that defines the remote queue.

  To create a local definition of a remote queue use the DEFINE QREMOTE command; on WebSphere MQ for IBM i, use the CRTMQMQ command.

  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.
- 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 144. These are `RemoteQName` (the name that the queue's owning queue manager knows it by), `RemoteQMgrName` (the name of the owning queue manager), and `XmitQName` (the name of the local transmission queue that is used when forwarding messages to other queue managers).

For a fuller description of these attributes, see Attributes for queues (*WebSphere MQ V7.1 Reference*).

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 a WebSphere 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 a local queue, the local definition of a remote queue, or a shared queue (a type of local queue only available on WebSphere MQ for z/OS). It can also be either a predefined queue or a dynamic queue, as supported by the platform.

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 resolve to another alias.

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.

WebSphere 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 144, 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) (*WebSphere MQ V7.1 Reference*).

The *InhibitGet* and *InhibitPut* attributes (see “Queues” on page 144) 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.

Shared and cluster queues:

This information defines and explains the terms shared queues and cluster queues, as well as providing a comparison between the two.

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. **Shared queues are available only on WebSphere MQ for z/OS.** (This is not the same as a queue being *shared* by more than one application, using the same queue manager.) Shared queues are held by a coupling facility (CF), and are accessible by any queue manager in the queue-sharing group. Each shared queue in a queue-sharing group must have a name that is unique within that group. See “Shared queues and queue-sharing groups” on page 183 for more information.

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.

The cluster queue manager makes a local queue definition for the queue specifying the name of the cluster that the queue is to be available in. This definition advertises the queue to the 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. See Cluster and Configuring a queue manager cluster (*WebSphere MQ V7.1 Installing Guide*) for further information.

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.

Mover costs

In cluster queues, messages are sent by the mover, so allow for mover 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 the mover. 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.

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 (*WebSphere MQ V7.1 Programming Guide*)).

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 WebSphere 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 a WebSphere 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 (*WebSphere MQ V7.1 Reference*).

Defining queues:

You define queues to IBM WebSphere MQ by using the MQSC command DEFINE 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

For further details about defining queue objects, see Script (MQSC) Commands (*WebSphere MQ V7.1 Administering Guide*).

Queues used by IBM WebSphere MQ:

IBM WebSphere MQ uses some local queues for specific purposes related to its operation.

You must define these queues before IBM WebSphere 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 (*WebSphere MQ V7.1 Administering Guide*), runmqtrm (*WebSphere MQ V7.1 Reference*), and Starting WebSphere MQ applications using triggers (*WebSphere MQ V7.1 Programming Guide*)

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 (*WebSphere MQ V7.1 Administering Guide*). For information about the use of transmission queues in distributed queuing, see WebSphere MQ distributed-messaging techniques (*WebSphere MQ V7.1 Installing Guide*).

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`. A definition of this queue is created by default when you define a queue manager.

A queue manager that is part of the cluster can send messages on `SYSTEM.CLUSTER.TRANSMIT.QUEUE` to any other queue manager that is in the same cluster.

During name resolution, a cluster transmission queue takes precedence over the default transmission queue.

If a queue manager is part of a cluster, the default action is to use the `SYSTEM.CLUSTER.TRANSMIT.QUEUE`, unless the destination queue manager is not part of the cluster.

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 a IBM WebSphere 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 (*WebSphere MQ V7.1 Administering Guide*).

Queue-sharing groups

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. **Supported only on WebSphere MQ for z/OS.** (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.

See “Shared queues and queue-sharing groups” on page 183 for more information.

WebSphere 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. WebSphere 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 WebSphere MQ commands. You can inquire about the values of all the attributes, except those used for Secure Sockets Layer (SSL) 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, z/OS)
- 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 WebSphere MQ commands
- The maximum length of messages the queue manager can process (fixed only in WebSphere 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 (WebSphere 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 (*WebSphere MQ V7.1 Reference*) 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 a WebSphere MQ queue manager. The process definition attributes include the application ID, the application type, and data specific to the application. See the "Initiation queues" entry under "Queues used by IBM WebSphere MQ" on page 152 for more information.

To allow an application to be started without the need for operator intervention (described in Starting WebSphere MQ applications using triggers (*WebSphere MQ V7.1 Programming Guide*)), 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; you can change the others using the WebSphere MQ commands or the WebSphere MQ for z/OS operations and control panels.

You can inquire about the values of *all* the attributes using MQINQ – Inquire object attributes (*WebSphere MQ V7.1 Reference*).

For a full description of the attributes of process definitions, see Attributes for process definitions (*WebSphere MQ V7.1 Reference*).

Clusters

You can group queue managers in a cluster. Queue managers in a cluster can make the queues that they host available to every other queue manager in the cluster. Any queue manager can send a message to any other queue manager in the same cluster without the need for many of the object definitions required for standard distributed queuing.

In a traditional WebSphere MQ network using distributed queuing, every queue manager is independent. If one queue manager needs to send messages to another queue manager, it must define a transmission queue, a channel to the remote queue manager, and a remote queue definition for every queue to which it wants to send messages.

A *cluster* is a group of queue managers set up in such a way that the queue managers can communicate directly with one another over a single network, without the need for transmission queue, channel, and remote queue definitions.

Each queue manager in the cluster has a single transmission queue from which it can transmit messages to any other queue manager in the cluster.

Related concepts:

Designing clusters

Related tasks:

Configuring a queue manager cluster (*WebSphere MQ V7.1 Installing Guide*)

Setting up a new cluster (*WebSphere MQ V7.1 Installing Guide*)

Namelists

A *namelist* is a WebSphere 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 a WebSphere MQ object that contains a list of other WebSphere 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 WebSphere MQ object.

You can define and modify namelists only using the operations and control panels of WebSphere MQ for z/OS or MQSC commands.

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.

For a full description of the attributes of namelists, see Attributes for namelists (*WebSphere MQ V7.1 Reference*).

Authentication information objects

An introduction to queue manager authentication information objects and a link to further information.

The queue manager authentication information object forms part of WebSphere MQ support for Secure Sockets Layer (SSL) and Transport Layer Security (TLS). It provides the definitions needed to check for revoked certificates. Certification Authorities revoke certificates that can no longer be trusted.

This section describes using the **setmqaut**, **dspmqaut**, **dmpmqaut**, **rcrmqobj**, **rcdmqimg**, and **dspmqfls** commands with the authentication information object. For an overview of SSL and TLS, and the use of the authentication information objects, see WebSphere MQ support for SSL and TLS (*WebSphere MQ V7.1 Administering Guide*).

For more information about SSL and TLS, see Secure Sockets Layer (SSL) and Transport Layer Security (TLS) concepts (*WebSphere MQ V7.1 Administering Guide*).

An authentication information object provides the definitions required to perform certificate revocation checking.

For a full description of the attributes of authentication information objects, see Parameter descriptions for DEFINE AUTHINFO.

Communication information objects

IBM WebSphere 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 a IBM WebSphere MQ object that contains the attributes associated with multicast transmission. For more information about these attributes, see DEFINE COMMINFO (*WebSphere MQ V7.1 Reference*). For more information about creating a COMMINFO object, see Getting started with multicast (*WebSphere MQ V7.1 Administering Guide*).

Related concepts:

"WebSphere MQ Multicast" on page 164

Channels

A *channel* is a communication link used by distributed queue managers.

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.

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.

There are two categories of channel in WebSphere MQ:

- *Message* channels, which are unidirectional, and transfer messages from one queue manager to another; see Channel control function (*WebSphere MQ V7.1 Installing Guide*) for more information.
- *MQI* channels, which are bidirectional, and transfer MQI calls from a WebSphere MQ MQI client to a queue manager, and responses from a queue manager to a WebSphere MQ client; see "What is a channel?" for more information.

Related concepts:

Administering remote WebSphere MQ objects (*WebSphere MQ V7.1 Administering Guide*)

"Concepts of intercommunication" on page 81

Related reference:

Channel-exit calls and data structures (*WebSphere MQ V7.1 Reference*)

"Communications" on page 160

What is a channel?:

A channel is a logical communication link between a WebSphere MQ MQI client and a WebSphere MQ server, or between two WebSphere MQ servers.

A channel has two definitions: one at each end of the connection. 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 WebSphere MQ, with different channel types within these categories:

Related concepts:

"Message Channels"

"MQI Channels" on page 159

"Stopping channels" on page 160

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.

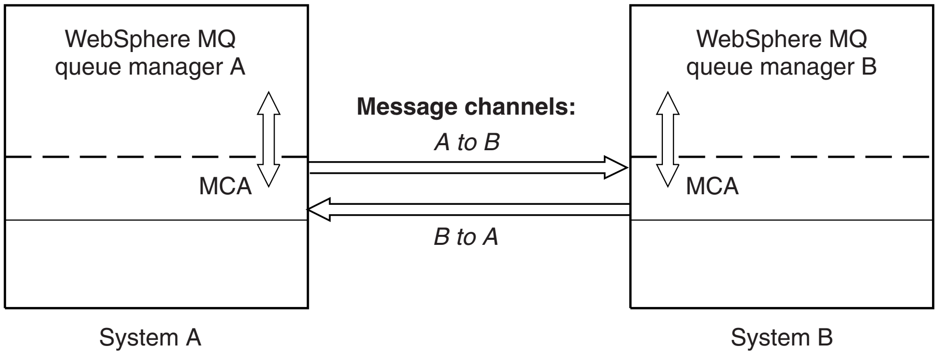


Figure 29. Message channels between two queue managers

MQI Channels:

An MQI channel connects a WebSphere MQ MQI client to a queue manager on a server machine, and is established when you issue an MQCONN or MQCONNX call from a WebSphere 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 (*WebSphere MQ V7.1 Installing Guide*)).

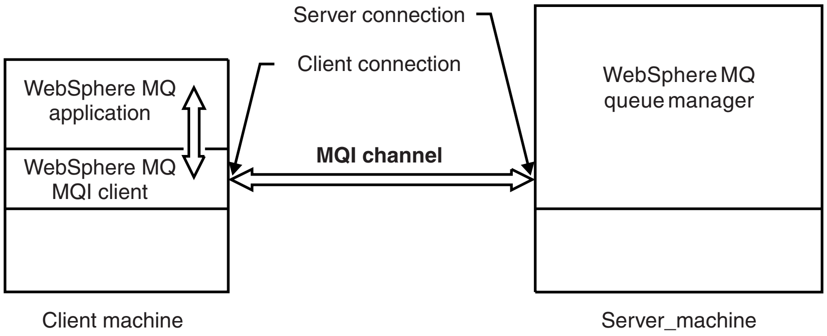


Figure 30. Client-connection and server-connection on an MQI channel

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 (*WebSphere MQ V7.1 Installing Guide*)).

There are two channel types for MQI channel definitions. They define the bi-directional MQI channel.

Client-connection channel
This type is for the WebSphere MQ MQI client.

Server-connection channel
This type is for the server running the queue manager, with which the WebSphere MQ application, running in a WebSphere MQ MQI client environment, is to communicate.

Stopping channels:

In WebSphere MQ, when you issue a STOP CHANNEL command against a server-connection channel, you can choose what method to use to stop the client-connection channel.

This means that a client channel issuing an MQGET wait call can be controlled, and you can decide how and when to stop the channel.

The STOP CHANNEL command can be issued with three modes, indicating how the channel is to be stopped:

Quiesce

Stops the channel after any current messages have been processed.

If sharing conversations is enabled, the WebSphere MQ MQI client becomes aware of the stop request in a timely manner; this time is dependent upon the speed of the network. The client application becomes aware of the stop request as a result of issuing a subsequent call to WebSphere MQ.

Force Stops the channel immediately.

Terminate

Stops the channel immediately. If the channel is running as a process, it can terminate the channel's process, or if the channel is running as a thread, its thread.

This is a multi-stage process. If mode terminate is used, an attempt is made to stop the server-connection channel, first with mode quiesce, then with mode force, and if necessary with mode terminate. The client can receive different return codes during the different stages of termination. If the process or thread is terminated, the client receives a communication error.

The return codes returned to the application vary according to the MQI call issued, and the STOP CHANNEL command issued. The client will receive either an MQRC_CONNECTION_QUIESCING or an MQRC_CONNECTION_BROKEN return code. If a client detects MQRC_CONNECTION_QUIESCING it should try to complete the current transaction and terminate. This is not possible with MQRC_CONNECTION_BROKEN. If the client does not complete the transaction and terminate fast enough it will get CONNECTION_BROKEN after a few seconds. A STOP CHANNEL command with MODE(FORCE) or MODE(TERMINATE) is more likely to result in a CONNECTION_BROKEN than with MODE(QUIESCE).

Communications:

WebSphere MQ MQI clients use MQI channels to communicate with the server.

A channel definition must be created at both the WebSphere MQ MQI client and server ends of the connection. How to create channel definitions is explained in Defining MQI channels (*WebSphere MQ V7.1 Installing Guide*).

The transmission protocols possible are shown in the following table:

Table 14. 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 WebSphere MQ MQI client and server platforms shows the possible combinations of WebSphere MQ MQI client and server platforms, using these transmission protocols.

A WebSphere MQ application on a WebSphere 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 WebSphere 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. WebSphere 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 WebSphere MQ client and server system. For dial-up support over a slow telephone line, it might be advisable to use WebSphere MQ channel compression.

Client connection channels

An introduction to client connection channel objects and a link to further information.

Client connection channels are objects that provide a communication path from a WebSphere 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 to the queue manager.

For information on client connection channels and how to use them, see “Intercommunication” on page 64.

Storage classes

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. **Supported only on WebSphere MQ for z/OS.**

For further information about storage classes, see the Planning on z/OS (*WebSphere MQ V7.1 Installing Guide*).

Listeners

Listeners are processes that accept network requests from other queue managers, or client applications, and start associated channels.

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 **runmqlsr** control command.

Listener objects are WebSphere 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.

Listener objects are not supported on WebSphere MQ for z/OS. For more information about how WebSphere MQ for z/OS implements listening, by using the channel initiator, see "The channel initiator on z/OS" on page 179.

Services

Service objects are a way of defining programs to be run when a queue manager starts or stops.

Service objects are a way of defining programs to be executed when a queue manager starts or stops. **Not supported on WebSphere MQ for z/OS.** The programs can be split into 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 WebSphere 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.

Topics

A IBM WebSphere MQ topic is a IBM WebSphere MQ object that identifies what a publication is about.

A topic is the subject of the information that is published in a publish/subscribe message.

For more information about topics, see Topics (*WebSphere MQ V7.1 Installing Guide*)

Administrative topic objects

An introduction to Administrative topic objects and links to further information.

An *administrative topic object* is a WebSphere 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 will be 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 will inherit their attributes from the base topic object, SYSTEM.BASE.TOPIC.

You can create an administrative topic object at any node in a topic tree by specifying that node's topic string in the TOPICSTR attribute of the administrative topic object. You can also define other attributes for the administrative topic node. For more information about these attributes, see the The MQSC commands (*WebSphere MQ V7.1 Reference*), or the Automating administration tasks (*WebSphere MQ V7.1 Administering Guide*). Each administrative topic object will, by default, inherit its attributes from its closest parent administrative topic node.

Administrative topic objects can also be used to hide the full topic tree from application developers. If an administrative 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 an administrative topic object.

For more information about administrative topic objects, see the Introduction to WebSphere MQ publish/subscribe messaging (*WebSphere MQ V7.1 Installing Guide*).

System default objects

An introduction to system default objects, and links to further information.

The *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.

See System and default objects (*WebSphere MQ V7.1 Reference*) for more information about system defaults.

WebSphere MQ Multicast

WebSphere 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. WebSphere MQ enables reliable Multicast messaging by using acknowledgements, negative acknowledgments, and sequence numbers to achieve low latency messaging with high fan out.

WebSphere MQ Multicast's fair delivery enables near simultaneous delivery, ensuring that no recipient gains an advantage. As WebSphere 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

WebSphere 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 WebSphere 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 (*WebSphere MQ V7.1 Reference*).

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 (*WebSphere MQ V7.1 Reference*).

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 (*WebSphere MQ V7.1 Administering Guide*)
- It is a non-managed subscription: Non-managed subscriptions are rejected with reason code 2046 (07FE) (RC2046): MQRC_OPTIONS_ERROR (*WebSphere MQ V7.1 Administering Guide*)
- It is a subscription from a non-multicast capable client: These subscriptions are rejected with reason code 2560 (0A00) (RC2560): MQRC_MULTICAST_ONLY (*WebSphere MQ V7.1 Administering Guide*)
- It is a subscription from a locally bound application: These subscriptions are rejected with reason code 2560 (0A00) (RC2560): MQRC_MULTICAST_ONLY (*WebSphere MQ V7.1 Administering Guide*)

Security

In IBM WebSphere MQ, there are four methods of providing security: the authorization service interface; user-written, or third party, channel exits; channel security using Secure Sockets Layer (SSL); and channel authentication records.

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 WebSphere MQ entities is controlled through IBM WebSphere 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 (*WebSphere MQ V7.1 Administering Guide*).

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 (*WebSphere MQ V7.1 Programming Guide*).

Channel security using SSL

The Secure Sockets Layer (SSL) protocol provides industry-standard channel security, with protection against eavesdropping, tampering, and impersonation.

SSL uses public key and symmetric techniques to provide message confidentiality and integrity and mutual authentication.

For a comprehensive review of security in IBM WebSphere MQ including detailed information about SSL, see Security (*WebSphere MQ V7.1 Administering Guide*). For an overview of SSL, including pointers to the commands described in this section, see Cryptographic security protocols: SSL and TLS (*WebSphere MQ V7.1 Administering Guide*).

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 (*WebSphere MQ V7.1 Administering Guide*).

Related concepts:

Security (*WebSphere MQ V7.1 Administering Guide*)

Planning for your security requirements (*WebSphere MQ V7.1 Administering Guide*)

Clients and servers

An introduction to how IBM WebSphere MQ supports client-server configurations for its applications.

A IBM WebSphere 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.

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

The difference between a IBM WebSphere 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 Connecting applications using distributed queuing (*WebSphere MQ V7.1 Installing Guide*).

For information about clients in general, see “Overview of IBM WebSphere MQ MQI clients” on page 167.

IBM WebSphere MQ applications in a client-server environment

When linked to a server, client IBM WebSphere 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 WebSphere MQ MQI clients (*WebSphere MQ V7.1 Programming Guide*).

Related concepts:

"Transaction management and support" on page 174

"Extending queue manager facilities" on page 175

Overview of IBM WebSphere MQ MQI clients

A *WebSphere MQ MQI client* is a component of the IBM WebSphere MQ product that can be installed on a system on which no queue manager runs.

Using a IBM WebSphere 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 a *WebSphere MQ MQI client application* and the queue manager is called a *server queue manager*.

A IBM WebSphere 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.

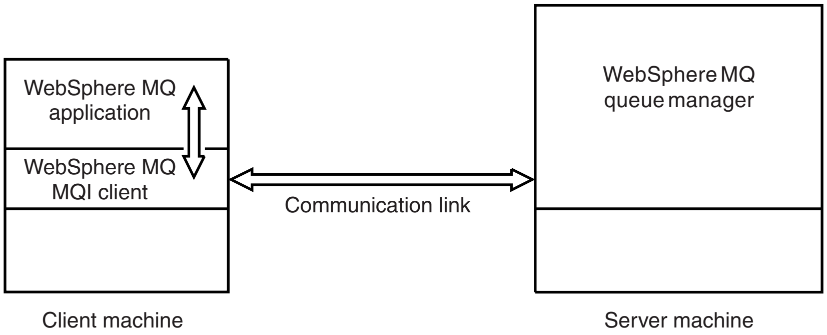


Figure 31. Link between a client and server

The following platforms can be used. The combinations depend on which IBM WebSphere MQ product you are using and are described in "Platform support for WebSphere MQ clients" on page 169.

IBM WebSphere MQ MQI client	IBM WebSphere MQ server
UNIX and Linux	UNIX and Linux
Windows	Windows
IBM i	IBM i
HP Integrity NonStop Server	z/OS
	HP Integrity NonStop Server

The MQI is available to applications running on the client platform; the queues and other IBM WebSphere 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 WebSphere MQ MQI client environment must first be linked with the relevant client library. When the application issues an MQI call, the IBM WebSphere 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 WebSphere MQ MQI client.

The link between the application and the IBM WebSphere MQ MQI client is established dynamically at run time.

You can also develop client applications using the IBM WebSphere MQ classes for .NET, IBM WebSphere MQ classes for Java or IBM WebSphere MQ classes for Java Messaging Service (JMS). You can use Java and JMS clients on IBM i, UNIX, Linux and Windows. The use of Java and JMS is not described here. For full details on how to install, configure, and use IBM WebSphere MQ classes for Java and IBM WebSphere MQ classes for JMS see Using WebSphere MQ classes for Java (*WebSphere MQ V7.1 Programming Guide*) and Using WebSphere MQ classes for JMS (*WebSphere MQ V7.1 Programming Guide*).

Related concepts:

"Why use WebSphere MQ clients?"

"How do I set up a IBM WebSphere MQ MQI client?" on page 170

"What is an extended transactional client?" on page 171

"How the client connects to the server" on page 172

Why use WebSphere MQ clients?:

Using WebSphere MQ clients is an efficient way of implementing WebSphere 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 WebSphere MQ implementation on the client machine.
- Hardware requirements on the client system are reduced.
- System administration requirements are reduced.
- a WebSphere 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 a WebSphere MQ MQI client?"

"Platform support for WebSphere MQ clients" on page 169

What applications run on a WebSphere MQ MQI client?:

The full MQI is supported in the client environment.

This enables almost any WebSphere MQ application to be configured to run on a WebSphere MQ MQI client system by linking the application on the WebSphere 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 15. 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 may have already been sent to the client and may 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 a WebSphere 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 WebSphere MQ MQI client applications to queue managers (*WebSphere MQ V7.1 Programming Guide*)).

Platform support for WebSphere MQ clients:

WebSphere MQ on all server platforms accepts client connections from WebSphere MQ MQI clients on IBM i, UNIX or Linux systems, and Windows.

WebSphere MQ installed as a *Base product and Server* (*Base product and Client Attachment feature* on WebSphere MQ for z/OS) can accept connections from the WebSphere MQ MQI clients on the following platforms:

- HP Integrity NonStop Server
- IBM i
- UNIX and Linux systems
- Windows

Client connections are subject to differences in coded character set identifier (CCSID) and communications protocol.

How do I set up a IBM WebSphere MQ MQI client?:

Follow these instructions to set up a client.

To set up a IBM WebSphere MQ MQI client you must have a IBM WebSphere 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 a IBM WebSphere MQ MQI client and that the hardware and software satisfy the requirements. Platform support is described in "Platform support for WebSphere MQ clients" on page 169.
2. Decide how you are going to install IBM WebSphere MQ on your client workstation, and then follow the instructions for your particular combination of client and server platforms. Installation is described in Installing a IBM WebSphere MQ client (*WebSphere MQ V7.1 Installing Guide*).
3. Ensure that your communication links are configured and connected. Configuration of communication links is described in Configuring connections between the server and client (*WebSphere MQ V7.1 Installing Guide*).
4. Check that your installation is working correctly. Verifying your installation is described in Verifying a client installation (*WebSphere MQ V7.1 Installing Guide*).
5. When you have the verified IBM WebSphere MQ MQI client installation, consider whether you must secure your client. Client security is described in Setting up WebSphere MQ MQI client security (*WebSphere MQ V7.1 Administering Guide*).
6. Set up the channels between the IBM WebSphere MQ MQI client and server that are required by the IBM WebSphere MQ applications you want to run on the client. Setting up channels is described in Defining MQI channels (*WebSphere MQ V7.1 Installing Guide*). There are some additional considerations if you are using SSL. These considerations are described in Specifying that an MQI channel uses SSL (*WebSphere MQ V7.1 Administering Guide*). You might need to use a IBM WebSphere MQ MQI client configuration file or IBM WebSphere MQ environment variables to set up the channels. IBM WebSphere MQ environment variables are described in Using WebSphere MQ environment variables (*WebSphere MQ V7.1 Installing Guide*).
7. IBM WebSphere MQ applications are fully described in the Developing applications (*WebSphere MQ V7.1 Programming Guide*).
8. There are some differences from a queue manager environment to consider when designing, building, and running applications in the IBM WebSphere MQ MQI client environment. For information about these differences, see:
   - Using the message queue interface (MQI) in a client application (*WebSphere MQ V7.1 Programming Guide*)
   - Building applications for WebSphere MQ MQI clients (*WebSphere MQ V7.1 Programming Guide*)
   - Connecting IBM WebSphere MQ MQI client applications to queue managers (*WebSphere MQ V7.1 Programming Guide*)
   - Resolving problems with IBM WebSphere MQ MQI clients (*WebSphere MQ V7.1 Administering Guide*)

What is an extended transactional client?:

A WebSphere 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 174.

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 a WebSphere MQ extended transactional client removes this restriction.

A *WebSphere MQ extended transactional client* is a IBM WebSphere 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 a WebSphere 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, a WebSphere MQ MQI client that does not have the extended transactional function is referred to as a *WebSphere MQ base client*. You can consider, therefore, a WebSphere MQ extended transactional client to consist of a WebSphere MQ base client with the addition of the extended transactional function.

Note: IBM WebSphere MQ MQI client on IBM i does not support the WebSphere MQ extended transactional function.

Related reference:

"Platform support for extended transactional clients"

Platform support for extended transactional clients:

IBM WebSphere MQ extended transactional clients are available for all the platforms that support a base client except z/OS.

A client application that is using an extended transactional client can connect to a queue manager of the following IBM WebSphere MQ Version 7.1 products only:

- IBM WebSphere MQ for AIX
- IBM WebSphere MQ for HP-UX
- IBM WebSphere MQ for HP Integrity NonStop Server

- IBM WebSphere MQ for IBM i
- IBM WebSphere MQ for Linux
- IBM WebSphere MQ for Solaris
- IBM WebSphere 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 WebSphere MQ base client. A programming language is supported by an extended transactional client if it is supported by the IBM WebSphere MQ base client and by the transaction manager you are using.

The external transaction managers for each platform are listed on the following web pages.

Extended transactional client platform	Web page
AIX	Minimum Requirements for WebSphere MQ V7.1 - AIX
HP-UX	Minimum Requirements for WebSphere MQ V7.1 - HP-UX
HP Integrity NonStop Server	Planning your IBM WebSphere MQ client environment on HP Integrity NonStop Server (*WebSphere MQ V7.1 Installing Guide*)
IBM i	Minimum Requirements for WebSphere MQ V7.1 - IBM i
Linux	Minimum Requirements for WebSphere MQ V7.1 - Linux
Solaris	Minimum Requirements for WebSphere MQ V7.1 -Solaris
Windows	Minimum Requirements for WebSphere MQ V7.1 -Windows

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 WebSphere 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"

"Clients on different platforms" on page 173

"Using different versions of client and server software" on page 173

Client and queue manager on the same machine:

You can also run an application in the WebSphere 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 WebSphere MQ MQI client environment.

Clients on different platforms:

Here is another example of a WebSphere MQ MQI client and server system. In this example, the server machine communicates with three WebSphere MQ MQI clients on different platforms.

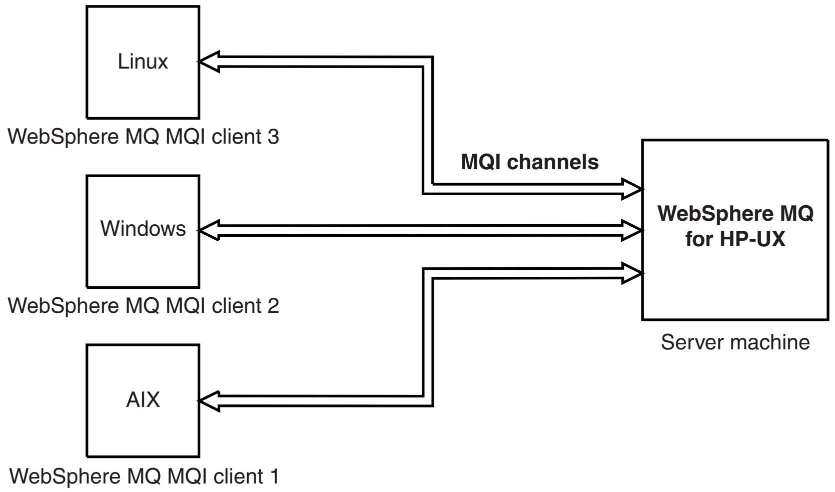

Linux
WebSphere MQ MQI client 3
Windows
WebSphere MQ MQI client 2
AIX
WebSphere MQ MQI client 1
MQI channels
WebSphere MQ for HP-UX
Server machine

Figure 32. WebSphere MQ server connected to clients on different platforms

Other more complex environments are possible. For example, a WebSphere 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 WebSphere MQ products, make sure that code conversion from the CCSID of your client is supported by the server.

A IBM WebSphere MQ Version 7.0 client can connect to all supported versions of queue manager, whether Version 7.0 or earlier. If you are connecting to an earlier version queue manager, you cannot use Version 7.0 features and structures in your IBM WebSphere MQ application on the client.

A IBM WebSphere MQ Version 7.0 queue manager accepts connections from all supported versions of client, whether Version 7.0 or earlier.

See the programming languages supported in Deciding which programming language to use (*WebSphere MQ V7.1 Programming Guide*) for more information.

Transaction management and support

An introduction to transaction management and how WebSphere MQ supports transactions.

A *resource manager* is a computer subsystem that owns and manages resources that can be accessed and updated by applications. The following are examples of resource managers:

- A WebSphere 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 *sync point coordinator*.

A *local* unit of work is one in which the only resources updated are those of the WebSphere 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, a WebSphere 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 (*WebSphere MQ V7.1 Programming Guide*).

WebSphere MQ also provides support for the Microsoft Transaction Server (COM+). Using the Microsoft Transaction Server (COM+) (*WebSphere MQ V7.1 Programming Guide*) provides information on how to set up WebSphere 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 (*WebSphere MQ V7.1 Programming Guide*).

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 (*WebSphere MQ V7.1 Programming Guide*).

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 (*WebSphere MQ V7.1 Reference*).

API exits

API exits let you write code that changes the behavior of WebSphere 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 (*WebSphere MQ V7.1 Programming Guide*).

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 WebSphere 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 (*WebSphere MQ V7.1 Administering Guide*).

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 WebSphere 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 WebSphere MQ client for HP Integrity NonStop Server technical overview

A technical overview of the HP Integrity NonStop Server operating system.

IBM WebSphere MQ client for HP Integrity NonStop Server SupportPac

The IBM WebSphere MQ client for HP Integrity NonStop Server is released in SupportPac MAT1.

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 http://h20223.www2.hp.com/NonStopComputing/cache/76385-0-0-0-121.html.

IBM WebSphere MQ client for HP Integrity NonStop Server supported environments and features

Provides details about the IBM WebSphere MQ client for the HP Integrity NonStop Server platform and describes supported client API and environments and client functionality specific to HP Integrity NonStop Server systems.

Supported client API and environments

IBM WebSphere MQ client for HP Integrity NonStop Server supports the following execution environments:

Table 16.

	OSS	**Guardian**
C	✓	✓
JMS	✓	
COBOL	✓	✓
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 WebSphere MQ client for HP Integrity NonStop Server:

- C (native), PTAL, COBOL (native)
  - Network Protocol: TCP (IPv4 and IPv6)
  - Transport Type: Client only
  - Transport Security: SSL/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: SSL/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 written in other languages are not supported)
- IBM WebSphere MQ Headers and PCF: The following classes are not supported: com.ibm.mq.headers.* and com.ibm.mq.pcf.*

WebSphere MQ for z/OS concepts

Some of the concepts used by WebSphere 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 WebSphere MQ for z/OS. Use this topic for further information about these concepts.

Related concepts:

"The queue manager on z/OS"

"The channel initiator on z/OS" on page 179

"Terms and tasks" on page 181

"Shared queues and queue-sharing groups" on page 183

"Intra-group queuing" on page 231

"Storage management" on page 244

"Logging" on page 249

"Recovery and restart" on page 269

"Security concepts on z/OS" on page 286

"Availability" on page 292

"Unit of recovery disposition" on page 296

Related reference:

"Defining your system" on page 259

"Monitoring and statistics" on page 295

The queue manager on z/OS

Before you can let your application programs use WebSphere MQ on your z/OS system, you must install the WebSphere MQ for z/OS product and start a queue manager. The queue manager owns and manages the set of resources that are used by WebSphere 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 WebSphere 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 WebSphere MQ API
- The WebSphere MQ channel initiator, which allows communication between WebSphere 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 33 illustrates a queue manager, showing connections to different application environments, and the channel initiator.

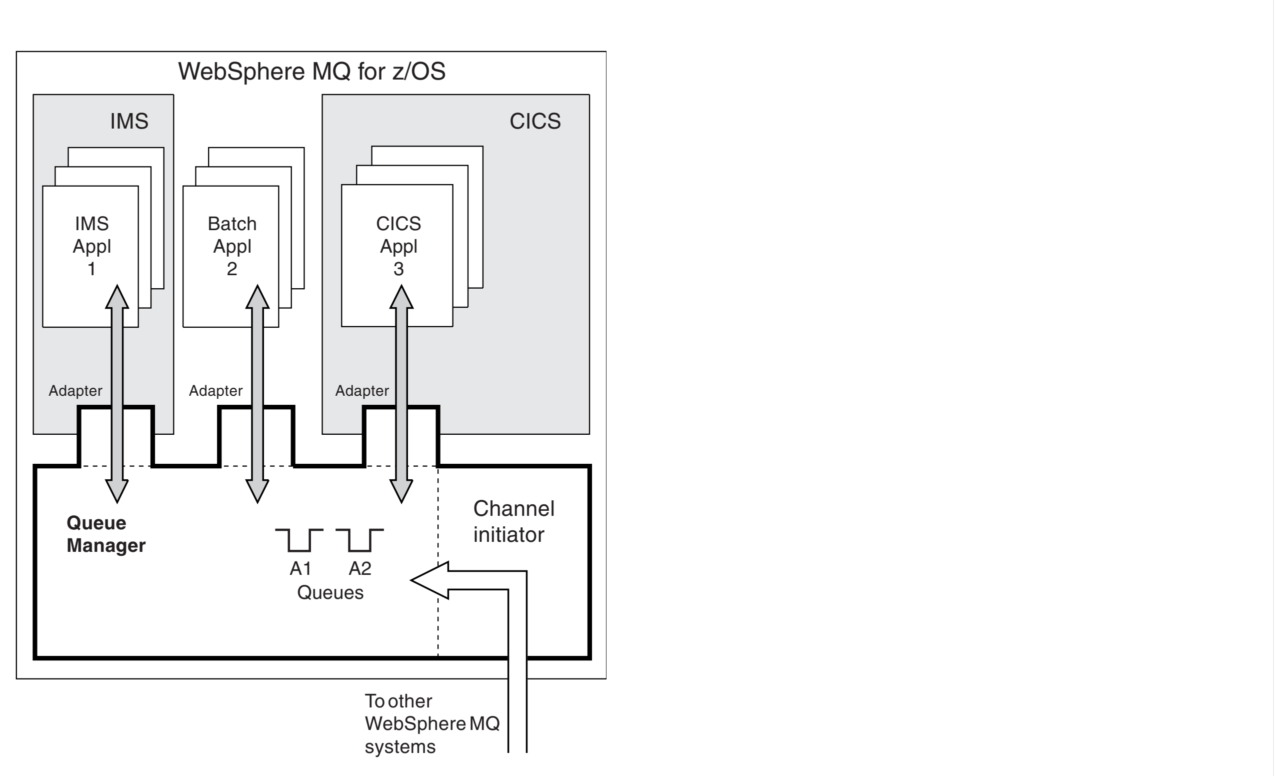


Figure 33. Overview of WebSphere MQ for z/OS

The queue manager subsystem on z/OS

On z/OS, WebSphere 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 WebSphere MQ distributed queuing. WebSphere 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 WebSphere 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 34 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.

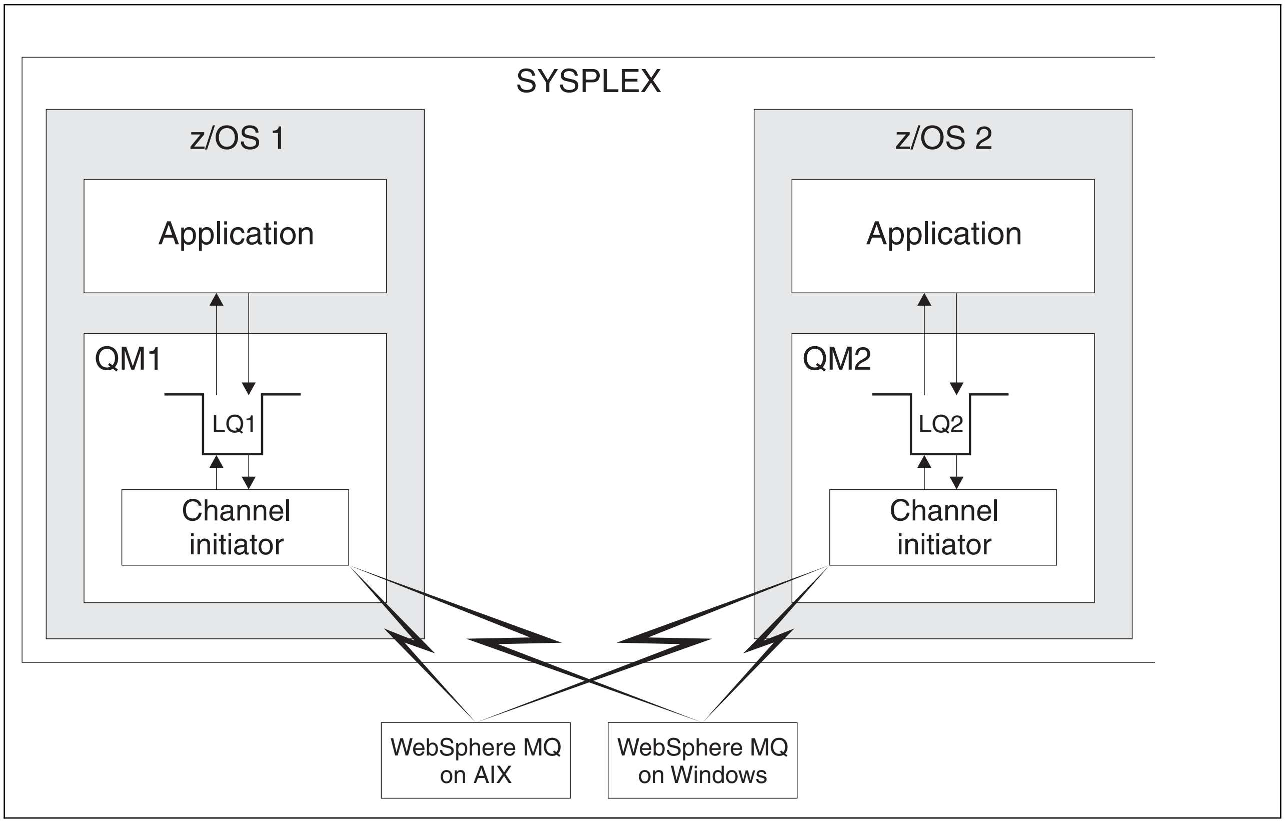


Figure 34. Communication between queue managers

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.

SSL tasks
These are used to perform encryption and decryption and check certificate revocation lists.

Terms and tasks

Use this topic as an introduction to the terminology, and tasks that are specific to WebSphere MQ for z/OS.

Some of the terms and tasks required for managing WebSphere 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 WebSphere 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 WebSphere 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” on page 249.

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 WebSphere MQ commands are read, and the commands they contain are executed. Typically, these data sets contain definitions of the system objects required for WebSphere MQ to run, and you can tailor these to define or initialize the WebSphere 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 page 259.

Recovery and restart

At any time during the operation of WebSphere 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 WebSphere 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 page 269.

Security

You can use an external security manager, such as Security Server (previously known as RACF) to protect the resources that WebSphere MQ owns and manages from access by unauthorized users. You can also use the Secure Sockets Layer (SSL) for channel security. SSL is included as part of the WebSphere MQ product.

For more information about WebSphere MQ security, see “Security concepts on z/OS” on page 286.

Availability

There are several features of WebSphere 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 page 292.

Manipulating objects

When the queue manager is running, you can manipulate WebSphere 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 WebSphere MQ objects. You can also control and display the status of various WebSphere MQ and queue manager functions.

You can also manipulate WebSphere MQ objects using the WebSphere 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 (*WebSphere MQ V7.1 Administering Guide*).

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 page 295.

Application environments

When the queue manager has started, applications can connect to it and start using the WebSphere MQ API. These can be CICS, IMS, Batch, or WebSphere Application Server applications. WebSphere MQ applications can also access applications on CICS and IMS systems that are not aware of WebSphere MQ, using the CICS and IMS bridges.

For more information about these facilities, see “IBM WebSphere MQ and other z/OS products” on page 299.

For information about writing WebSphere MQ applications, see the following documentation:

- Developing applications (*WebSphere MQ V7.1 Programming Guide*)
- Using C++ (*WebSphere MQ V7.1 Programming Guide*)
- Using WebSphere MQ classes for Java (*WebSphere MQ V7.1 Programming Guide*)

Shared queues and queue-sharing groups

You can use shared queues and queue-sharing groups, to implement high availability of MQ resources.

Shared queues and queue-sharing groups are functions unique to WebSphere® 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?”

“What is a queue-sharing group?” on page 185

“Where are shared queue messages held?” on page 187

“Advantages of using shared queues” on page 204

“Distributed queuing and queue-sharing groups” on page 224

“Influencing workload distribution with shared queues” on page 229

Related reference:

“Where to find more information about these concepts” on page 230

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.

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.

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 WebSphere MQ, large messages (> 63 KB) have a placeholder stored in the coupling facility (4 K), and their message data stored in Db2.

Figure 35 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.

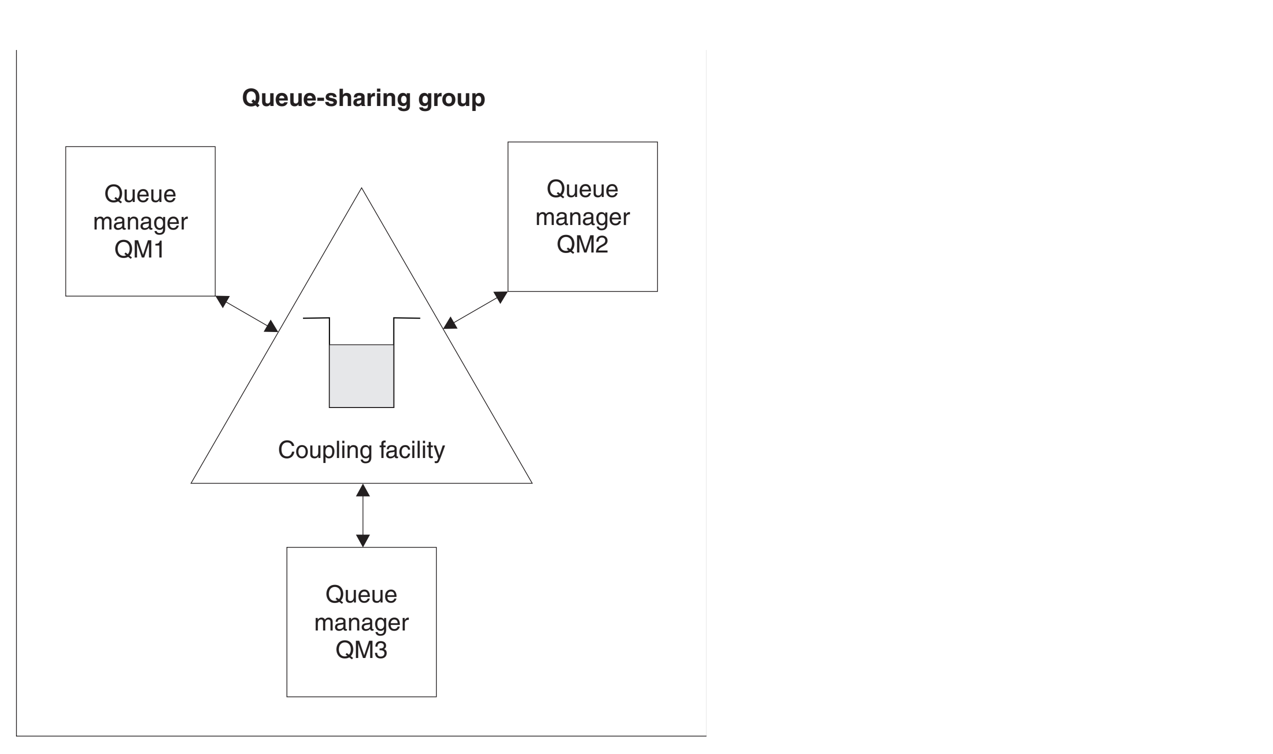


Figure 35. A queue-sharing group

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?:

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 36 on page 186 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 WebSphere 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 (*WebSphere MQ V7.1 Administering Guide*).

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 WebSphere MQ system parameters at startup.

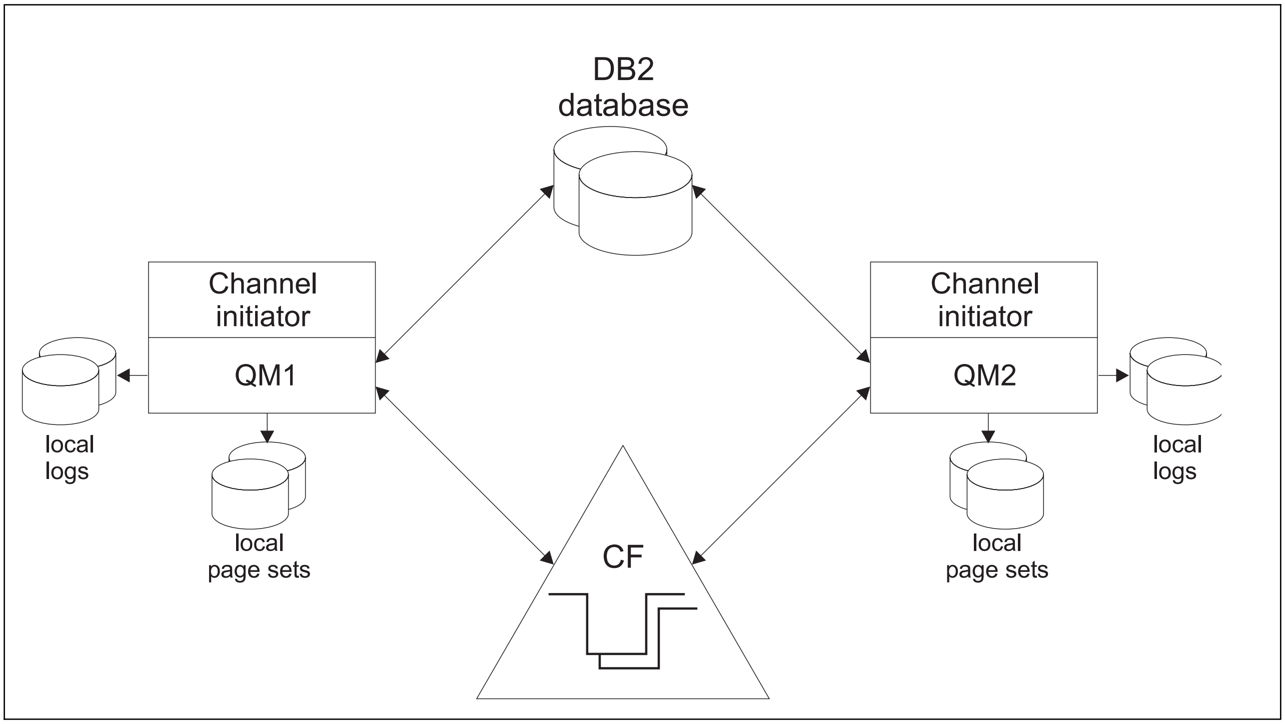


Figure 36. The components of queue managers in a queue-sharing group

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 (*WebSphere MQ V7.1 Administering Guide*).

When a queue manager runs as a member of a queue-sharing group it must be possible to distinguish between WebSphere MQ objects defined privately to that queue manager and WebSphere 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 (*WebSphere MQ V7.1 Administering Guide*).

You can define a single set of security profiles that control access to WebSphere 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?:

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.

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 189 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 WebSphere 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 WebSphere MQ and other products.

Planning the CF structure size

If you require guidance on the sizing of your CF structures you can use the MP16: WebSphere 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) WebSphere 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, a WebSphere 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 WebSphere 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 WebSphere 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 (*WebSphere MQ V7.1 Reference*) and RECOVER CFSTRUCT (*WebSphere MQ V7.1 Reference*) commands for more details.

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 205 for more information.

Related concepts:

"Specifying offload options for shared messages"

"Managing your shared message data set (SMDS) environment" on page 191

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 (CF).

The message data for shared queues can be offloaded from the coupling facility and stored in either a Db2 table or in a WebSphere 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 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 WebSphere MQ Version 7.1 or higher.

The **OFFLOAD** parameter is only valid from **CFLEVEL(5)**. See DEFINE CFSTRUCT (*WebSphere MQ V7.1 Reference*) 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. There are three rules, and each rule specifies a matching pair of parameters. These parameters are a corresponding coupling facility 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 is more than 70% full offload data for messages exceeding 32 KB
Rule pair 2	OFFLD2TH(80) and OFFLD2SZ(4K)	If the coupling facility is more than 80% full offload data for messages exceeding 4 KB
Rule pair 3	OFFLD3TH(90) and OFFLD3SZ(0K)	If the coupling facility 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 above 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 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 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 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 WebSphere 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 203

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 recreated 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 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 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 recreated 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 recreated. 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 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 below.

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.

[1] `(Hits÷(Hits+Misses))×100`

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 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 been 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 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 was simply that the data set contained residual data from another use and had not been preformatted since, then this can be fixed simply by preformatting the data set then using the **RESET SMDS** command to change the status to **STATUS(RECOVERED)**.

Otherwise, the data set needs to be recovered, as described below.

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 recreated 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 (*WebSphere MQ V7.1 Reference*)
- DEFINE CFSTRUCT (*WebSphere MQ V7.1 Reference*)
- ALTER CFSTRUCT (*WebSphere MQ V7.1 Reference*)
- DELETE CFSTRUCT (*WebSphere MQ V7.1 Reference*)

Display **CFSTRUCT** status relating to large message offload **(OFFLDUSE)**:

- DISPLAY CFSTATUS (*WebSphere MQ V7.1 Reference*)

Display and alter override data set options (**DSEXPAND** and **DSBUFS**) for individual queue managers:

- DISPLAY SMDS (*WebSphere MQ V7.1 Reference*)
- ALTER SMDS (*WebSphere MQ V7.1 Reference*)

Display or modify the status and availability of the data sets within the queue-sharing group:

- DISPLAY CFSTATUS TYPE(SMDS) (*WebSphere MQ V7.1 Reference*)
- RESET SMDS (*WebSphere MQ V7.1 Reference*)

Display SMDS data set space usage and buffer usage information for a queue manager:

- DISPLAY USAGE TYPE(SMDS) (*WebSphere MQ V7.1 Reference*)

Display or modify the status and availability of the connections (**SMDSCONN**) to the data sets from an individual queue manager:

- DISPLAY SMDSCONN (*WebSphere MQ V7.1 Reference*)
- START SMDSCONN (*WebSphere MQ V7.1 Reference*)
- STOP SMDSCONN (*WebSphere MQ V7.1 Reference*)

Backup and recover shared messages, including large message data in SMDS when necessary:

- BACKUP CFSTRUCT (*WebSphere MQ V7.1 Reference*)
- RECOVER CFSTRUCT (*WebSphere MQ V7.1 Reference*)

Advantages of using shared queues:

Shared queue allows for WebSphere 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 a WebSphere 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.

WebSphere 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 37 on page 205.

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 a WebSphere 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 224.

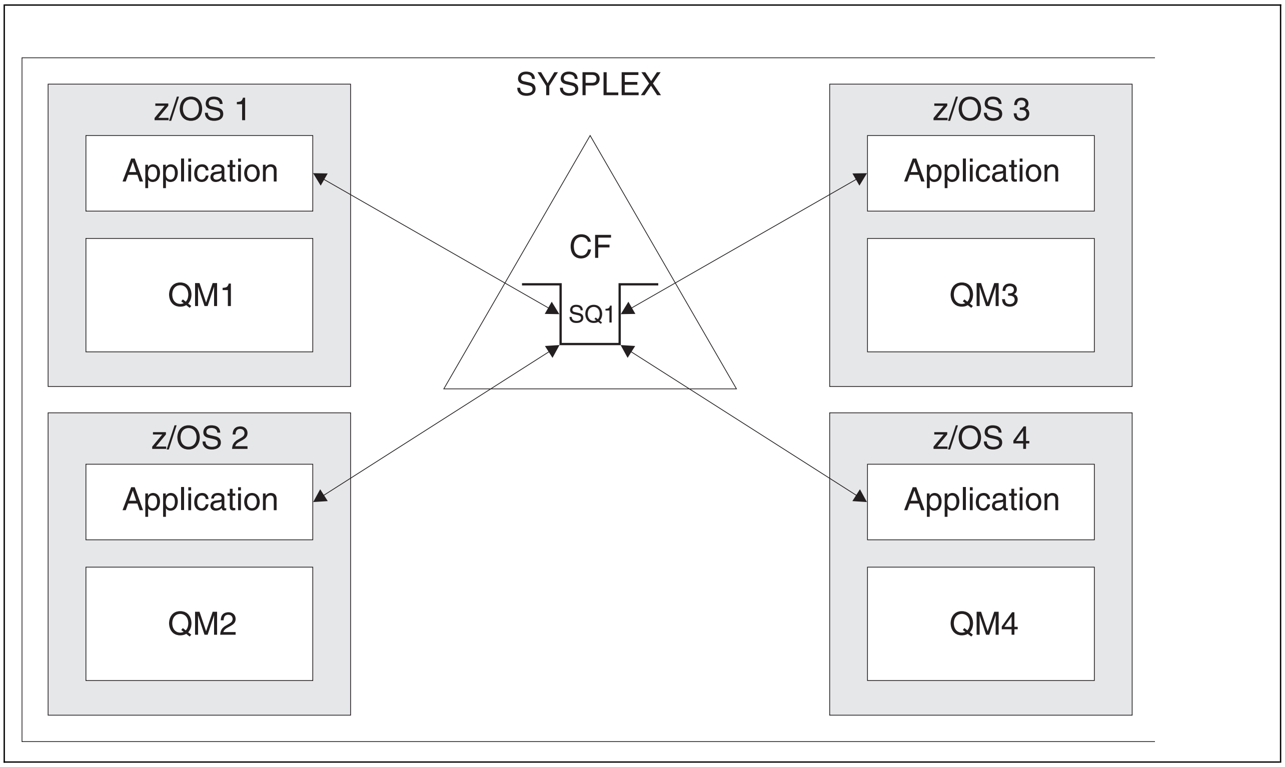


Figure 37. Multiple instances of an application servicing a shared queue

Peer recovery

To further enhance the availability of messages in a queue-sharing group, WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere MQ shared queues use CF list structures to store messages. IBM WebSphere MQ calls CF structures, which contain messages and application structures.

Application structures are referenced using the information stored in IBM WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 (*WebSphere MQ V7.1 Reference*) 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)
OFFLD1SZ(32K) OFFLD1TH(70)
OFFLD2SZ(4K) OFFLD2TH(80)
OFFLD3SZ(0K) OFFLD3TH(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 WebSphere MQ for z/OS:

An overview of the use of storage class memory (SCM) with IBM WebSphere 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 WebSphere 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 WebSphere 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.

Why use SCM:

Emergency storage and improved performance are two use cases for using SCM with IBM WebSphere 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 213
- “Improved performance - basic configuration” on page 219

Important: The use of SCM with CF structures is not dependent on any specific version of IBM WebSphere 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 WebSphere 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 207 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 WebSphere 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 WebSphere 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 38 on page 214.

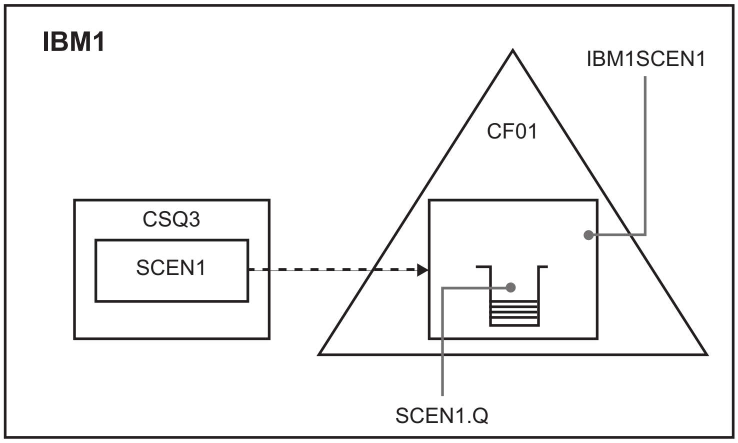


Figure 38. Basic configuration

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 WebSphere 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 WebSphere MQ to make use of the structure defined in the CFRM policy.

   a. Use the DEFINE CFSTRUCT (*WebSphere MQ V7.1 Reference*) command, with the structure name of SCEN1 to create a IBM WebSphere 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 (*WebSphere MQ V7.1 Reference*) 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 WebSphere 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 205

Adding SMDS and SCM to the initial structure:

How you add SMDS and SCM for emergency storage on IBM WebSphere MQ.

About this task

This part of the task uses the basic configuration described in "Emergency storage - basic configuration" on page 213. 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 39 on page 216.

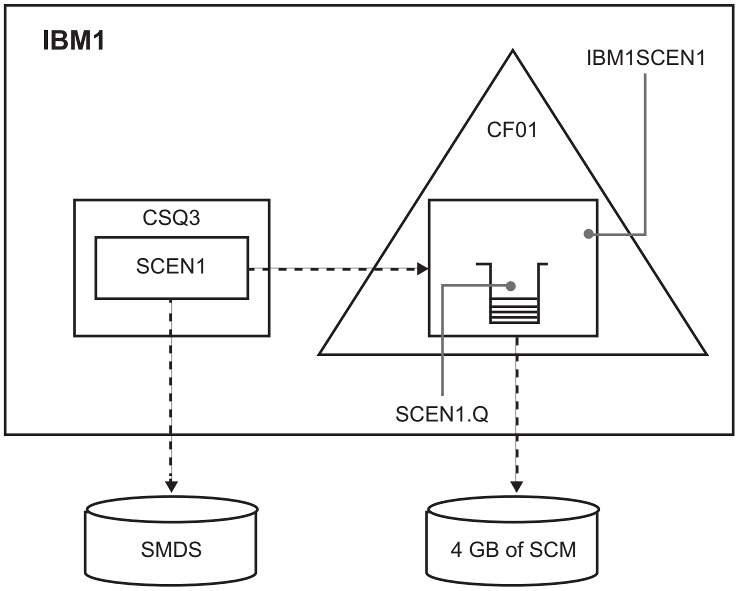


Figure 39. Configuration adding SMDS and SCM for emergency storage

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 (*WebSphere MQ V7.1 Reference*) 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 : IXCLO053
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 WebSphere 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 17. 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	1,135775	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 WebSphere 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 40 on page 220.

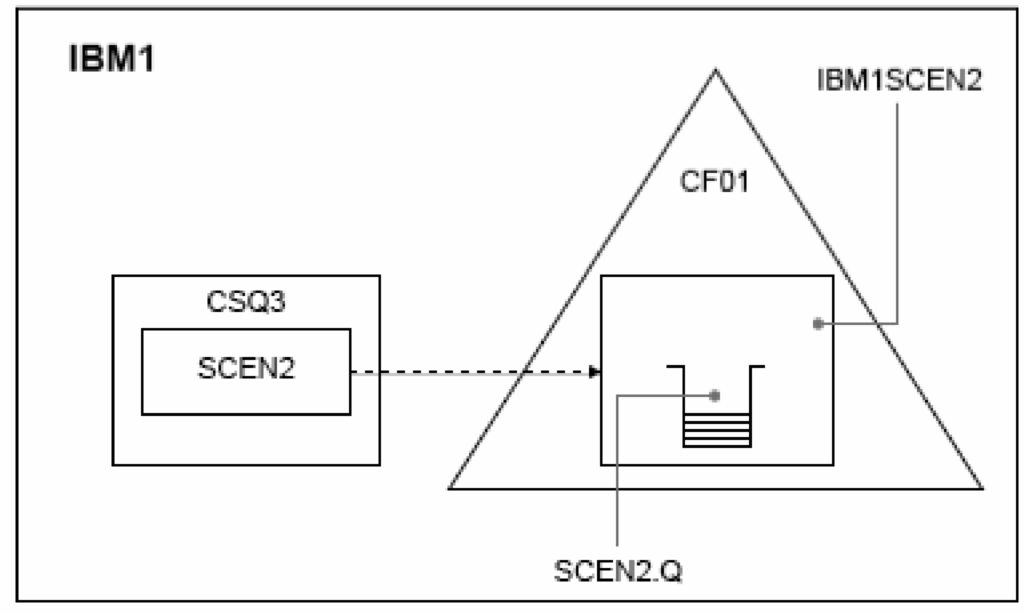


Figure 40. Basic configuration

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 WebSphere MQ to make use of the structure defined in the CFRM policy.
   a. Use the DEFINE CFSTRUCT (*WebSphere MQ V7.1 Reference*) command, with the structure name of SCEN2 to create a IBM WebSphere 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 (*WebSphere MQ V7.1 Reference*) 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 WebSphere 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.”

Related concepts:

“Use of storage class memory with shared queues” on page 205

Adding SCM to the initial structure:

How you add SCM for improved performance on IBM WebSphere MQ.

About this task

This part of the task uses the basic configuration described in “Improved performance - basic configuration” on page 219. The scenario describes the addition of SCM to the initial structure.

This final configuration is illustrated in Figure 41.

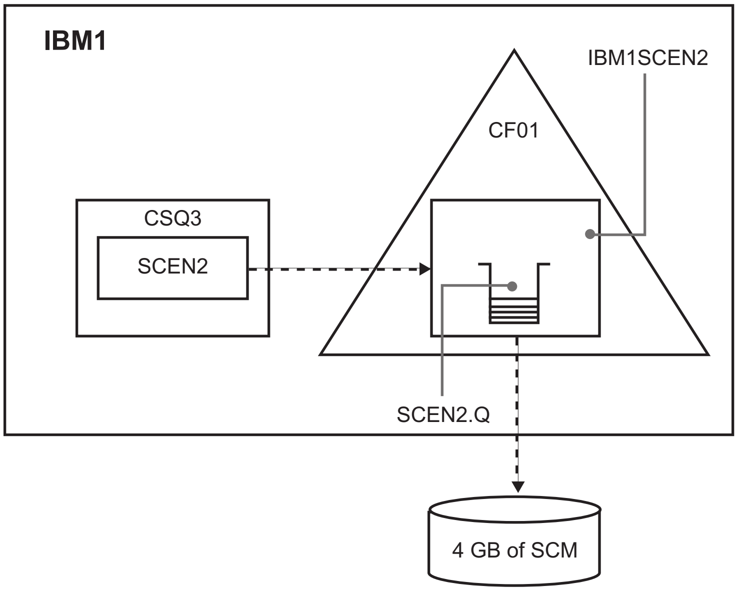


Figure 41. Configuration adding SCM for improved performance

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:
      ```
      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(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 219:
  - 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 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 219.

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:

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 WebSphere 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 42 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.

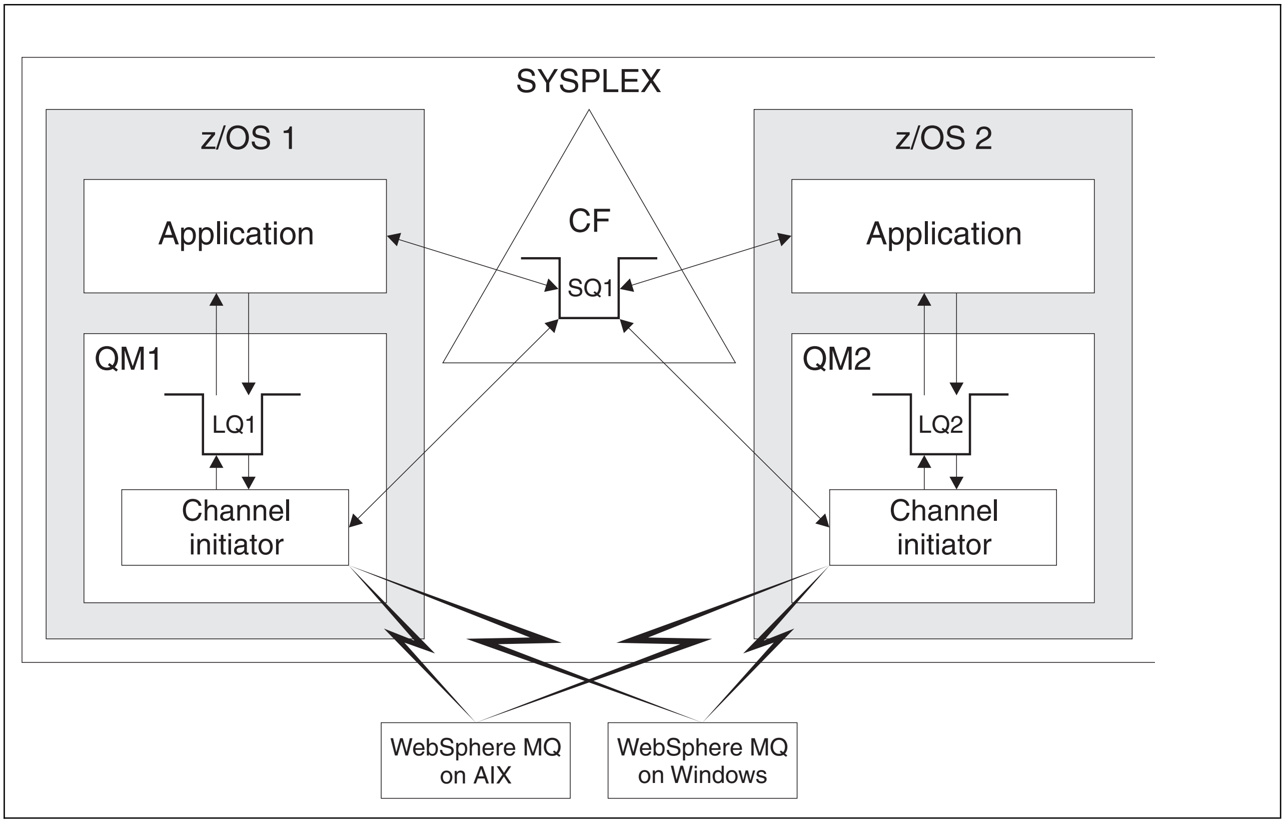


Figure 42. Distributed queuing and queue-sharing groups

Related concepts:
"Shared channels"
"Intra-group queuing" on page 227
"Clusters and queue-sharing groups" on page 228

Shared channels:

Use this topic to understand the concepts of shared channels and their use with WebSphere 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
- TCP/IP Domain Name System (DNS)
- 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), unless stored on the shared repository and not available anywhere. 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 QSG 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 WebSphere 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, WebSphere 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:

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 43 on page 229 shows how members of a cluster can access a shared queue through any member of the queue-sharing group.

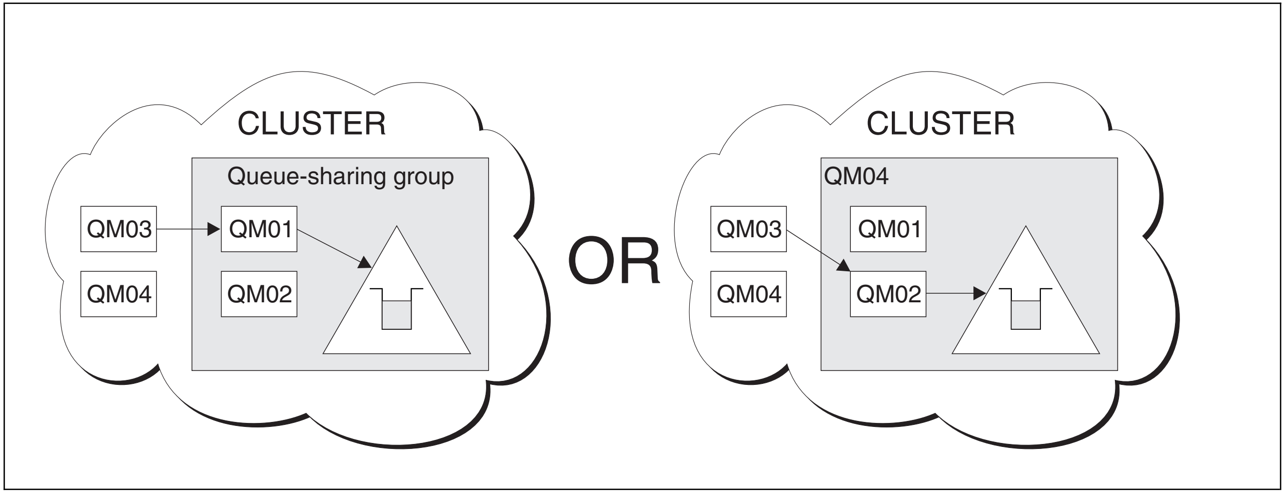


Figure 43. A queue-sharing group as part of a cluster

Influencing workload distribution with shared queues:

Use this topic to understand the factors that affect workload distribution with shared queues in a queue-sharing group.

WebSphere 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 QSG 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 QSG 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 QSG 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 QSG 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 QSG 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 performance enhancements described above. 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 these concepts:

Use the table in this topic to find more information about the other topics in this product documentation.

You can find more information about the topics contained in this product documentation from the following sources:

Table 18. Where to find more information about shared queues and queue-sharing groups

Topic	**Where to look**
Queue-sharing group recovery	"Recovery and restart" on page 269
Queue-sharing group security	"Security concepts on z/OS" on page 286
Private and global object definitions Directing commands to different queue managers	Issuing commands (*WebSphere MQ V7.1 Administering Guide*)
Planning your coupling facility environment	Defining coupling facility resources
Planning your SMDS environment	Planning your shared message data set (SMDS) environment (*WebSphere MQ V7.1 Installing Guide*)
Planning your Db2 environment	Planning your Db2 environment (*WebSphere MQ V7.1 Installing Guide*)

Table 18. Where to find more information about shared queues and queue-sharing groups (continued)

Topic	Where to look
Setting up your shared queues System parameters	“Shared queues and queue-sharing groups” on page 183
Utility programs Migrating queues	Using the WebSphere MQ Utilities (*WebSphere MQ V7.1 Reference*)
Console messages	Messages for WebSphere MQ for z/OS (*WebSphere MQ V7.1 Reference*)
MQSC commands	MQSC reference (*WebSphere MQ V7.1 Reference*)
WebSphere MQ clusters	Configuring a queue manager cluster (*WebSphere MQ V7.1 Installing Guide*)
WebSphere MQ distributed queuing Channel names	Introduction to distributed queue management (*WebSphere MQ V7.1 Installing Guide*)
Writing applications	Overview of application design (*WebSphere MQ V7.1 Programming Guide*)
MQCONNX call	MQCONNX (*WebSphere MQ V7.1 Reference*)

Intra-group queuing

This section describes intra-group queuing, a IBM WebSphere 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 183.

Related concepts:

“Concepts”

“Benefits” on page 233

“Limitations” on page 235

“Getting started” on page 235

“Configurations” on page 236

“Intra-group queuing messages” on page 241

“Security” on page 243

“Specific properties” on page 243

Related information:

“Terminology of intra-group queuing” on page 233

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.

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.

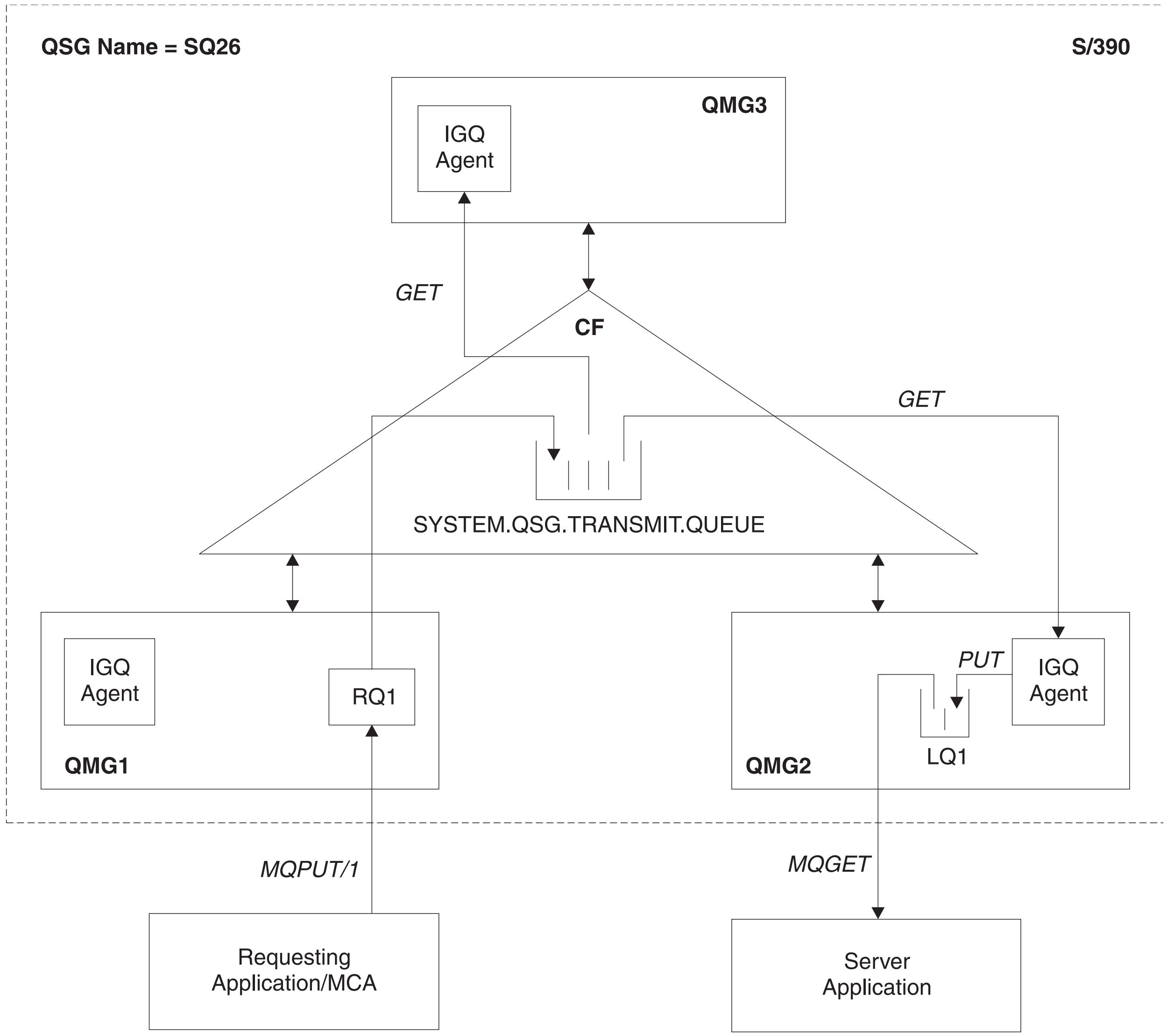


Figure 44. An example of intra-group queuing

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.

Terminology of intra-group queuing:

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:

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.

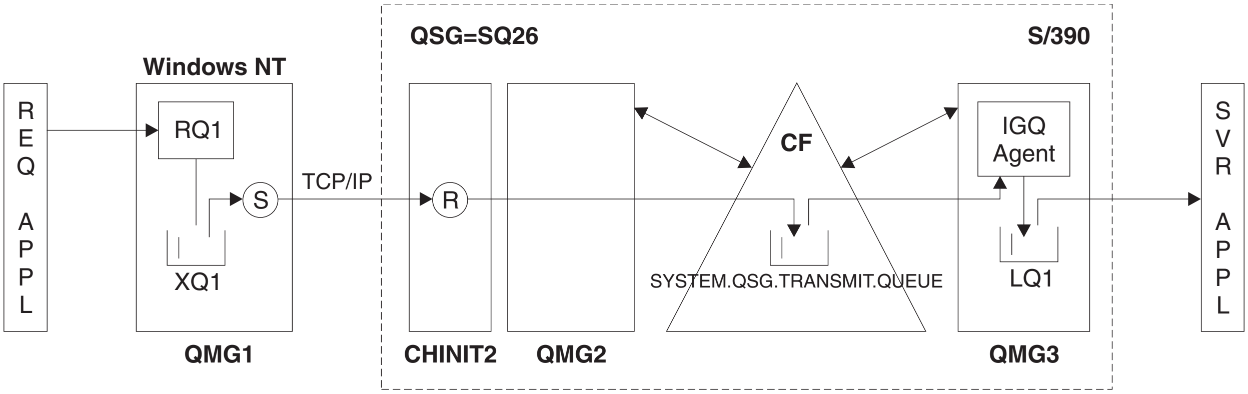


Figure 45. An example of migration support

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:

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.

Getting started:

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.SCSQPROCS(CSQ4INSS), the CSQINP2 sample for SYSTEM objects for queue-sharing groups. This queue must be defined with the correct attributes, as stated in thlqual.SCSQPROCS(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.

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.

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.

Configurations:

In addition to the typical intra-group queuing configuration, other configurations are possible.

Figure 44 on page 232 describes the typical configuration.
Related concepts:
"Distributed queuing with intra-group queuing (multiple delivery paths)"
"Clustering with intra-group queuing (multiple delivery paths)" on page 238
"Clustering, intra-group queuing and distributed queuing" on page 240

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.

The choice of intra-group queueing 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.IGQ.TRANSMIT.QUEUE.

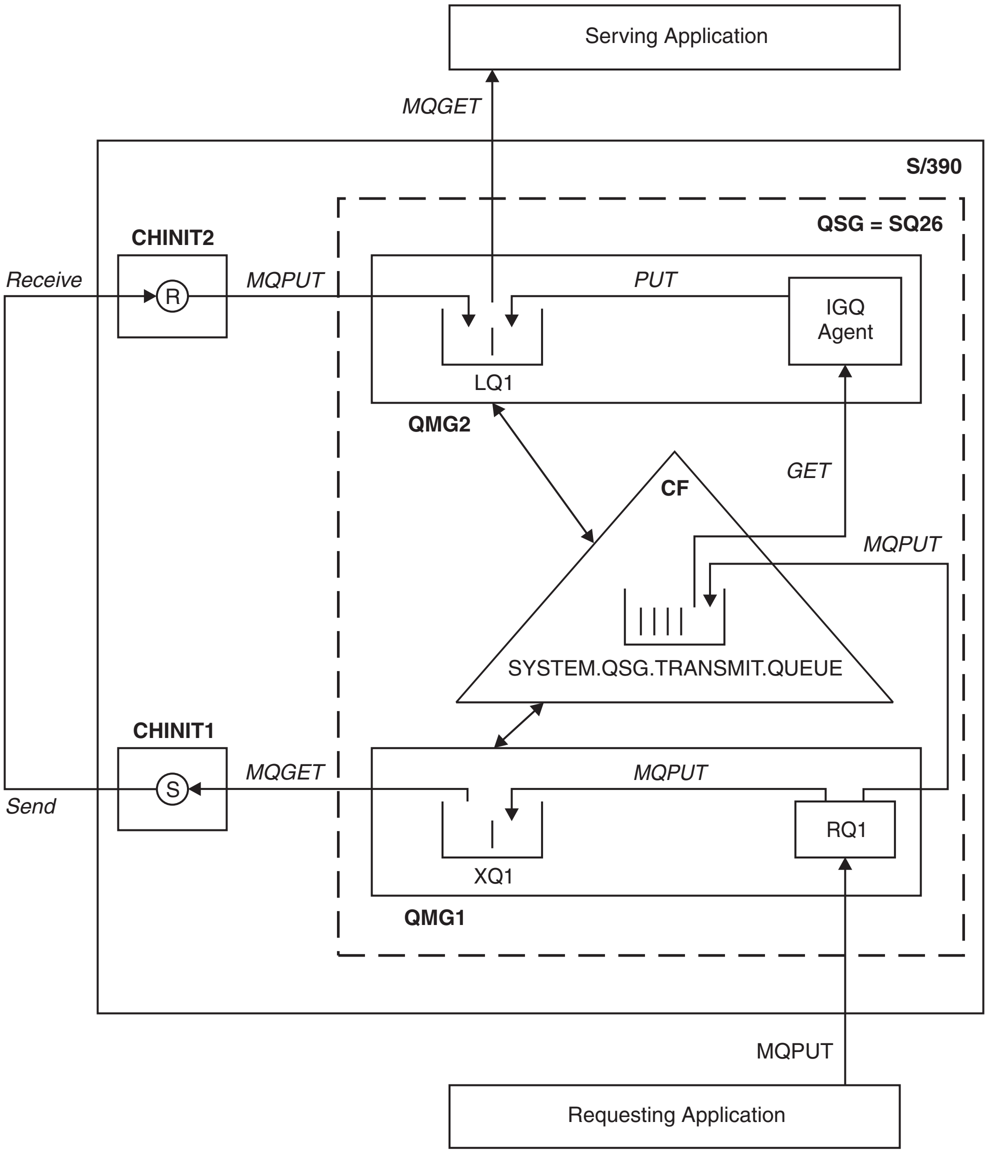


Figure 46. An example configuration

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).

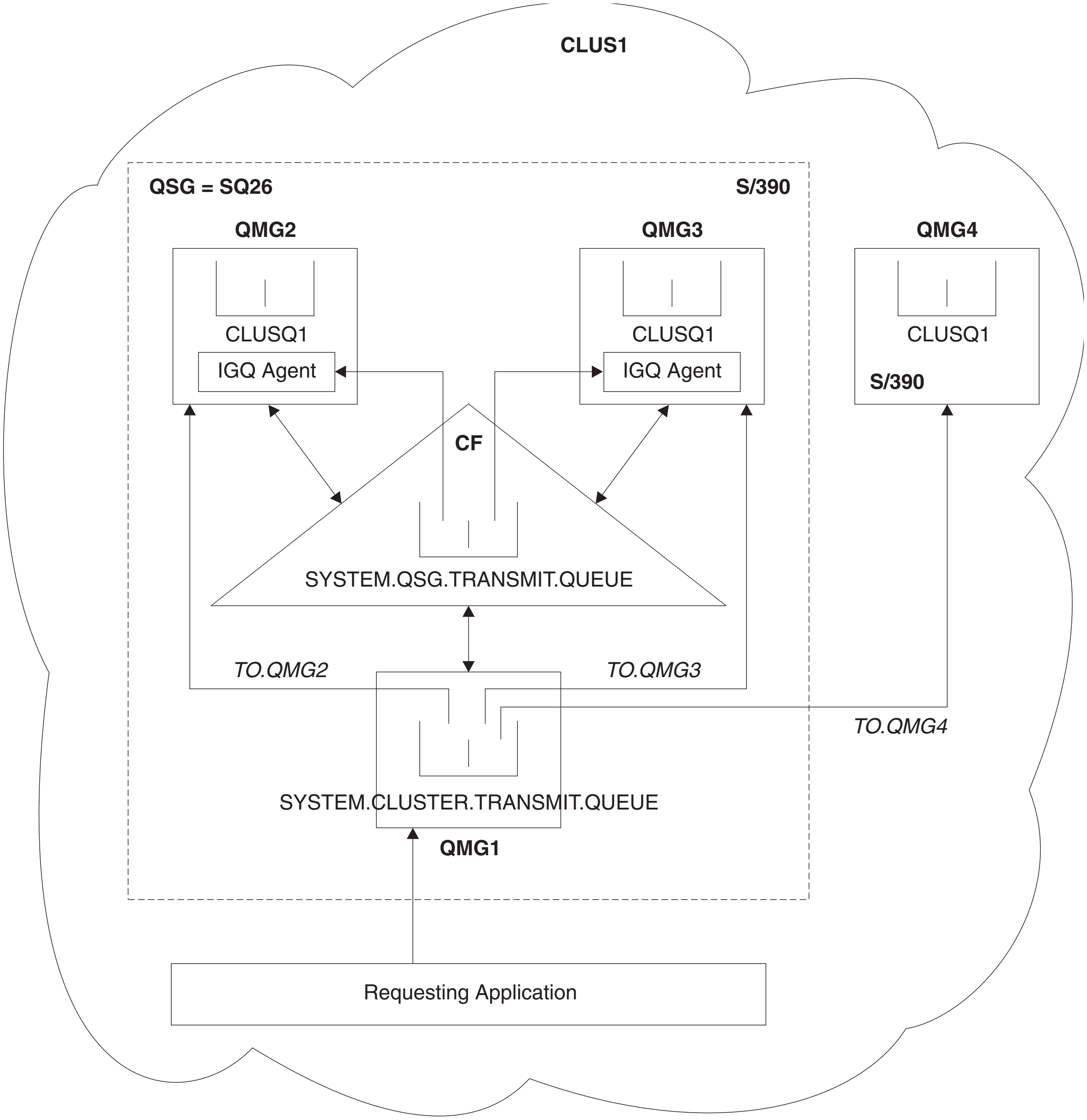


Figure 47. An example of clustering with intra-group queuing

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.
- The shared SYSTEM.QSG.TRANSMIT.QUEUE defined in the CF, which is in a WebSphere 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 236.

Clustering with intra-group queuing is described in “Clustering with intra-group queuing (multiple delivery paths)” on page 238.

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 WebSphere MQ Version 5 Release 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 WebSphere MQ messages (*WebSphere MQ V7.1 Programming Guide*) section for more information about the rules of queue selection).

Related concepts:

"Undelivered/unprocessed messages"

"Report messages"

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 244.
- 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:

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, when intra-group queuing is used.

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 244.
- non-persistent, the IGQ agent discards the exception report and continues processing the next message on the SYSTEM.QSG.TRANSMIT.QUEUE.

Security:

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:

This section describes the specific properties of intra-group queuing.

Related concepts:

"Invalidation of object handles (MQRC_OBJECT_CHANGED)"

"Self recovery of the intra-group queuing agent" on page 244

"Retry capability of the intra-group queuing agent" on page 244

"The intra-group queuing agent and Serialization" on page 244

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 new 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:

Constant	**Value**
Short retry count	10
Short retry interval	60 secondss = 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 MQCONNX 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.

Storage management

IBM WebSphere 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 WebSphere MQ utilizes these page sets and buffers.

Related concepts:
"Page sets"
"Storage classes" on page 246
"Buffers and buffer pools" on page 248
Related reference:
"Where to find more information about storage management" on page 249

Page sets:

Use this topic to understand how WebSphere 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 WebSphere 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 183, and for more information about global definitions, see Private and global definitions (*WebSphere MQ V7.1 Administering Guide*).

WebSphere 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.

WebSphere MQ uses page set zero (PSID=00) to store object definitions and other important information relevant to the queue manager. For normal operation of WebSphere 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 WebSphere MQ provides a FORMAT utility for this; see Formatting page sets (FORMAT) (*WebSphere MQ V7.1 Reference*). Page sets must also be defined to the WebSphere MQ subsystem.

WebSphere MQ for z/OS can be configured to expand a page set dynamically if it becomes full. WebSphere 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, WebSphere MQ cannot expand the page sets beyond 64 GB.

You cannot use page sets from one WebSphere MQ queue manager on a different WebSphere 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.

In previous releases, page sets were limited to 4 GB. It is not possible to modify such existing page sets to be larger than 4 GB. Instead, you must create new page sets with extended addressability and extended format attributes.

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:

A *storage class* is a WebSphere 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 that 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 “WebSphere MQ and IMS” on page 300).

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(MAXI) PSID(05)
DEFINE QLOCAL(QE1) STGCLASS(ARC1) ...
DEFINE QLOCAL(QE2) STGCLASS(ARC1) ...
DEFINE QLOCAL(QE3) STGCLASS(MAXI) ...
DEFINE QLOCAL(QE4) STGCLASS(ARC2) ...
DEFINE QLOCAL(QE5) STGCLASS(ARC2) ...
```

In Figure 48, both storage classes ARC1 and MAXI 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.

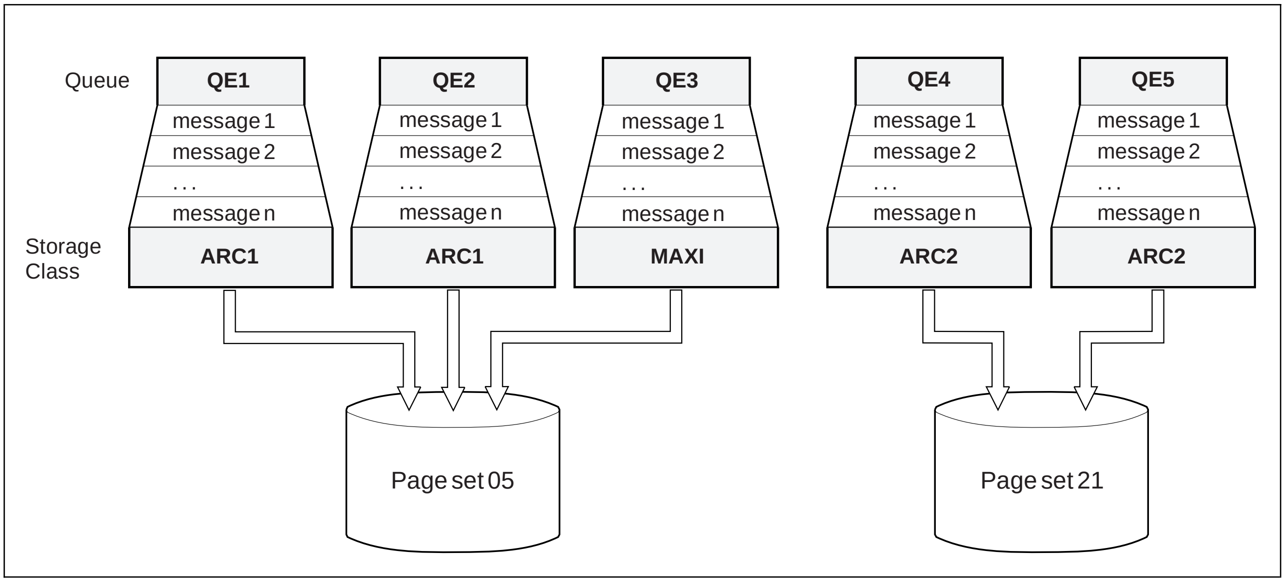


Figure 48. Mapping queues to page sets through storage classes

If you define a queue without specifying a storage class, WebSphere 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:

WebSphere 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, WebSphere 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 WebSphere MQ.

The buffers are organized into *buffer pools*. You can define up to 16 buffer pools (0 through 15) 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 49, and any data isolation requirements your application might have. Each buffer is 4 KB long. The maximum number of buffers is determined by the amount of storage available in the queue manager address space; do not use more than about 70% of the space for buffers. Typically, the more buffers you have, the more efficient the buffering and the better the performance of WebSphere MQ.

Figure 49 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.

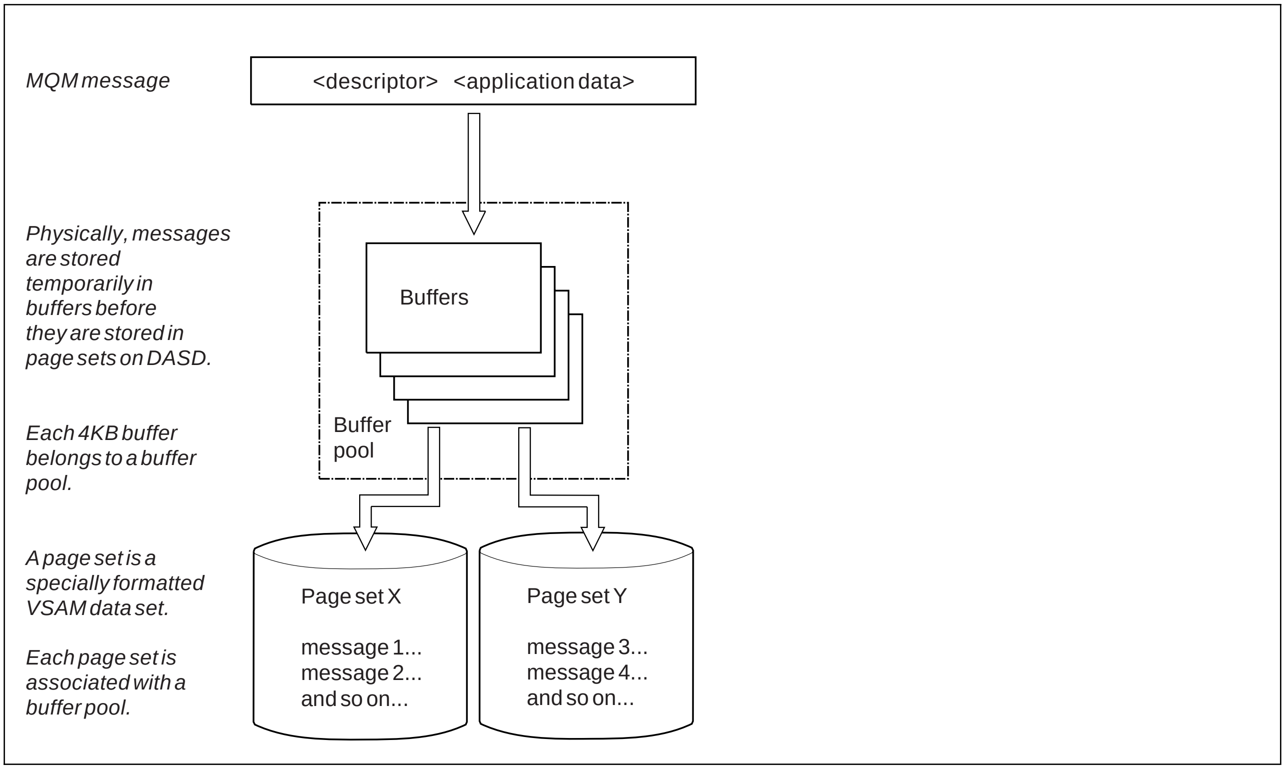

MQM message
<descriptor> <application data>
Physically, messages are stored temporarily in buffers before they are stored in page sets on DASD.
Buffers
Each 4KB buffer belongs to a buffer pool.
Buffer pool
A page set is a specially formatted VSAM data set.
Each page set is associated with a buffer pool.
Page set X
message 1...
message 2...
and so on...
Page set Y
message 3...
message 4...
and so on...

Figure 49. Buffers, buffer pools, and page sets

It is possible for an administrator to dynamically issue commands for modifying buffer pool size, numbers, and their association with page sets at any time (not just at queue manager restart), using the ALTER BUFFPOOL command.

If a buffer pool is too small, WebSphere 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 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. These commands are described in the WebSphere MQ Script (MQSC) Command Reference (*WebSphere MQ V7.1 Reference*) manual.

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:

Use this topic as a reference to find further information about storage management for WebSphere MQ for z/OS.

You can find more information about the topics in this section from the following sources:

Table 19. 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 (*WebSphere MQ V7.1 Installing Guide*)
How large to make your page sets and buffer pools	Plan your page sets and buffer pools (*WebSphere MQ V7.1 Installing Guide*)
Managing page sets	Managing page sets (*WebSphere MQ V7.1 Administering Guide*)
Console messages	"z/OS Messages and Codes" on page 69
MQSC commands	The MQSC commands (*WebSphere MQ V7.1 Reference*)

Logging

WebSphere 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.

WebSphere MQ maintains *logs* of data changes and significant events as they occur. 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 WebSphere MQ collects, see Monitoring and statistics.

For more information about logging, see the following topics:

- "The log files" on page 250
- "How the log is structured" on page 254
- "How the logs are written" on page 254

- “The bootstrap data set” on page 257

Related concepts:

Planning your logging environment (*WebSphere MQ V7.1 Installing Guide*)

Setting logs using the system parameter module (*WebSphere MQ V7.1 Installing Guide*)

Administering z/OS (*WebSphere MQ V7.1 Administering Guide*)

Messages for WebSphere MQ for z/OS (*WebSphere MQ V7.1 Reference*)

MQSC reference (*WebSphere MQ V7.1 Reference*)

The log files:

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

WebSphere MQ records all significant events as they occur in an *active log*. The log contains the information needed to recover:

- Persistent messages
- WebSphere MQ objects, such as queues
- The WebSphere MQ queue manager

The active log comprises a collection of data sets (up to 31) 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, WebSphere 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, WebSphere 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 WebSphere 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 switched off, the active log with new log records wraps, overwriting earlier log records. This means that WebSphere 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 (*WebSphere MQ V7.1 Installing Guide*).

To help prevent problems with unplanned long-running units of work, WebSphere 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 WebSphere 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 (*WebSphere MQ V7.1 Administering Guide*).

Log compression

You can configure WebSphere 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 (*WebSphere MQ V7.1 Administering Guide*) and Accounting and statistics messages (*WebSphere MQ V7.1 Administering Guide*).

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 (*WebSphere MQ V7.1 Reference*) and DISPLAY LOG (*WebSphere MQ V7.1 Reference*)
- The Set Log and Inquire Log functions in the PCF interface; see Set log (*WebSphere MQ V7.1 Reference*) and Inquire log (*WebSphere MQ V7.1 Reference*)
- The CSQ6LOGP macro in the system parameter module; see Using CSQ6LOGP (*WebSphere MQ V7.1 Installing Guide*)

In addition the Log Print utility CSQ1LOGP has support for expanding any compressed log records.

Log data

The log can contain up to 140 million million ($1.4*10^{14}$) 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 field giving a total addressable range of 2^{48} bytes. However, when WebSphere MQ detects that the used range is beyond x'700 000 000 000' (that is, more than $1.2*10^{14}$ bytes of the log have been used), messages CSQI045, CSQI046 and CSQI047 are issued, warning you to reset the log RBA. Do not allow the log RBA to go beyond 2^{47} bytes (x'800 000 000 000'). Once the warning messages about the used log range are being issued, you should plan a queue manager outage during which the log can be reset following the procedure documented in Resetting the queue manager's log (*WebSphere MQ V7.1 Administering Guide*).

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 WebSphere 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 WebSphere MQ queues. All such changes are made within units of recovery.

WebSphere 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, WebSphere 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 (*WebSphere MQ V7.1 Installing Guide*).
- At the end of a successful restart.
- At normal termination.
- Whenever WebSphere MQ switches to the next active log data set in the cycle.

At the time a checkpoint is taken, WebSphere MQ issues the DISPLAY CONN command (described in DISPLAY CONN (*WebSphere MQ V7.1 Reference*)) 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 known to the WebSphere 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.

How the log is structured:

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 logs are written:

Use this topic to understand how WebSphere MQ processes log file records.

WebSphere MQ writes each log record to a DASD data set called the *active log*. When the active log is full, WebSphere MQ copies its contents to a DASD or tape data set called the *archive log*. This process is called *offloading*.

Figure 50 on page 255 illustrates the process of logging. Log records typically go through the following cycle:

1. WebSphere MQ notes changes to data and significant events in recovery log records.
2. WebSphere 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 Controls Intervals (CI). Each log record is identified by a relative byte address in the range zero through 2^{48}-1.
4. The CIs are written to a set of predefined DASD active log data sets, which are used sequentially and recycled.
5. If archiving is active, as each active log data set becomes full, its contents are automatically offloaded to a new archive log data set.

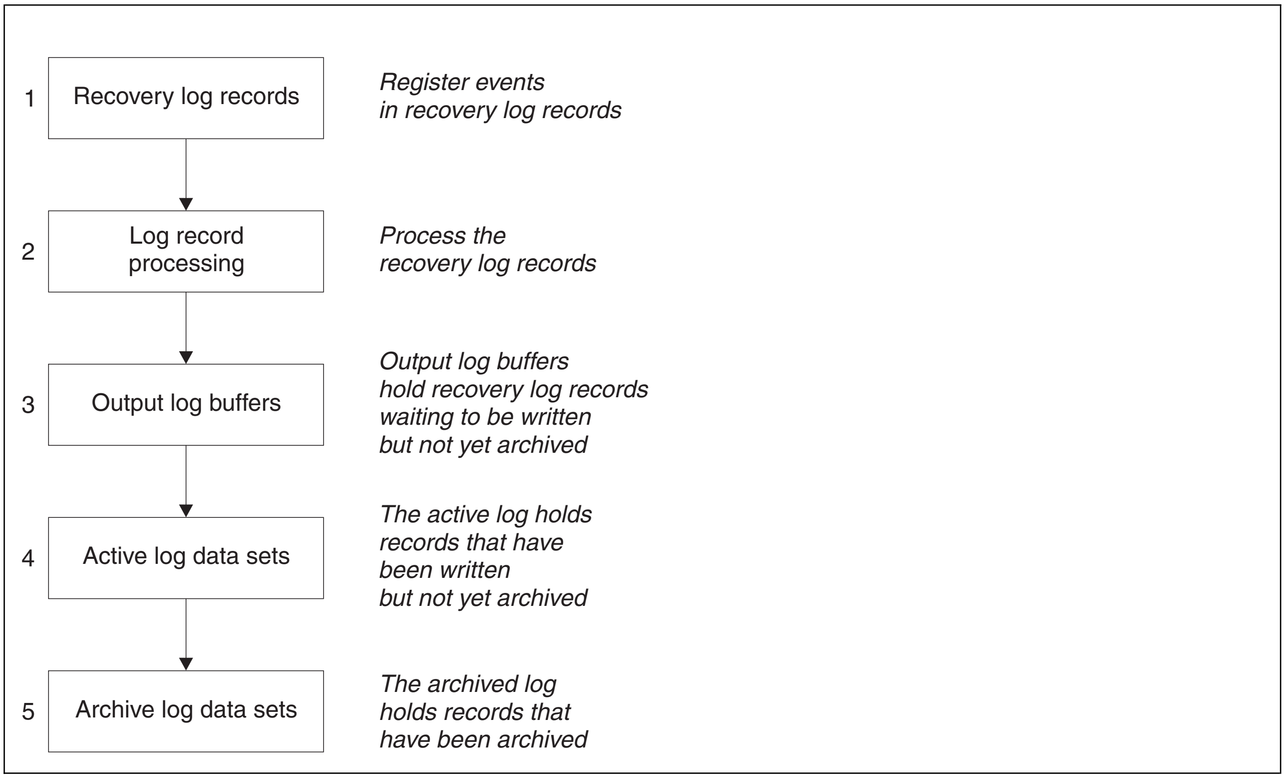


Figure 50. The logging process

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 a WebSphere 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 WebSphere 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 in the WebSphere MQ Script (MQSC) Command Reference (*WebSphere MQ V7.1 Reference*)

Note: To redefine or remove active logs you must terminate and restart the queue manager.

WebSphere MQ and Storage Management Subsystem

WebSphere MQ parameters enable you to specify Storage Management Subsystem (MVS/DFP SMS) storage classes when allocating WebSphere MQ archive log data sets dynamically. WebSphere 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 archive log is written"

When the archive log is written:

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 51.

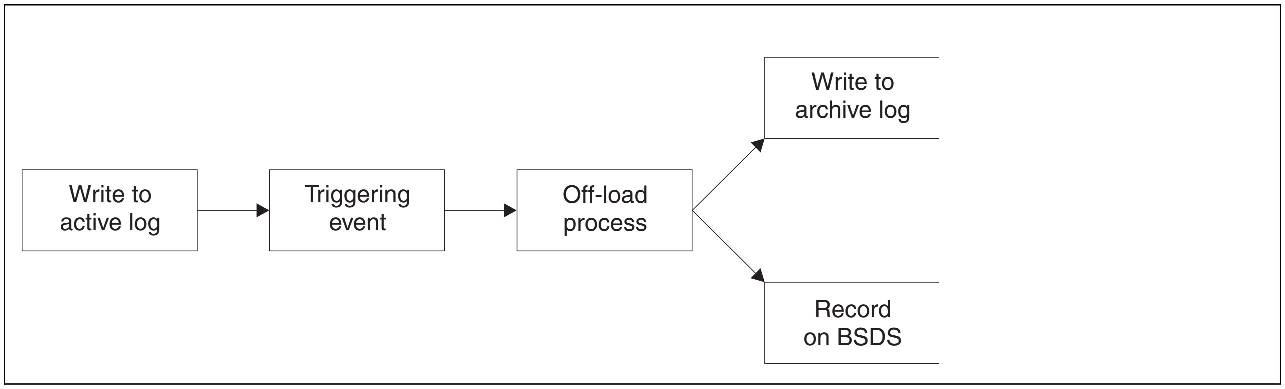


Figure 51. The offloading process

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, WebSphere 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, WebSphere 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, WebSphere 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 (*WebSphere MQ V7.1 Installing Guide*)) 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 WebSphere MQ performance unless the operator delays the response for so long that WebSphere 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 WebSphere 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 WebSphere MQ and the offloading process. You can use these messages to find the RBA ranges in the various log data sets. For an explanation of offloading messages, see “z/OS Messages and Codes” on page 69.

The bootstrap data set:

The bootstrap data set is required by WebSphere 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 WebSphere MQ. It contains the following:

- An inventory of all active and archived log data sets known to WebSphere MQ. WebSphere 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

  WebSphere 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 WebSphere 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. WebSphere MQ **must** have a BSDS. To minimize the likelihood of problems during a restart, you can configure WebSphere 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 WebSphere MQ is customized and you can manage the inventory using the change log inventory utility (CSQJU003). For more information about this utility, see Administering z/OS (*WebSphere MQ V7.1 Administering Guide*). It is referenced by a DD statement in the queue manager startup procedure.

Normally, WebSphere 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 z/OS (*WebSphere MQ V7.1 Administering Guide*).

The active logs are first registered in the BSDS when WebSphere 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 (WebSphere 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.

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

WebSphere 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 WebSphere MQ for z/OS, a system parameter module controls the logging, archiving, tracing, and connection environments that WebSphere 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 WebSphere 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 WebSphere 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 (*WebSphere MQ V7.1 Reference*).

For more information about defining , see the following topics:
- "Defining system objects"
- "Tuning your queue manager" on page 264
- "Sample definitions supplied with WebSphere MQ" on page 265

Related concepts:

Customize the sample initialization input data sets (*WebSphere MQ V7.1 Installing Guide*)

Administering z/OS (*WebSphere MQ V7.1 Administering Guide*)

MQSC reference (*WebSphere MQ V7.1 Reference*)

Monitoring WebSphere MQ (*WebSphere MQ V7.1 Administering Guide*)

Related tasks:

Configuring clusters (*WebSphere MQ V7.1 Installing Guide*)

Defining system objects:

IBM WebSphere 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 WebSphere 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
- Pending data queue
- "Channel authentication queue" on page 263

Publish/subscribe objects

There are several system objects that you need to define before you can use publish/subscribe applications with IBM WebSphere MQ for z/OS. Sample definitions are supplied with IBM WebSphere 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.

For further information about publish/subscribe, topics, subscriptions, and subscription points refer to Publish/Subscribe Users Guide

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 WebSphere 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 WebSphere MQ operations and control panels to issue commands to a IBM WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere MQ bridges.

Do **not** define the dead-letter queue as a shared queue.

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 (*WebSphere MQ V7.1 Administering Guide*).

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.

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.

Channel authentication queue

For internal use of channel authentication the SYSTEM.CHLAUTH.DATA.QUEUE queue is required. Sample definitions are supplied with IBM WebSphere MQ to help you define these objects. This sample is described in CSQ4INSA, which also defines some default rules.

Tuning your queue manager:

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. WebSphere 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.

In Publish/Subscribe, the MAXUMSGS limit should be calculated based on the number of messages that can be delivered to subscribers in a single unit of work; see Publishers and publications (*WebSphere MQ V7.1 Installing Guide*) for more information.

`MAXUMSGS` has no effect on IBM WebSphere MQ Telemetry. IBM WebSphere 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 WebSphere MQ:

Use this topic as a reference for the sample JCL, and code supplied with WebSphere MQ for z/OS.

The following sample definitions are supplied with WebSphere 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 (*WebSphere MQ V7.1 Administering Guide*)).

Table 20. WebSphere MQ sample definitions for system objects

Initialization input data set	**Sample name**
CSQINP1	CSQ4INP1 CSQ4INPR
CSQINP2	CSQ4INSA CSQ4INYS[1] CSQ4INSX CSQ4INSG CSQ4INSR CSQ4INSS CSQ4INSJ CSQ4INYG CSQ4INYR CSQ4INYC CSQ4INYD
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 isn't active):

- The WebSphere 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.

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.

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.

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 WebSphere 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 WebSphere 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 WebSphere 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`, has been moved from CSQ4INSG to CSQ4INST.

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 WebSphere 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.

Recovery and restart

Use the links in this topic to find out about the features of WebSphere MQ for z/OS for restart and recovery.

WebSphere MQ for z/OS has robust features for restart and recovery. See the following links for information about how a queue manager recovers after it has stopped, and what happens when it is restarted:

- "How changes are made to data"
- "How consistency is maintained" on page 271
- "What happens during termination" on page 273
- "What happens during restart and recovery" on page 274
- "How in-doubt units of recovery are resolved" on page 276
- "Shared queue recovery" on page 279

Related concepts:

Planning for backup and recovery (*WebSphere MQ V7.1 Installing Guide*)

WebSphere MQ for z/OS recovery actions (*WebSphere MQ V7.1 Administering Guide*)

Administering z/OS (*WebSphere MQ V7.1 Administering Guide*)

Messages for WebSphere MQ for z/OS (*WebSphere MQ V7.1 Reference*)

MQSC reference (*WebSphere MQ V7.1 Reference*)

How changes are made to data:

WebSphere MQ 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 WebSphere 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.

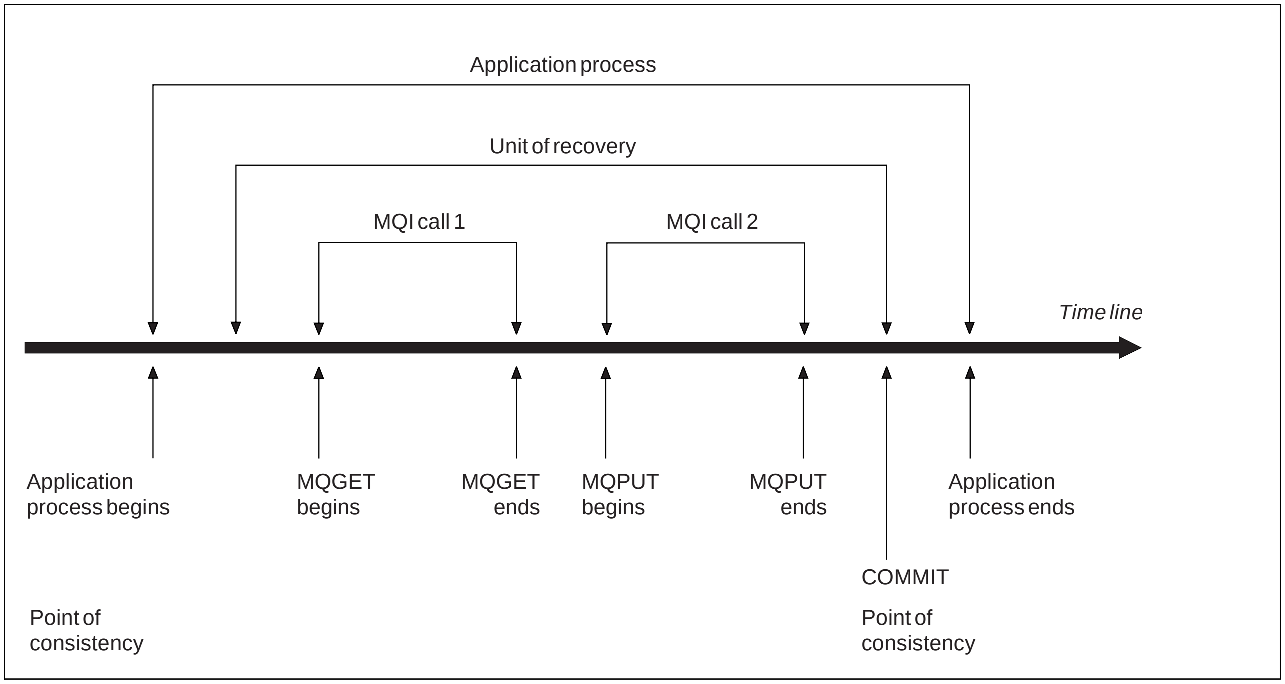


Figure 52. A unit of recovery within an application program. Typically, the unit of recovery consists of more than one MQI call. More than one unit of recovery can occur within an application program.

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 52 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 WebSphere 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, WebSphere MQ removes any changes to data, returning the data to its state at the start of the unit of recovery; that is, WebSphere MQ backs out the work. The events are shown in Figure 53 on page 271.

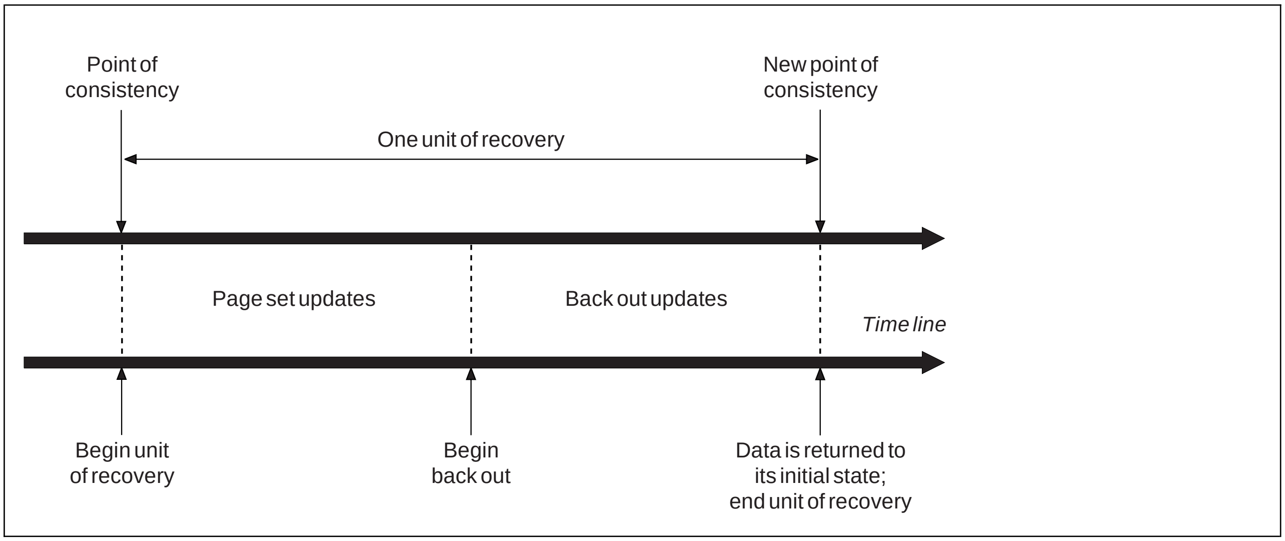


Figure 53. A unit of recovery showing back out

How consistency is maintained:

Data in WebSphere MQ 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 WebSphere MQ, and WebSphere MQ is always the participant. In the batch or TSO environment, WebSphere MQ can participate in two-phase commit protocols coordinated by z/OS RRS.

During a *single-phase commit* (for example under TSO or batch), WebSphere 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 WebSphere 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 (WebSphere MQ).

  This protocol is optimized for logging and message flows.
- Bypass of syncpoint - for transactions that involve WebSphere 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 WebSphere 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 54 illustrates the two-phase commit process. Events in the CICS or IMS coordinator are shown on the upper line, events in WebSphere MQ on the lower line.

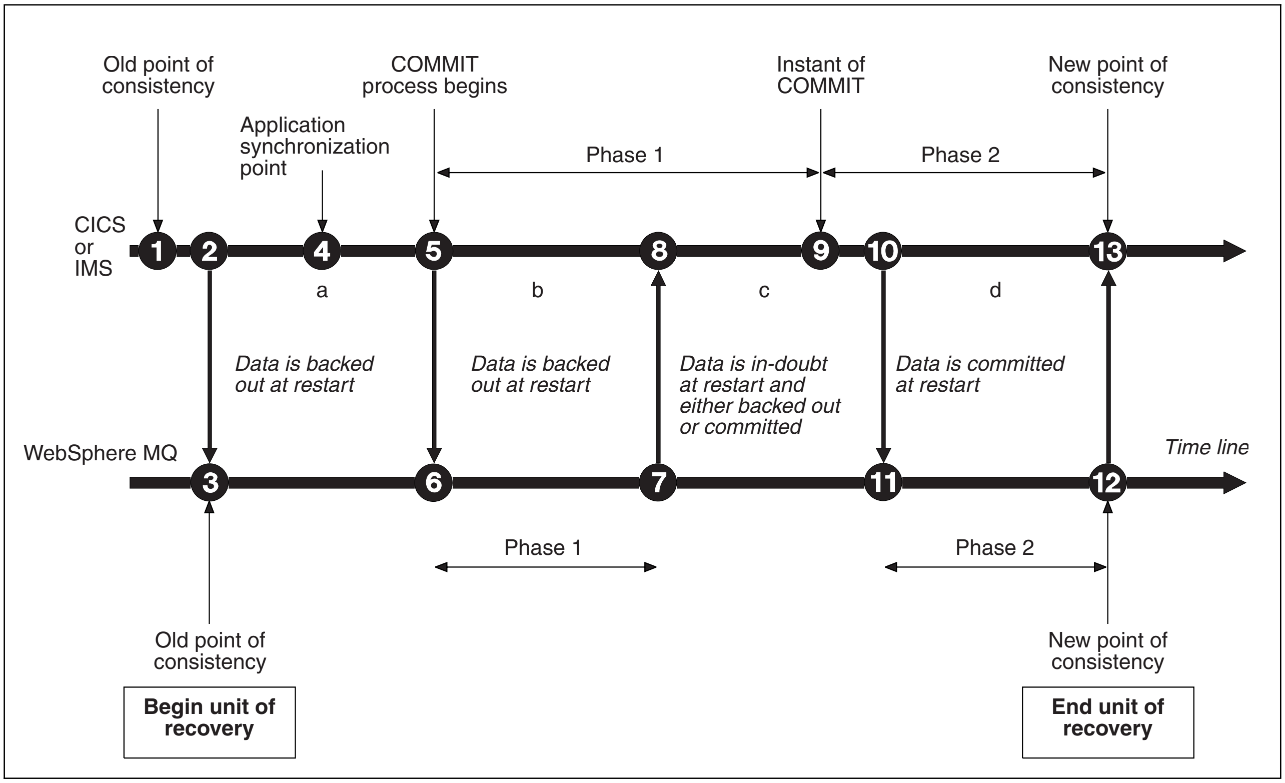


Figure 54. The two-phase commit process

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 WebSphere MQ to update a queue by adding a message.
3. This starts a unit of recovery in WebSphere 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 WebSphere MQ.
7. WebSphere 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 WebSphere MQ to begin its phase 2.
11. WebSphere MQ logs the start of phase 2.
12. Phase 2 is successfully completed, and this is now a new point of consistency for WebSphere MQ. WebSphere 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, WebSphere 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 54 on page 272 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, WebSphere 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, WebSphere MQ must back out its changes; if it happened after, WebSphere MQ must make its changes and commit them. At restart, WebSphere 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, WebSphere MQ continues to back out the changes.

What happens during termination:

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, WebSphere 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, WebSphere MQ stops all activity in an orderly way. You can stop WebSphere MQ using either quiesce, force, or restart mode. The effects are given in Table 21.

Table 21. 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 WebSphere 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. WebSphere MQ ceases to accept commands.
3. WebSphere MQ ensures that any outstanding updates to the page sets are completed.
4. The DISPLAY USAGE command is issued internally by WebSphere 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.

WebSphere MQ resolves any data inconsistencies arising from abnormal termination during restart and recovery.

What happens during restart and recovery:

WebSphere 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 WebSphere MQ checkpoint in the log.

Introduction to restart and recovery

After WebSphere 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 WebSphere 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 276).

If dual BSDSs are in use, WebSphere MQ checks the consistency of the time stamps in the BSDS:

- If both copies of the BSDS are current, WebSphere MQ tests whether the two time stamps are equal. If they are not, WebSphere 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. WebSphere 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, WebSphere 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. WebSphere 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 WebSphere 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, WebSphere MQ restarts without waiting for indexes to rebuild.

How in-doubt units of recovery are resolved:

If WebSphere MQ loses its connection to another resource manager, it typically attempts to recover all inconsistent objects at restart.

If WebSphere 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 WebSphere MQ process to resolve in-doubt units of recovery. When this happens, WebSphere 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 z/OS Messages and Codes (*WebSphere MQ V7.1 Reference*) 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 WebSphere MQ is being reconnected.

One of the functions of the CICS adapter is to keep data synchronized between CICS and WebSphere MQ. If a queue manager abends while connected to CICS, it is possible for CICS to commit or back out work without WebSphere MQ being aware of it. When the queue manager restarts, that work is termed *in doubt*.

WebSphere MQ cannot resolve these in-doubt units of recovery (that is, commit or back out the changes made to WebSphere 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 WebSphere 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, WebSphere 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 z/OS (*WebSphere MQ V7.1 Administering Guide*).

How in-doubt units of recovery are resolved from IMS

Resolving in-doubt units of recovery in IMSdoes 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 WebSphere MQ connects.

During queue manager restart, WebSphere 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 WebSphere MQ, IMS indicates to WebSphere MQ whether to commit or back out units of work marked in WebSphere MQ as in doubt.

When in-doubt units are resolved:

1. If WebSphere MQ recognizes that it has marked an entry for commit and IMS has marked it to be backed out, WebSphere MQ issues message CSQQ010E. WebSphere MQ issues this message for all inconsistencies of this type between WebSphere MQ and IMS.
2. If WebSphere MQ has any remaining in-doubt units, the adapter issues message CSQQ008I.

For all resolved units, WebSphere MQ updates queues as necessary and releases the corresponding locks.

WebSphere 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 z/OS (*WebSphere MQ V7.1 Administering Guide*).

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 WebSphere 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 WebSphere MQ and other RRS-participating resource managers. If a failure occurs when WebSphere 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 WebSphere 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. WebSphere MQ cannot resolve these in-doubt units of recovery (that is, commit or back out the changes made to WebSphere MQ resources) until the connection to RRS is reestablished.

Under some circumstances, RRS cannot resolve in-doubt units of recovery. When this happens, WebSphere 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 z/OS Messages and Codes (*WebSphere MQ V7.1 Reference*) manual.

For all resolved units of recovery, WebSphere 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 z/OS (*WebSphere MQ V7.1 Administering Guide*).

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 QSG 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:

Use this topic to understand WebSphere MQ recovery and resilience of various components in the queue-sharing group environment.

- "Transactional recovery"
- "Peer recovery"
- "Shared queue definitions" on page 280
- "Logging" on page 280
- "Coupling facility and structure failures" on page 280
- "Structure failure scenarios" on page 281
- "Resilience to coupling facility connectivity failures" on page 282
- "Managing Resilience to coupling facility connectivity failures" on page 283
- "Operational behavior " on page 285

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 WebSphere 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 WebSphere MQ objects. You can also use the WebSphere MQ CSQUTIL utility to create MQSC commands for replay into a queue manager to redefine WebSphere 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 WebSphere 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 WebSphere 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. Version 6 and later queue managers might be able to do this without terminating (depending on the type of failure), otherwise the queue manager rebuilds its administration structure entries when it is started.

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 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 WebSphere MQ.
- The structure type (application or administration)
- The queue manager level (Version 6.0 , Version 7.0, and Version 7.1)
- The CFLEVEL of the MQ CFSTRUCT object (2, 3, 4 or 5. This is not the CFLEVEL of the CFCC microcode)

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 and the queue manager is running at Version 6.0 and later, the structure is reallocated and rebuilt automatically without the queue manager terminating. Because a structure failure has occurred the structure is not allocated in the CF, it 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 only the entries for itself into the structure. This processing is carried out by the queue manager and is not part of XES rebuild processing.

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 until all the queue managers in the queue-sharing group have rebuilt their administration structure entries. Certain actions on the shared queue are suspended until the queue manager has reconnected to the administration structure and finished rebuilding the entries in the structure. 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.

When the administration structure entries for the queue manager have been rebuilt, the suspended actions are resumed.

You cannot back up or recover an application structure until all the queue managers in the queue-sharing group have rebuilt their administration structure entries. 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 are rebuilt by another queue manager in the queue-sharing group if that queue manager is running at Version 7.0.1 or later. If there is no queue manager running at Version 7.0.1 or later in the queue-sharing group, restart the queue manager so that it can complete the rebuild of its part of the structure.

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 RECOVER CFSTRUCT command has successfully rebuilt the failed structure, at which point the application can open the queues again.

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?

WebSphere 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?

WebSphere 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 WebSphere 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 recreate 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 *MVS*™ *Setting Up a Sysplex* manual 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 WebSphere 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.

   WebSphere 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 WebSphere MQ needs to reallocate the structure.

   Refer to *MVS*™ *Setting Up a Sysplex* manual for more information about structure preference lists.

   Note: This step is only required if the structures are to be automatically rebuilt into another coupling facility following loss of connectivity.
3. Migrate all queue managers in the queue-sharing-group to WebSphere MQ V7.1 and start the queue managers with OPMODE set to enable new V7.1 function. This is required before loss of connectivity to the administration structure can be tolerated by any member of the queue-sharing-group.
4. 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.
5. 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.
6. 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 WebSphere MQ v7.1 or higher. If there are queue managers at a level lower then WebSphere MQ v7.1 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. During this time, queue managers that did not lose connectivity are still able to access shared queues defined on the affected application structure, although operations may experience some delay 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 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 WebSphere MQ v7.1 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 WebSphere 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 *MVS™ System Commands*.

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 on z/OS

Use this topic to understand the importance of security for WebSphere MQ, and the implications of not having adequate security settings on your system.

Why you must protect WebSphere MQ resources

Because IBM WebSphere MQ handles the transfer of information that is potentially valuable, it needs the safeguard of a security system. This is to ensure that the resources IBM WebSphere MQ owns and manages are protected from unauthorized access that might lead to the loss or disclosure of the information. Ensure that none of the following are accessed or changed by any unauthorized user or process:

- Connections to IBM WebSphere MQ
- IBM WebSphere MQ objects such as queues, processes, and namelists
- IBM WebSphere MQ transmission links
- IBM WebSphere MQ system control commands
- IBM WebSphere MQ messages
- Context information associated with messages

To provide the necessary security, IBM WebSphere 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 WebSphere 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 WebSphere MQ provides channel authentication records, channel exits, the MCAUSER channel attribute, and SSL or TLS.

The decision to allow access to an object is made by the ESM and IBM WebSphere MQ follows that decision. If the ESM cannot make a decision, IBM WebSphere MQ prevents access to the object.

What happens if you do not protect WebSphere 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.

For more information about security, see the following topics:

- "Security controls and options" on page 287
- "Resources you can protect" on page 289

Related concepts:

Security concepts (*WebSphere MQ V7.1 Administering Guide*)

Channel authentication records (*WebSphere MQ V7.1 Administering Guide*)

Authority to work with WebSphere MQ objects on z/OS (*WebSphere MQ V7.1 Administering Guide*)

Setting up security on z/OS (*WebSphere MQ V7.1 Administering Guide*)

Comparing link level security and application level security (*WebSphere MQ V7.1 Administering Guide*)

Cryptographic security protocols: SSL and TLS (*WebSphere MQ V7.1 Administering Guide*)

Messages for WebSphere MQ for z/OS (*WebSphere MQ V7.1 Reference*)

MQSC reference (*WebSphere MQ V7.1 Reference*)

Security controls and options:

You can specify whether security is turned on for the whole WebSphere 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 WebSphere 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 WebSphere MQ checking is done; if you leave it turned on, WebSphere MQ checks your security requirements for other WebSphere 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 (*WebSphere MQ V7.1 Administering Guide*).

Controlling the number of user IDs checked

RESLEVEL is a Security Server profile that controls the number of user IDs checked for WebSphere MQ resource security. Normally, when a user attempts to access a WebSphere 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 WebSphere MQ RACF classes

You can now use mixed case RACF profile support, which allows you to use mixed case resource names and define WebSphere MQ RACF profiles to protect them.

You can choose to either:

- Continue using uppercase only WebSphere MQ RACF Classes as in previous releases, or
- Use the new mixed case WebSphere MQ RACF classes.

Without the use of mixed case RACF profiles, you can still use mixed case resource names in WebSphere MQ for z/OS; however, these resource names can only be protected by generic RACF profiles in the uppercase WebSphere MQ classes. When using mixed case WebSphere MQ RACF profile support you can provide a more granular level of protection by defining WebSphere MQ RACF profiles in the mixed case WebSphere MQ classes.

Resources you can protect:

When a queue manager starts, or when instructed by an operator command, IBM WebSphere MQ 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 WebSphere 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 (*WebSphere MQ V7.1 Administering Guide*). 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 WebSphere MQ resources. When command resource security is active, each time a command involving a resource is issued, IBM WebSphere 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 SSL or 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 (*WebSphere MQ V7.1 Administering Guide*) and Security exit overview (*WebSphere MQ V7.1 Administering Guide*)

Related reference:

"API-resource security"

API-resource security:

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 a WebSphere 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 WebSphere 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 (*WebSphere MQ V7.1 Programming Guide*). For descriptions of the message descriptor fields relating to context, see MQMD – Message descriptor (*WebSphere MQ V7.1 Reference*).

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

WebSphere MQ for z/OS has many features for high availability. This topic describes some of the considerations for availability.

Several features of WebSphere 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
- WebSphere 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. WebSphere 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 WebSphere 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 204.

To further enhance the availability of messages in a queue-sharing group, WebSphere 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 279.

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 WebSphere 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)" on page 294.

Shared channels

In the queue-sharing group environment, WebSphere 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). WebSphere 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 225.

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.

WebSphere MQ network availability

WebSphere MQ messages are carried from queue manager to queue manager in a WebSphere 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 a WebSphere 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 KAINT 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 WebSphere 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 WebSphere MQ is described in Using ARM in a WebSphere MQ network (*WebSphere MQ V7.1 Administering Guide*).

Using the z/OS Extended Recovery Facility (XRF)

You can use WebSphere MQ in an extended recovery facility (XRF) environment. All WebSphere 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.

WebSphere 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 WebSphere 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 QSG 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 QSG appears to the transactional application as a single entity providing all the WebSphere 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 22. Where to find more information about availability

Topic	Where to look
Queue-sharing groups	“Shared queues and queue-sharing groups” on page 183
System parameters	Configuring system parameters
Using the Automatic Restart Manager Utility programs	Using ARM in a WebSphere MQ network (*WebSphere MQ V7.1 Administering Guide*)
Console messages	“z/OS Messages and Codes” on page 69
MQSC commands	MQSC commands (*WebSphere MQ V7.1 Reference*)

Monitoring and statistics

WebSphere MQ for z/OS has a set of facilities for monitoring the queue manager, and gathering statistics.

WebSphere MQ supplies facilities for monitoring the system and collecting statistics. For further information about these facilities, see the following sections:
- “Online monitoring”
- “WebSphere MQ trace” on page 296
- “Events” on page 296

Online monitoring

WebSphere MQ includes the following commands for monitoring the status of WebSphere 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 (*WebSphere MQ V7.1 Reference*).

WebSphere MQ trace

WebSphere 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.
- WebSphere MQ can also gather information about each task using WebSphere MQ. This data is gathered as a thread-level accounting record. For each thread, WebSphere 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

WebSphere 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 WebSphere 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 concepts:

"Monitoring" on page 66

Using IBM WebSphere MQ trace (*WebSphere MQ V7.1 Administering Guide*)

Using IBM WebSphere MQ events (*WebSphere MQ V7.1 Administering Guide*)

Unit of recovery disposition

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 QSG 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 QSG 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"

"Application support" on page 298

Enabling GROUP units of recovery:

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 GROUPUR queue manager attribute. For more information about this concept, see "Unit of recovery disposition" on page 296 before reading the rest of this topic.

When the GROUPUR 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 QSG 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 QSG 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."

When you attempt to enable the GROUPUR 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 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 GROUPUR attribute remains disabled and a message code is returned. Use the message code to identify, in "Messages and codes" on page 65, which specific check failed.

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:

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 WebSphere MQ for z/OS is a resource manager but not the transaction coordinator. Currently supported transactional applications are:

- WebSphere MQ extended transactional client applications
- WebSphere 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:

"WebSphere MQ extended transactional client applications"

"CICS applications" on page 299

WebSphere MQ extended transactional client applications:

Use this page to determine how WebSphere MQ extended transactional client applications can use the GROUP unit of recovery disposition.

An example of a WebSphere MQ extended transactional client application is one that uses JMS and runs in WebSphere Application Server, connecting to WebSphere MQ over TCP/IP, rather than local bindings. These client applications connect to WebSphere 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 (*WebSphere MQ V7.1 Installing Guide*) and Additional error processing for xa_open (*WebSphere MQ V7.1 Installing Guide*). 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 QSG 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 Communications Server and Dynamic virtual addressing for more details. If you want to enable GROUP units of recovery on a subset of the queue managers in your QSG ensure that your client applications cannot be routed to those on which it is not enabled, which includes any queue manager at a version earlier than V7.0.1.

To use the group unit of recovery disposition feature in a JMS client application you must use the WebSphere MQ V7.0.1 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 WebSphere MQ and other z/OS products

Use this topic to understand how IBM WebSphere MQ can work with other z/OS products.

Related concepts:

"WebSphere MQ and CICS"

"WebSphere MQ for z/OS and WebSphere Application Server" on page 306

Related reference:

"WebSphere MQ and IMS" on page 300

"WebSphere MQ and the z/OS Batch, TSO, and RRS adapters" on page 304

WebSphere MQ and CICS

You can use WebSphere MQ with CICS, by configuring the CICS adapter and CICS bridge.

The CICS adapter allows you to connect your queue manager to CICS, and enables CICS applications to use the MQI.

The optional additional WebSphere MQ CICS bridge enables applications to run a CICS program or transaction that does not use the MQI. This means that you can use your legacy applications with WebSphere MQ, without the need to rewrite them.

The minimum level of CICS required is specified in WebSphere MQ prerequisites for z/OS.

For more information about configuring the WebSphere MQ CICS adapter, and the WebSphere MQ CICS bridge components, see the Configuring connections to MQ section of the CICS documentation.

Related concepts:

Using WebSphere MQ with CICS (*WebSphere MQ V7.1 Installing Guide*)

WebSphere MQ and IMS

Use this topic to understand how WebSphere 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 WebSphere 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 WebSphere MQ, without the need to rewrite them.

For more information about these components, see the following topics:

- "The IMS adapter"
- "The IMS bridge" on page 302

Related concepts:

Setting up the IMS adapter (*WebSphere MQ V7.1 Installing Guide*)

Setting up the IMS bridge (*WebSphere MQ V7.1 Installing Guide*)

Operating the IMS adapter (*WebSphere MQ V7.1 Administering Guide*)

IMS and IMS Bridge applications on WebSphere MQ for z/OS (*WebSphere MQ V7.1 Programming Guide*)

z/OS Messages and Codes

Related reference:

MQIIH – IMS information header (*WebSphere MQ V7.1 Reference*)

The IMS adapter:

The IMS adapter is an interface between IMS application programs and a WebSphere MQ subsystem.

The WebSphere 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 a WebSphere MQ subsystem. It makes it possible for IMS application programs to use the MQI.

The IMS adapter receives and interprets requests for access to WebSphere MQ using the External Subsystem Attach Facility (ESAF) provided by IMS. This facility is described in the *IMS Customization Guide*. Usually, IMS connects to WebSphere MQ automatically without operator intervention.

The IMS adapter provides access to WebSphere 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 WebSphere MQ.

The adapter supports a two-phase commit protocol for changes made to resources owned by WebSphere 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 WebSphere 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 WebSphere 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 WebSphere MQ. However, the IMS terminal operator has no control over the WebSphere MQ address space. For example, the operator cannot shut down WebSphere MQ from an IMS address space.

Restrictions

The following WebSphere 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 a WebSphere MQ-supplied IMS application that starts an IMS transaction when a WebSphere 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 WebSphere 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 IMS bridge:

The WebSphere MQ-IMS bridge is the component of WebSphere MQ for z/OS that allows direct access from WebSphere MQ applications to applications on your IMS system.

The WebSphere 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 WebSphere 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 WebSphere 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. WebSphere 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 WebSphere MQ message.

The IMS bridge is illustrated in Figure 55 on page 303.

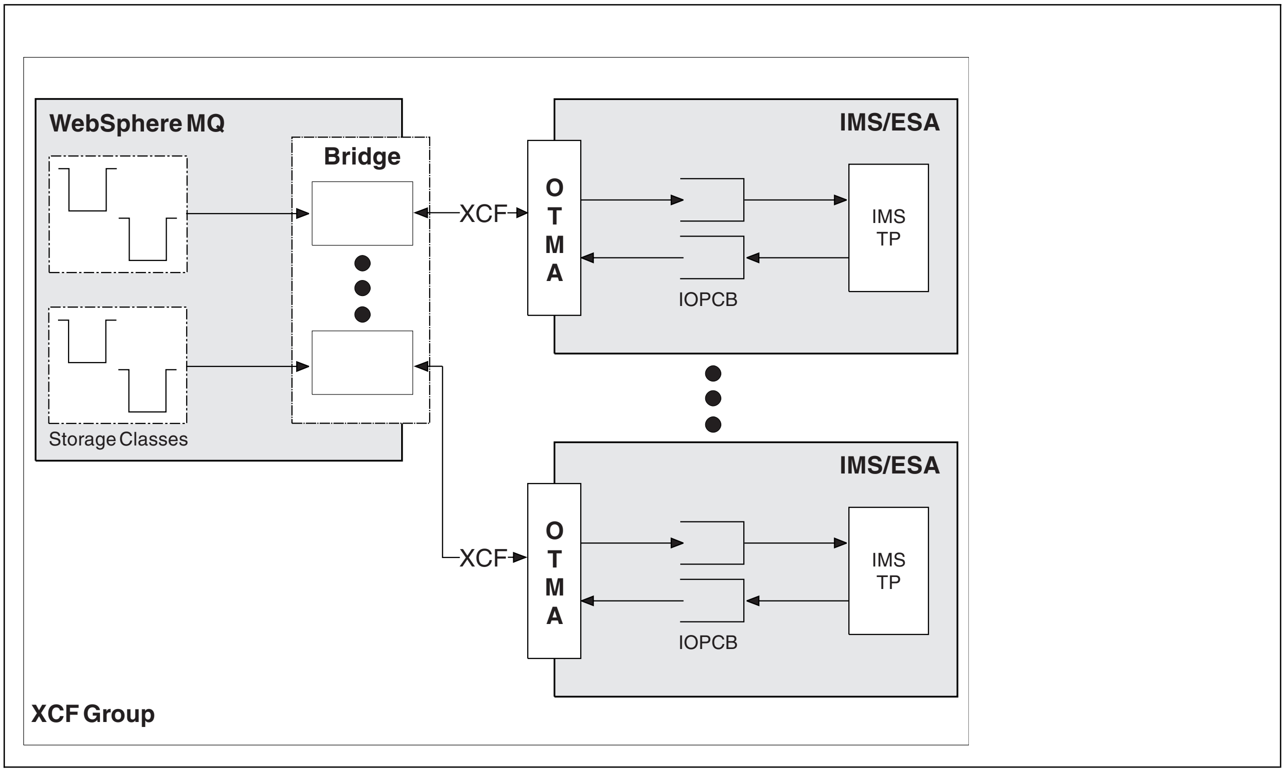


Figure 55. The WebSphere MQ-IMS bridge

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 WebSphere MQ-IMS Bridge in WebSphere 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 WebSphere MQ that a flood warning has occurred. In response WebSphere 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 WebSphere 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 WebSphere MQ in response to a change in the status of IMS partners.

WebSphere 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 (*WebSphere MQ V7.1 Installing Guide*).

If IMS v10 partners are being used you should ensure that PTF UK45082 has been applied.

Submitting IMS transactions from WebSphere MQ

To submit an IMS transaction that uses the bridge, applications put messages on a WebSphere MQ queue as usual. The messages contain IMS transaction data; they can have an IMS header (the MQIIH structure) or allow the WebSphere MQ-IMS bridge to make assumptions about the data in the message.

WebSphere MQ then puts the message to an IMS queue (it is queued in WebSphere MQ first to enable the use of syncpoints to assure data integrity). The storage class of the WebSphere MQ queue determines whether the queue is an *OTMA queue* (that is, a queue used to transmit messages to the WebSphere 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 WebSphere MQ for z/OS.

Data returned from the IMS system is written directly to the WebSphere 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 concepts:

Customizing the IMS bridge (*WebSphere MQ V7.1 Installing Guide*)

IMS and IMS Bridge applications on WebSphere MQ for z/OS (*WebSphere MQ V7.1 Programming Guide*)

Related reference:

"WebSphere MQ and IMS" on page 300

WebSphere MQ and the z/OS Batch, TSO, and RRS adapters

Use this topic to understand how WebSphere 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 WebSphere MQ. They enable z/OS application programs to use the MQI.

The adapters provide access to WebSphere 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 WebSphere MQ are at the task level. The adapters provide a connection thread from an application task control block (TCB) to WebSphere MQ.

The Batch/TSO adapter supports a single-phase commit protocol for changes made to resources owned by WebSphere MQ. It does not support multi-phase commit protocols. The RRS adapter enables WebSphere 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 WebSphere 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 WebSphere MQ (for example, a signal or a wait).

The Batch/TSO adapter

The WebSphere MQ Batch/TSO adapter provides WebSphere 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 WebSphere MQ Batch/TSO RRS adapter (the RRS adapter) provides WebSphere MQ support for z/OS Batch and TSO applications that want to use these services. The RRS adapter enables WebSphere 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**; WebSphere 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 (*WebSphere MQ V7.1 Programming Guide*).

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 23. Where to find more information about using z/OS Batch with WebSphere MQ

Topic	Where to look
Setting up the Batch adapters	Task 19: Set up Batch, TSO, and RRS adapters (*WebSphere MQ V7.1 Installing Guide*)
RRS callable resource recovery services	*MVS Programming: Callable Services for High Level Languages*

WebSphere MQ for z/OS and WebSphere Application Server

Use this topic to understand the use of WebSphere 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 may be provided by a WebSphere MQ for z/OS queue manager.

A benefit of using a WebSphere MQ for z/OS queue manager to provide the messaging is that connecting JMS applications can participate fully in the functionality of a WebSphere 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. If you choose client transport, you must install the *Client Attachment* feature of WebSphere MQ for z/OS.

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 *WebSphere MQ Using Java*.

Using WebSphere MQ functions from JMS applications

By default, JMS messages held on WebSphere MQ queues use an MQRFH2 header to hold some of the JMS message header information. Many legacy WebSphere MQ applications cannot process messages with these headers, and require their own characteristic headers, for example the MQCIH for CICS Bridge, or MQWIH for WebSphere MQ Workflow applications. For more details about these special considerations, see Mapping JMS messages onto WebSphere MQ messages.

Glossary

This glossary includes terms and definitions for IBM WebSphere MQ.

The following cross-references are used in this glossary:

- See refers you from a term to a preferred synonym, or from an acronym or abbreviation to the defined full form.
- See also refers you to a related or contrasting term.

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.

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.

accountability
The quality of being responsible for one's actions.

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 (ASID)
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 WebSphere MQ Administration Interface (MQAI), a type of data bag that is created for administering WebSphere 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 WebSphere MQ objects, such as queues, processes, and namelists.

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.

advanced telemetry client
See telemetry advanced client.

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.

alert monitor
In WebSphere MQ for z/OS, a component of the CICS adapter that handles unscheduled events occurring as a result of connection requests to WebSphere MQ for z/OS.

alias queue
A WebSphere 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.

alias queue object
A WebSphere 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 WebSphere 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 a WebSphere 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.

alternate user security
On z/OS, the authority checks that are performed when an application requests alternate user authority when opening a WebSphere MQ object.

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 WebSphere 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 invoked 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 invoked when an application issues an MQI call.

application log
In Windows systems, a log that records significant application events.

application queue
A local queue which, when it has triggering set on and when the triggering conditions are met, requires that trigger messages are written.

archive log
A data set on a storage device to which WebSphere 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.

ASID See address space.

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.

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 invoking 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. 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.

2. 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.

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 WebSphere 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 WebSphere MQ on UNIX and Linux systems and WebSphere 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 2GB. The bar separates storage below the 2-gigabyte address from storage above the 2-gigabyte 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 WebSphere MQ for z/OS, and a wrap-around inventory of all recent WebSphere MQ for z/OS activity. The BSDS is required to restart the WebSphere 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 in response to a certificate signing request. The certificate authority verifies the identity of the individuals who are granted the unique certificate. See also Secure Sockets Layer.

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 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.

certificate store
The Windows name for a key repository.

certificate signing request (CSR)
A request that contains the public key and subject distinguished name of a utility or organization. Sent to the CA so that the CA issues a digital signature to that utility.

CF See coupling facility.

CFSTRUCT
A WebSphere MQ object used to describe the queue manager's use of a Coupling Facility list structure

channel
A WebSphere 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.

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 WebSphere MQ distributed queuing that monitors the initiation queue to see when triggering criteria have been met and then starts the sender channel.

channel listener
A component of WebSphere 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 has been encrypted. Ciphertext is unreadable until it has been converted into plaintext (decrypted) with a key. See also cleartext.

circular logging
In WebSphere MQ on UNIX and Linux systems and WebSphere MQ for Windows, the process of keeping all restart data in a ring of log files. See also linear 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. They may be encoded for the purposes of compression, but 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 WebSphere MQ MQI client, WebSphere MQ Java client, WebSphere MQ fully-managed .NET 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.

client-connection channel type
The type of MQI channel definition associated with a WebSphere MQ client. See also server-connection channel type.

CLUSRCVR
See cluster-receiver channel.

CLUSSDR
See cluster-sender channel.

cluster
In WebSphere MQ, a group of two or more queue managers on one or more computers, providing automatic interconnection, and allowing queues 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 (CLUSSDR)
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 WebSphere 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 WebSphere 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 been executed successfully.

Command Language (CL)

In WebSphere MQ for iSeries, a language that can be used to issue commands, either at the command line or by writing a CL program.

command prefix

1. A 1- to 8-character command identifier. The command prefix distinguishes the command as belonging to an application or subsystem rather than to z/OS.

2. In WebSphere MQ for z/OS, a character string that identifies the queue manager to which WebSphere MQ for z/OS commands are directed, and from which WebSphere MQ for z/OS operator messages are received.

command server

The WebSphere 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 WebSphere 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 WebSphere MQ on UNIX and Linux systems and WebSphere MQ for Windows, a command that can be entered interactively from the operating system command line. Such a command requires only that the WebSphere 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.

CR (certificate request)
Synonym for certificate signing request.

CRL See certificate revocation list.

cross-system coupling facility (XCF)
A component that provides functions to support cooperation between authorized programs running within a sysplex.

cryptography
Protecting information by transforming it (encrypting it) into an unreadable format, called ciphertext. Only those 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 WebSphere MQ interface to which customer- or vendor-written programs that convert application data between different machine encodings and CCSIDs must conform. A part of the WebSphere MQ Framework.

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 item contained within a data bag. This can be an integer item or a character-string item, and 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 WebSphere 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 WebSphere MQ for z/OS before it has started.

derivation
In object-oriented programming, the refinement or extension of one class from another.

destination

1. In JMS, an object that specifies where and how messages should be sent and received.

2. An end point to which messages are sent, such as a queue or topic.

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 iSeries 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.

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) that uniquely identifies an entity in a digital certificate. Note that the Distinguished Name is unique only within the namespace of a given certificate authority. It is entirely possible that certificates with identical distinguished names can be issued by different certificate authorities. Therefore, ensure that a key repository contains as few trusted root CA certificates as possible, preferably no more than one. See also certificate authority, digital certificate, X509.

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 queue management
In message queuing, the setup and control of message channels to queue managers on other systems.

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 WebSphere 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 a subscribing application's connection 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 WebSphere 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 in the event of a software, hardware, or network interruption.

FAP See Formats and Protocols.

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 United States Federal Information Processing Standards

first-failure data capture (FFDC)

1. A problem diagnosis aid that identifies errors, gathers and logs information about these errors, and returns control to the affected runtime software.

2. The IBM i implementation of the FFST architecture providing problem recognition, selective dump of diagnostic data, symptom string generation, and problem log entry.

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 WebSphere 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 WebSphere 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 WebSphere MQ, a collection of programming interfaces that allow customers or vendors to write programs that extend or replace certain functions provided in WebSphere 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
A WebSphere MQ for z/OS trace option where the trace data comes from the entire WebSphere 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

idenitity context
Information that identifies the user of the application that puts the message on a queue first.

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 WebSphere 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 WebSphere MQ Administration Interface (MQAI), a means of referencing data items.

in-doubt unit of recovery
The status of a unit of recovery for which a syncpoint 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 used by WebSphere MQ for z/OS when it starts up.

initiation queue
A local queue on which the queue manager puts trigger messages.

initiator
In distributed queueing, a program that requests network connections on another system. See also responder.

input parameter
A parameter of an MQI call in which you supply information when you make the call.

insertion order
In the WebSphere MQ Administration Interface (MQAI), the order that data items are placed into a data bag.

installable service
In WebSphere MQ on UNIX and Linux systems and WebSphere 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.

instrumentation event
A way of monitoring queue manager resource definitions, performance conditions, and channel conditions in a network of WebSphere 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 run-time bindable application program 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 terminal user.

Intermediate certificate
A signer certificate that is not the root certificate.

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 WebSphere MQ for iSeries 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
Generic term for a store for digital certificates and their associated keys. Different types of key repository include Certificate Management System (CMS), Java Keystore (JKS), Java Cryptography Extension Keystore (JCEKS), Public Key Cryptography Standard 12 (PKCS12) Keystore, and RACF key rings. When it is important to differentiate between key repository types, the documentation refers to the key repository type by its specific name. In contexts applicable to multiple keystore types, the generic term key repository is used.

key ring
In computer security, a file that contains public keys, private keys, trusted roots, and certificates.

key store
The place for a private key and corresponding personal certificate. See also trust store

L

last will and testament
An object that is registered by a client with a monitor, 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 WebSphere MQ on UNIX and Linux systems, and WebSphere 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.

link level security
The security services that are invoked, 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
A WebSphere 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 WebSphere 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 WebSphere MQ on UNIX and Linux systems, and WebSphere 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 WebSphere MQ on UNIX and Linux systems, and WebSphere 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, WebSphere 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 conversation level security is also known as end user verification.

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 WebSphere MQ on UNIX and Linux systems and WebSphere 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. A communication sent from a person or program to another person or program.

2. In system programming, information intended for the terminal operator or system administrator.

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 WebSphere MQ interface to which customer- or vendor-written programs that transmit

messages between a WebSphere MQ queue manager and another messaging system must conform. A part of the WebSphere MQ Framework. See also Message Queue Interface.

message consumer

1. A program or function that gets and processes messages. See also consumer.

2. In JMS, an object that is created within a session to receive messages from a destination.

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 WebSphere 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 will be 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 WebSphere 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.

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 WebSphere 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 WebSphere MQ for HP Integrity 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 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.

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 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 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 WebSphere MQ Administration Interface.

MQI See Message Queue Interface.

MQI channel
A connection between a WebSphere 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 WebSphere MQ script commands.

MQSeries
A previous name for WebSphere MQ.

MQ Telemetry Transport
MQ Telemetry Transport (MQTT) is an open, lightweight publish/subscribe 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 to work in environments where the network might be constrained by bandwidth, or the device might be constrained by memory or processors for example.

MQTT
See MQ Telemetry Transport.

MQTT client
An MQTT client application connects to MQTT capable servers such as WebSphere MQ Telemetry channels. You can write your own clients to use the published protocol, or use one of the clients supplied with the installation of WebSphere MQ Telemetry. 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. Some clients are provided with WebSphere MQ Telemetry; see Telemetry clients and Telemetry advanced clients.

MQTT server
An MQTT server handles the server side of the MQTT protocol. 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. A WebSphere MQ queue manager with the telemetry service is an MQTT server.

MSCS See Microsoft Cluster Server.

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 single instance queue manager.

N

namelist
A WebSphere MQ object that contains a list of object names, for example, queue names.

name service
In WebSphere MQ on UNIX and Linux systems and WebSphere MQ for Windows, the facility that determines which queue manager owns a specified queue.

name service interface (NSI)
The WebSphere MQ interface to which customer- or vendor-written programs that resolve queue-name ownership must conform. A part of the WebSphere MQ Framework.

name transformation
In WebSphere MQ on UNIX and Linux systems and WebSphere 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 WebSphere MQ Administration Interface (MQAI), a system bag that is inserted into another data bag

nesting
In the WebSphere MQ Administration Interface (MQAI), a means of grouping information returned from WebSphere 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 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. 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 WebSphere 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 WebSphere MQ on UNIX and Linux systems, WebSphere MQ for IBM i, and WebSphere 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 WebSphere 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 WebSphere 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 comprise the problem and how they are manipulated.

OCSP Online Certificate Status Protocol. A method of checking if a certificate is revoked.

offloading
In WebSphere MQ for z/OS, an automatic process whereby a queue manager's active log is transferred to its archive log.

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 or a topic

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 WebSphere 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 WebSphere 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
A WebSphere 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 WebSphere MQ for z/OS, a set of backup copies of WebSphere 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 parent class's method. 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 WebSphere 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
A WebSphere MQ object that contains the definition of a WebSphere 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 publisher, message producer.

programmable command format (PCF)
A type of WebSphere 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 WebSphere MQ script commands.

program temporary fix (PTF)
For System i, System p, and System z 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.

PTF See program temporary fix.

public key
The key known to everyone. This key is usually embedded in a digital certificate that specifies the owner of the public key.

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 anything encrypted with one key can be decrypted only by the corresponding private key.

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.

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.

publish/subscribe
A type of messaging interaction in which information, provided by publishing applications, is delivered by an infrastructure to all subscribing applications that have expressed interest in 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-queueing applications. A queue is owned and maintained by a queue manager.

queue index
In WebSphere 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.

queue manager event
An event that indicates one of the following: 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 WebSphere 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 WebSphere MQ Explorer that allows a user to perform actions on all of the queue managers in the group.

queue-sharing group
In WebSphere 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 WebSphere 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. A type of shutdown of the CICS adapter where the adapter disconnects from WebSphere MQ, but only after all the currently active tasks have been completed. See also forced shutdown.

2. In WebSphere MQ, a shutdown of a queue manager that allows all connected applications to disconnect. See also immediate shutdown, preemptive shutdown.

quiescing
In WebSphere 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 WebSphere MQ for z/OS, data sets containing information needed to recover messages, queues, and the WebSphere 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, the program item 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 exploits 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
A WebSphere 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 WebSphere 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 a WebSphere 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. WebSphere 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 WebSphere MQ, an option to direct a channel to start up 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

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 WebSphere MQ interface to which customer- or vendor-written programs that check authorization, supply a user identifier, or perform authentication must conform. A part of the WebSphere MQ Framework.

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.

self-signed certificate
The digital signature in the certificate is generated using the private key corresponding to the public key in the certificate.

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 WebSphere MQ Administration Interface (MQAI), there are two types of selector: a user selector and a system selector.

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 WebSphere 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 queue manager that provides queue services to client applications running on a remote workstation.

2. A software program or a computer that provides services to other software programs or other computers. See also client.

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
In WebSphere MQ for z/OS, the CICS-unique identifier 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere MQ for z/OS and WebSphere MQ for Windows, 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
A certificate that is used for encipherment or signing.

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 WebSphere 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 WebSphere 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
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 WebSphere 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 WebSphere MQ for z/OS, a RACF profile used when WebSphere MQ starts up or when a refresh security command is issued. Each switch profile that WebSphere 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. This system does not offer any authentication. 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 on start up.

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 WebSphere 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 Telemetry channel is a communication link between a queue manager on WebSphere MQ, and MQTT clients. Each channel might have one or more telemetry devices connected to it.

telemetry advance client
The Advanced telemetry client is installed in the mqxr subfolder of the main WebSphere MQ installation. They are small footprint, MQTT servers allowing multiple MQTT clients to connect to it and provide an uplink or bridge to WebSphere MQ. Advanced clients can start messages on behalf of the clients when the uplink connection is broken.

telemetry client
Telemetry clients are MQTT clients installed within the mqxr subfolder of the main WebSphere MQ installation. The telemetry clients use the MQTT protocol to connect to MQ.

telemetry service
An MQ service that handles the server half of the MQTT protocol (see MWTT Server). The telemetry service hosts telemetry channels.

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 WebSphere 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 Transport Layer Security - successor to SSL.

TMF See Transaction Manager Facility.

TMI See trigger monitor interface.

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 WebSphere MQ for HP Integrity NonStop Server, a subsystem to protect your business transactions and the integrity of your databases. Often used synonymously with NonStop Transaction Manager/MP.

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.

triggered queue
A local queue which, when it has triggering set on and when the triggering conditions are met, requires that trigger messages are written.

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 WebSphere 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 WebSphere MQ interface to which customer- or vendor-written trigger monitor programs must conform. A part of the WebSphere MQ Framework.

trust store
The place where CA certificates are put to validate certificates from a remote system. See also key store

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.

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.

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 a WebSphere 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 syncpoint. It ends either at a user-requested syncpoint 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 WebSphere MQ Administration Interface (MQAI), the identifier that is placed with a data item into a data bag to identify the data item. WebSphere MQ provides predefined user selectors for WebSphere 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 WebSphere MQ, a supplied set of programs that provide the system operator or system administrator with facilities in addition to those provided by the WebSphere MQ commands.

UTOKEN
See user token.

V

value The content of a data item. This 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

WebSphere MQ
A family of IBM licensed programs that provides message queuing services.

WebSphere MQ Administration Interface (MQAI)
A programming interface that performs administration tasks on a WebSphere MQ queue manager through the use of data bags. Data bags allow the user to handle properties (or parameters) of WebSphere MQ objects.

WebSphere MQ classes for .NET
A set of classes that allow a program written in the .NET programming framework to connect to WebSphere MQ as a WebSphere MQ client or to connect directly to a WebSphere MQ server.

WebSphere MQ classes for C++
A set of classes that encapsulate the WebSphere MQ Message Queue Interface (MQI) in the C++ programming language.

WebSphere MQ classes for Java
A set of classes that encapsulate the WebSphere MQ Message Queue Interface (MQI) in the Java programming language.

WebSphere MQ fully-managed .NET client
Part of a WebSphere MQ product that can be installed on a system without installing the full queue manager. The WebSphere 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 WebSphere MQ MQI client. See also client, WebSphere MQ MQI client, WebSphere MQ Java client.

WebSphere MQ Java client
Part of a WebSphere MQ product that can be installed on a system without installing the full queue manager. The WebSphere MQ Java client is used by Java applications (both WebSphere MQ classes for Java and WebSphere MQ classes for JMS) and communicates with a queue manager on a server system. See also client, WebSphere MQ MQI client, WebSphere MQ fully-managed .NET client.

WebSphere MQ MQI client
Part of a WebSphere MQ product that can be installed on a system without installing the full queue manager. The WebSphere MQ MQI client accepts MQI calls from applications and communicates with a queue manager on a server system. See also client, WebSphere MQ Java client, WebSphere MQ fully-managed .NET client.

WebSphere MQ script commands (MQSC)
Human readable commands, uniform across all platforms, that are used to manipulate WebSphere MQ objects. See also programmable command format.

WebSphere MQ server
A queue manager that provides queuing services to one or more clients. All the WebSphere 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.

WebSphere MQ Telemetry
WebSphere MQ Telemetry provides small client libraries that can be embedded into smart devices running on a number of different device platforms. Applications built with the clients use MQ Telemetry Transport (MQTT) and the WebSphere MQ Telemetry service to publish and subscribe messages reliably with WebSphere MQ. When the WebSphere MQ custom installation option to install Telemetry is selected, it installs: 1) Telemetry service 2) Telemetry clients and 3) Telemetry Advanced clients.

WebSphere MQ Telemetry daemon for devices
The WebSphere MQ Telemetry daemon for devices is an advanced MQTT V3 client. It is a very small footprint MQTT server designed for embedded systems.

Windows NT Challenge/Response
The authentication protocol that is used on networks that include Windows NT systems and on standalone systems.

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.

Notices

This information was developed for products and services offered in the United States. IBM may not offer the products, services, or features discussed in this information 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 information. The furnishing of this information does not give 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:

IBM World Trade Asia Corporation,
Licensing,
2-31 Roppongi 3-chome, Minato-k,u
Tokyo 106, 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 information. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this information 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 United Kingdom Laboratories,
Mail Point 151,
Hursley Park,
Winchester,

Hampshire,
England
SO21 2JN.

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 Programming License Agreement, or any equivalent agreement between us.

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.

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. 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 IBM's application programming interfaces.

The following are trademarks of International Business Machines Corporation in the United States, or other countries, or both:

AIX
CICS
Db2
DB2 Universal Database™
FFST™
First Failure Support Technology™
IBM i
IBM
IBMLink
IMS
Integrated Language Environment®
MQSeries®
MVS™
OS/390®
Parallel Sysplex®
pSeries
RACF
SupportPac
VTAM
WebSphere
z/OS
zSeries

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft, Windows, and Windows NT 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.

Other company, product, or service names may be trademarks or service marks of others.

Accessibility features for IBM WebSphere MQ

Accessibility features help users who have a physical disability, such as restricted mobility or limited vision, to use information technology products successfully.

The following list includes the major accessibility features in IBM WebSphere MQ. You can use screen-reader software to hear what is displayed on the screen.

- Supports keyboard-only operation
- Supports interfaces commonly used by screen readers

Tip: This product documentation, and its related publications, are accessibility-enabled for the IBM Home Page Reader. You can operate all features using the keyboard instead of the mouse.

Keyboard navigation

This product uses standard Linux and Microsoft Windows navigation keys.

For more information, see Accessibility features.

Visit the http://www.ibm.com/able for more information about the commitments that IBM makes towards accessibility.

Accessibility

The IBM WebSphere MQ user interfaces do not use any special keys, but instead follow the Windows user interface guidelines for accelerator keys on items such as context menus, dialogs, and dialog controls such as buttons. Access the accelerator keys in the usual way. See the Windows help for more information (look in the Windows help index for *keyboard*; for accessibility features look for *Accessibility*).

Special features for accessibility

Some of the user interfaces in IBM WebSphere MQ are normally visual, but they behave differently when accessibility features are activated, as follows:

- High Contrast Mode

  In this mode Launchpad, Prepare IBM WebSphere MQ Wizard, Postcard, and Default Configuration all hide their background bitmaps and ensure that they use the system text colors so that they are easily visible and readable.
- Screen Reader Mode

  When a screen reader is active, Prepare IBM WebSphere MQ Wizard, Default Configuration, and Postcard, simplify their appearance by hiding background bitmaps, raised effects, shadow boxes, and other effects that can otherwise confuse the screen reader.
- Explorer Object Status

  The Explorer component of IBM WebSphere MQ uses icons to indicate the status of objects, such as queue managers. Screen readers cannot interpret these icons, so there is an option to show a textual description of the icon. To select this option, from within the Explorer click **Window** > **Preferences** > **WebSphere MQ Explorer** and select **Show status of objects after object name**.

Legal information for WebSphere MQ

Read the information that describes the legal statements for this product.

- "Notices"
- "Trademarks" on page 354

This product documentation provides links or references to non-IBM Web sites and resources. IBM makes no representations, warranties, or other commitments whatsoever about any non-IBM Web sites or third-party resources (including any Lenovo Web site) that may be referenced, accessible from, or linked to any IBM site. A link to a non-IBM Web site does not mean that IBM endorses the content or use of such Web site or its owner. In addition, IBM is not a party to or responsible for any transactions you may enter into with third parties, even if you learn of such parties (or use a link to such parties) from an IBM site. Accordingly, you acknowledge and agree that IBM is not responsible for the availability of such external sites or resources, and is not responsible or liable for any content, services, products, or other materials on or available from those sites or resources. When you access a non-IBM Web site, even one that may contain the IBM-logo, be aware that it is independent from IBM, and that IBM does not control the content on that Web site. It is up to you to take precautions to protect yourself from viruses, worms, trojan horses, and other potentially destructive programs, and to protect your information as you deem appropriate.

Notices

Legal notices

This information was developed for products and services offered in the United States. IBM may not offer the products, services, or features discussed in this information 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 information. The furnishing of this information does not give 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 information. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this information 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 United Kingdom Laboratories,
Mail Point 151,
Hursley Park,
Winchester,
Hampshire,
England,
SO21 2JN

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 Programming License Agreement, or any equivalent agreement between us.

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.

COPYRIGHT LICENSE:

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

Trademarks

Trademarks in this product documentation

The following terms are trademarks of International Business Machines Corporation in the United States, other countries, or both:

AD/Cycle
AIX
AS/400
C/370
C/400
C/MVS
CICS
CICS/400
CICS/ESA
CICS/VSE
COBOL/400
Common User Access
DB2
DB2 Universal Database
DFSMS
DFSMSdss
DFSMS/MVS
Encina
Extended Services
FFST
First Failure Support Technology
HACMP
IBM i
IBM
IBMLink
IMS
IMS/ESA
Informix
Integrated Language Environment
iSeries
Language Environment
Lotus
Lotus Notes
LotusScript
MQSeries
MVS
MVS/DFP
MVS/ESA
NetView
Notes
OS/390
OS/400
Parallel Sysplex
pSeries
POWER
PowerPC
RACF
RAMAC
Redbooks
Retain
REXX
RPG/400
RMF
S/390
SAA
SecureWay
SP
SupportPac
System/360
System/370
System/390
System i
System z
Tivoli
TXSeries
VisualAge
VM/ESA
VSE/ESA
VTAM
WebSphere
z/OS
zSeries

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in countries such as the United States.

Intel, Intel logo, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium and Pentium are trademarks of Intel Corporation in the United States, other countries, or both.

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

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

Microsoft, Windows, Windows NT, and the Windows logo 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.

Index

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.

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.