

IBM WebSphere MQ



Product Overview

Version 7 Release 5

Note

Before using this information and the product it supports, read the information in "Notices" on page 217.

This edition applies to version 7 release 5 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.

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

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

Contents

Figures v

Tables vii

Product overview. 1

Introduction to IBM WebSphere MQ 1

IBM WebSphere MQ license information 4

Introduction to IBM WebSphere MQ Telemetry 6

IBM WebSphere MQ information roadmap 9

IBM WebSphere MQ Version 7.5 PDF documentation 12

What's new in IBM WebSphere MQ Version 7.5 12

 IBM WebSphere MQ client for HP Integrity

 NonStop Server 14

What's changed in IBM WebSphere MQ Version 7.5 17

What's changed in IBM WebSphere MQ Version 7.5

Fix Packs 18

Mappings between earlier IBM WebSphere MQ

publications and the current information structure . 23

 Quick beginnings for AIX 24

 Quick beginnings for HP-UX 24

 Quick beginnings for Linux 24

 Quick beginnings for Solaris. 25

 Quick beginnings for Windows. 25

 Application programming guide 25

 Application programming reference 26

 Clients 26

 Constants 27

 Intercommunication 27

 Messages and codes 28

 Migration 28

 Monitoring 28

 Programmable Command Formats and

 Administration Interface 28

 Publish/Subscribe User's Guide 28

 Queue manager clusters 29

 Script (MQSC) Command Reference 29

 Security 29

 System Administration Guide 29

 Using .NET 30

 Using C++. 30

 Using Java. 30

 Web services 30

 Using the Component Object Model Interface . . 30

IBM WebSphere MQ Version 7.5, IBM i and z/OS 30

IBM WebSphere MQ Technical overview. 31

 Introduction to message queuing 31

 Concepts of intercommunication 39

 IBM WebSphere MQ Telemetry 58

 Administering objects 93

 WebSphere MQ Multicast 118

 Security 119

 Clients and servers 120

 Transaction management and support 127

 Extending queue manager facilities 129

 IBM WebSphere MQ client for HP Integrity

 NonStop Server technical overview 130

Scenarios 132

 Getting started with WebSphere MQ V 7.5 132

 Basic file transfer using the scripts 139

 Basic file transfer in detail 147

 Two computer file transfer using the scripts . . . 152

 Two computer file transfer in detail 158

 Adding audit capability to managed file transfer 165

Glossary 168

 A 168

 B 171

 C 172

 D 177

 E 179

 F 180

 G 182

 H 182

 I 183

 J 185

 K 185

 L 186

 M 187

 N 191

 O 192

 P 194

 Q 197

 R 198

 S 200

 T 206

 U 208

 V 209

 W 209

 X 210

Accessibility features for IBM WebSphere MQ . . 210

 Accessibility on Windows 212

Index 213

Notices 217

 Programming interface information 218

 Trademarks 219

Sending your comments to IBM . . . 221

Figures

1. Smart electricity metering	6	18. A cluster of queue managers	56
2. Smart health monitoring	7	19. A cluster of queue managers with sender channels	57
3. Telemetry: One in a Crowd.	7	20. A cluster of queue managers, showing auto-defined channels	58
4. Message queuing compared with traditional communication	34	21. MQTT v3 client subscriber	72
5. Overview of the components of distributed queuing.	40	22. WebSphere MQ publisher	72
6. Sending messages	41	23. messageArrived method	72
7. Sending messages in both directions	42	24. MQTT v3 client publisher.	73
8. A cluster of queue managers.	43	25. Console log from starting connection bridge	83
9. A sender-receiver channel.	45	26. Alter maximum number of handles on Windows	92
10. A requester-server channel	45	27. Alter maximum number of handles on Linux	92
11. A requester-sender channel	46	28. Message channels between two queue managers	108
12. A cluster-sender channel	46	29. Client-connection and server-connection on an MQI channel.	109
13. Passing through intermediate queue managers	48	30. Link between a client and server	121
14. Sharing a transmission queue	48	31. WebSphere MQ server connected to clients on different platforms	127
15. Using multiple channels	49		
16. Queue manager alias	51		
17. Reply-to queue alias used for changing reply location	53		

Tables

1. IBM WebSphere MQ information roadmap table	10	6. MQTT fixed header properties	86
2. Mapping of PDF files to product documentation sections	12	7. MQTT Variable header properties	87
3. MQMD	85	8. Transmission protocols for MQI channels	110
4. RFH2	86	9. MQGET options permitted when read ahead is enabled.	123
5. Message contents	86	10.	131

Product overview

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

Related information:

Designing a WebSphere MQ architecture

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?

- IBM WebSphere MQ is messaging for applications. It sends messages across networks of diverse components. Your application connects to IBM WebSphere MQ to send or receive a message. IBM WebSphere MQ handles the different processors, operating systems, subsystems, and communication protocols it encounters in transferring the message. If a connection or a processor is temporarily unavailable, IBM WebSphere MQ queues the message and forwards it when the connection is back online.
- An application has a choice of programming interfaces, and programming languages to connect to IBM WebSphere MQ.
- IBM WebSphere MQ is *messaging* and *queuing* middleware, with *point-to-point*, *publish/subscribe*, and *file transfer* modes of operation. Applications can publish messages to many subscribers over *multicast*.

Messaging

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

Queuing

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

Point-to-point

Applications send messages to a queue, or to a list of queues. The sender must know the name of the destination, but not where it is.

Publish/subscribe

Applications publish a message on a topic, such as the result of a game played by a team. IBM WebSphere MQ sends copies of the message to applications that subscribe to the results topic. They receive the message with the results of games played by the team. The publisher does not know the names of subscribers, or where they are.

Multicast

Multicast is an efficient form of publish/subscribe messaging that scales to many subscribers. It transfers the effort of sending a copy of a publication to each subscriber from IBM WebSphere MQ to the network. Once a path for the publication is established between the publisher and subscriber, IBM WebSphere MQ is not involved in forwarding the publication.

File transfer

Files are transferred in messages. IBM WebSphere MQ File Transfer Edition manages the transfer of files and the administration to set up automated transfers and log the results. You can integrate the file transfer with other file transfer systems, with IBM WebSphere MQ messaging, and the web.

Telemetry

IBM WebSphere MQ Telemetry is messaging for devices. IBM WebSphere MQ connects device and application messaging together. It connects the internet, applications, services, and decision makers with networks of instrumented devices. IBM WebSphere MQ Telemetry has an efficient messaging protocol that connects a large numbers of devices over a network. The messaging protocol is published, so that it can be incorporated into devices. You can also develop device programs with one of the published programming interfaces for the protocol.

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

How do I use it?

- Create and manage IBM WebSphere MQ with the IBM WebSphere MQ Explorer GUI or by running commands from a command window or application.
- Program applications to send and receive messages by calling one of the programming interfaces. Programming interfaces are provided for different languages, and include the standard JMS programming interface, and classes for the Windows communication foundation.
- Send and receive IBM WebSphere MQ messages from browsers with the HTTP protocol.

How does it work?

- An administrator creates and starts a queue manager with commands. Subsequently, the queue manager is usually started automatically when the operating system boots. Applications, and other queue managers can then connect to it to send and receive messages.
- An application or administrator creates a queue or a topic. Queues and topics are objects that are owned and stored by a *queue manager*.
- When your application wants to transfer data to another application, it puts the data into a message. It puts the message onto a queue, or publishes the message to a topic. There are three main ways that the message can be retrieved:

- A point-to-point application connected to the same queue manager retrieves the message from the same queue.

For example, an application puts messages on a queue as way of storing temporary or persistent data. A second example: An application that shares data with another application that is running in a different process.

- A point-to-point application connected to another queue manager retrieves the same message from a different queue.

Applications communicate with each other by exchanging messages on queues. The main use of IBM WebSphere MQ is to send or exchange messages. One application puts a message on a queue on one computer, and another application gets the same message from another queue on a different computer. The queue managers on the two computers work together to transfer the message from the first queue to the second queue. The applications do not communicate with each other, the queue managers do.

- A subscriber application connected to any queue manager retrieves messages on common topics.

A publisher application creates a message and publishes it to a topic on one computer. Any number of subscriber applications subscribe to the same topic on different computers. IBM

WebSphere MQ delivers the publication to queues that belong to the queue managers the subscribers are connected to. The subscribers retrieve the message from the queues.

- *MQ channels* connect one queue manager to another over a network. You can create MQ channels yourself, or a queue manager in a cluster of queue managers creates MQ channels when they are needed.
- You can have many queues and topics on one queue manager.
- You can have more than one queue manager on one computer.
- An application can run on the same computer as the queue manager, or on a different one. If it runs on the same computer, it is a IBM WebSphere MQ server application. If it runs on a different computer, it is a IBM WebSphere MQ client application. Whether it is IBM WebSphere MQ client or server makes almost no difference to the application. You can build a client/server application with IBM WebSphere MQ clients or servers.

What tools and resources come with IBM WebSphere MQ?

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

Related concepts:

“What’s new in IBM WebSphere MQ Version 7.5” on page 12

Learn about the main new functions in IBM WebSphere MQ Version 7.5.

WebSphere MQ Multicast

WebSphere MQ Multicast offers low latency, high fan out, reliable multicast messaging.

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.

Technical introduction to messaging and queueing

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.

Technical introduction to WebSphere MQ clients and servers

An introduction to how IBM WebSphere MQ supports client-server configurations for its applications.

Technical introduction to queue manager communication

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).

Related information:

WebSphere MQ Advanced Messages Security (AMS)

WebSphere MQ Managed File Transfer

IBM WebSphere MQ license information

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

What you can purchase with IBM WebSphere MQ

Distributed platforms

For IBM WebSphere MQ on distributed platforms, the product offering contains 11 chargeable components that can be independently purchased:

5724-H72 IBM IBM WebSphere MQ

- IBM IBM WebSphere MQ (Server)

- IBM IBM WebSphere MQ Telemetry

- IBM IBM WebSphere MQ Advanced Message Security

- IBM IBM WebSphere MQ Idle Standby

- IBM IBM WebSphere MQ Advanced Message Security Idle Standby

- IBM IBM WebSphere MQ Advanced

- IBM IBM WebSphere MQ Advanced Idle Standby

- IBM IBM WebSphere MQ Advanced for Developers

- IBM IBM WebSphere MQ Managed File Transfer Service

- IBM IBM WebSphere MQ Managed File Transfer Service Idle Standby

- IBM IBM WebSphere MQ Managed File Transfer Managed Endpoint

What is my enterprise entitled to install?

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

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

5724-H72 IBM IBM WebSphere MQ

- IBM WebSphere (Server)

Includes:

- ClientDevelopment Kit (SDK)

- IBM Global Security Kit (UNIX)

- IBM WebSphere MQ Explorer

- Java™ .NET Messaging and Web Services

- Sample programs

- Server / Runtime

- UNIX Man Pages

IBM IBM WebSphere MQ Telemetry

Includes:

Telemetry Service

IBM IBM WebSphere MQ Advanced Message Security

Includes:

Advanced Message Security

IBM IBM WebSphere MQ Managed File Transfer Service

Includes:

IBM WebSphere MQ Managed File Transfer Logger

IBM WebSphere MQ Managed File Transfer Service

IBM WebSphere MQ Managed File Transfer Tools

IBM WebSphere IBM WebSphere MQ Managed File Transfer Managed Endpoint

Includes:

IBM WebSphere MQ Managed File Transfer Agent

IBM WebSphere MQ Managed File Transfer Logger

IBM WebSphere MQ Managed File Transfer Tools

What is IBM IBM WebSphere MQ Advanced?

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

The IBM WebSphere MQ Advanced parts are:

5724-H72 IBM IBM WebSphere MQ

IBM IBM WebSphere MQ Advanced

IBM IBM WebSphere MQ Advanced for Developers

For IBM IBM WebSphere MQ on distributed platforms, purchasing 100 Processor Value Units (PVUs) of IBM WebSphere MQ Advanced gives your enterprise entitlement to install:

- 100 PVUs of IBM IBM WebSphere MQ (Server), **and**
- 100 PVUs of IBM IBM WebSphere MQ Advanced Message Security, **and**
- 100 PVUs of IBM IBM WebSphere MQ Managed File Transfer Service, **and**
- Unlimited installs of IBM IBM WebSphere MQ Telemetry

Additionally, your enterprise can mix and match IBM versions as required. Therefore, your 100 PVUs of IBM IBM WebSphere MQ (Server) entitlement could be split into 50 PVUs of the IBM WebSphere MQ 7.1 version and 50 PVUs of the IBM WebSphere MQ 7.5 version of this component.

IBM IBM WebSphere MQ Advanced for Developers gives entitlement to everything included with IBM IBM WebSphere MQ Advanced plus IBM IBM WebSphere MQ Managed File Transfer Managed Endpoint for development purposes only.

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

What are Idle Standby parts?

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

Notes:

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

Related concepts:

“What’s changed in IBM WebSphere MQ Version 7.5” on page 17

Review the list of changes carefully before upgrading queue managers to IBM WebSphere MQ Version 7.5. Decide whether you must plan to make changes to existing applications, scripts, and procedures before starting to migrate systems to Version 7.5.

“IBM WebSphere MQ Technical overview” on page 31

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

Introduction to IBM WebSphere MQ Telemetry

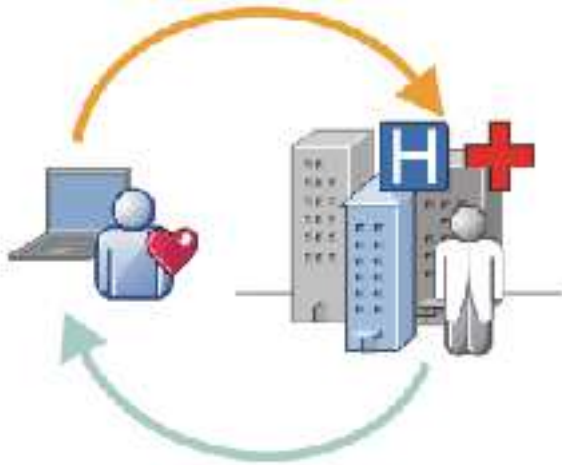
People, businesses, and governments increasingly want to use IBM WebSphere MQ Telemetry to interact more smartly with the environment we live and work in. IBM WebSphere MQ Telemetry connects all kinds of devices to the internet and to the enterprise, and reduces the costs of building applications for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

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



- An MQTT message that contains energy usage data sent to service provider.
- A telemetry application sends control commands that are based on analysis of energy usage data.
- For more information, see “Telemetry scenario: Home energy monitoring and control” on page 64.

Figure 1. Smart electricity metering



- A telemetry application sends your health data to your hospital and doctor.
- MQTT message alerts or feedback are returned, based on analysis of your health data.
- For more information, see “Telemetry scenario: Home patient monitoring” on page 62.

Figure 2. Smart health monitoring



- 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.

Figure 3. Telemetry: One in a Crowd

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 (MQXR) 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 87 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 WebSphere Message Broker and runs on Windows, Linux, and AIX®. Migration of telemetry applications from using WebSphere Message Broker version 6 to use IBM WebSphere MQ Telemetry and WebSphere Message Broker version 7.0 provides information to help you migrate applications from using the SCADA nodes in WebSphere Message Broker V6. Telemetry applications using 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 60

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

Related information:

“IBM WebSphere MQ Telemetry” on page 58

IBM WebSphere MQ Telemetry comprises a telemetry (MQXR) service that is part of a queue manager, telemetry clients that you can write yourself, or use one of the clients that are provided, and command line and explorer administrative interfaces. Telemetry refers to collecting data from and administering a wide range of remote devices. With IBM WebSphere MQ Telemetry you can integrate the collection of data and control of devices with web applications.

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

Installing WebSphere MQ Telemetry

Administering WebSphere MQ Telemetry

Migration of telemetry applications from using WebSphere Message Broker version 6 to use WebSphere MQ Telemetry and WebSphere Message Broker version 7.0

Developing applications for WebSphere MQ Telemetry

WebSphere MQ Telemetry Reference

Troubleshooting for WebSphere MQ Telemetry

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
- Scenarios
- 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	<p>Overview of the overall purpose, capabilities, and new features of IBM WebSphere MQ.</p> <p>“Product overview” on page 1 Introductory information to help you get started with IBM WebSphere MQ Version 7.5, including an introduction to the product and an overview of what is new and what is changed for this release.</p> <p>IBM WebSphere MQ Version 7.5 This IBM Redbooks publication covers the core enhancements made in IBM WebSphere MQ Version 7.5 and the concepts that must be understood.</p> <p>IBM WebSphere MQ product web page Product web page with links to resources and additional information.</p> <p>IBM WebSphere MQ system requirements Web page with links to the system requirements for the different releases of IBM WebSphere MQ.</p> <p>IBM 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.</p>
Technical overview	<p>“IBM WebSphere MQ Technical overview” on page 31 Information to help you to find out about message queuing and other features that IBM WebSphere MQ Version 7.5 provides.</p>
Scenarios	<p>Each scenario takes you through a significant set of tasks, and helps you to configure a major product feature. The scenarios include useful links to other content to help you to gain a better understanding of the area in which you are interested.</p> <p>“Scenarios” on page 132 The <i>Getting started</i> scenario explains how to get started with IBM WebSphere MQ. Use this scenario if you have not used IBM WebSphere MQ before and want to get started quickly. Further scenarios help you to configure or use product features by taking you through the appropriate task steps.</p> <p>Connecting WebSphere Application Server to IBM WebSphere MQ Contains information that leads you through the key tasks required to connect WebSphere Application Server to IBM WebSphere MQ in a variety of scenarios.</p> <p>Connecting WebSphere Application Server Liberty profile to IBM WebSphere MQ Contains information that leads you through the key tasks required to connect WebSphere Application Server Liberty profile to IBM WebSphere MQ in a variety of scenarios.</p> <p>Connecting IBM MessageSight to IBM WebSphere MQ and WebSphere Application Server Contains information that leads you through the key tasks required to connect IBM MessageSight to IBM WebSphere MQ and WebSphere Application Server in a variety of scenarios.</p>
Planning	<p>Planning 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.</p>

Table 1. IBM WebSphere MQ information roadmap table (continued)

Category	Information resources
Migrating and upgrading	<p>Migrating and upgrading 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</p> <p>IBM WebSphere MQ Self-help Migration Guide This guide provides information to help you plan the process of migrating from an older version to a new version. You can either view the guide in your web browser or download it as a PDF file.</p>
Installing	<p>Installing and uninstalling Information to help you to prepare for installation, install the product, and verify the installation. There is also information to help you to uninstall the product.</p>
Security	<p>Security Aspects of security to consider in your IBM WebSphere MQ installation including identification and authentication, authorization, auditing, confidentiality, and data integrity.</p>
Configuring	<p>Configuring Create one or more queue managers on one or more computers, and configure them and their related resources on your development, test, and production systems to process messages that contain your business data.</p>
Administering	<p>Administering IBM WebSphere MQ Administer your queue managers and associated resources.</p>
Developing applications	<p>Developing applications 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.</p>
Monitoring and performance	<p>Monitoring and performance Monitoring information and guidance to help improve the performance of your queue manager network and tuning tips to help improve the performance of your queue manager network.</p>
Troubleshooting and support	<p>Troubleshooting and support Techniques to help you diagnose and solve problems with your queue manager network or IBM WebSphere MQ applications.</p> <p>IBM SupportAssistant web page The IBM Support Assistant (ISA) helps you to resolve questions and problems with IBM software products by providing access to support-related information and troubleshooting tools.</p> <p>IBM Support Portal web page IBM Support Portal for IBM WebSphere MQ.</p>
Reference	<p>Reference Reference information for configuration, administration, developing applications, telemetry, security, monitoring, troubleshooting and support, and diagnostic messages.</p>

IBM WebSphere MQ Version 7.5 PDF documentation

You can download the IBM WebSphere MQ Version 7.5 documentation as a series of PDF files.

Table 2. Mapping of PDF files to product documentation sections

PDF file name and download link	Product documentation section
wmq75.overview.pdf	Product overview
wmq75.installconfig.pdf	Planning Installing Migrating and upgrading Configuring
wmq75.administer.pdf	Administering Security Monitoring and performance Troubleshooting and support
wmq75.develop.pdf	Developing applications
wmq75.reference.pdf	Reference
wmq75.mft.pdf	IBM WebSphere MQ Managed File Transfer
MQ_Migration_Guide.pdf	Migration guide for IBM WebSphere MQ on distributed systems

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

What's new in IBM WebSphere MQ Version 7.5

Learn about the main new functions in IBM WebSphere MQ Version 7.5.

- “IBM WebSphere MQ Managed File Transfer.”
- “IBM WebSphere MQ Advanced Message Security.”
- “Message Channel Agent (MCA) interception” on page 13.
- “Multiple cluster transmission queues” on page 13.
- “Extended transactional functionality is now a part of the core client” on page 13.
- “Identifying a connection to a queue manager by setting an application name” on page 13.
- “Certificate validation policies” on page 13.
- “More transactional visibility” on page 13.
- “Scenarios” on page 13.
- “IBM WebSphere MQ Explorer” on page 14.

IBM WebSphere MQ Managed File Transfer

IBM WebSphere MQ Managed File Transfer uses IBM WebSphere MQ to transfer files between queue managers. You can extend its reach to workstations and servers that do not have a queue manager. You can extend it using file transfer agents, Apache Ant, and integrating it with IBM Sterling Commerce®:Direct, web gateways, and with SFTP, FTP, or FTPS protocol servers.

With IBM WebSphere MQ Managed File Transfer, you can automate, control, secure, and audit the transfer of files; see IBM WebSphere MQ Managed File Transfer introduction.

IBM WebSphere MQ Advanced Message Security

IBM WebSphere MQ Advanced Message Security (AMS) is a separately installed component, which is separately charged. It provides a high level of protection for sensitive data that is flowing through the

IBM WebSphere MQ network. You do not need to modify existing applications to take advantage of AMS, see WebSphere MQ Advanced Message Security.

Message Channel Agent (MCA) interception

MCA interception feature allows a queue manager running under IBM WebSphere MQ with a licensed install of WebSphere MQ Advanced Message Security to selectively enable policies to be applied for server connection channels. MCA interception allows clients that remain outside WebSphere MQ AMS to still be connected to a queue manager and their messages to be encrypted and decrypted. See Message Channel Agent (MCA) interception.

Multiple cluster transmission queues

You can change the new queue manager attribute **DEFCLXQ** to assign a different cluster transmission queue to each cluster-sender channel. Messages to be forwarded by each cluster-sender channel are placed on separate cluster transmission queues; see Cluster transmission queues and cluster-sender channels . You can also configure cluster transmission queues manually by setting the new queue attribute **CLCHNAME**. You can decide which cluster-sender channels share which transmission queues, which have separate transmission queues, and which use the cluster transmission queue, or queues; see Clustering: Planning how to configure cluster transmission queues. The change assists system administrators that manage the transfer of messages between clustered queue managers.

Extended transactional functionality is now a part of the core client

Extended transactional functionality is now incorporated into the IBM WebSphere MQ core client. You do not need to purchase a separate extended transactional client license, or to install a separate Extended Transactional Client component; see “What is an extended transactional client?” on page 124.

Identifying a connection to a queue manager by setting an application name

An application can set a name that identifies its connection to the queue manager. Display the application name with the **DISPLAY CONN** command. The name is returned in the **APPLTAG** field. You can also display the name in the IBM WebSphere MQ Explorer Application Connections window. The field is called **App name**; see Setting up the WebSphere MQ environment for WebSphere MQ classes for Java. You can set the name of an application connection on all platforms, except z/OS®.

Certificate validation policies

On UNIX, Linux, and Windows, you can specify how strictly the certificate chain validation conforms to the RFC 5280 industry security standard; see Certificate validation policies in WebSphere MQ.

More transactional visibility

The **dspmqrn** command has two new parameters: **-a** and **-q** to provide more information when an asynchronous rollback occurs. Two new messages AMQ7486 and AMQ7487 provide information about the transaction that is being rolled back, and whether the transaction is associated with a connection.

Scenarios

“Scenarios” on page 132 show you how to quickly and easily use and combine new IBM WebSphere MQ Version 7.5 function. The scenarios include useful links to product documentation content to help you to gain a better understanding of the area in which you are interested.

IBM WebSphere MQ Explorer

New features in IBM WebSphere MQ Explorer Version 7.5 include the integration of IBM WebSphere MQ Managed File Transfer and IBM WebSphere MQ Advanced Message Security, improved multi version support, and reduced overhead.

For more details about what's new in IBM WebSphere MQ Explorer, see *What's new and what's changed in WebSphere MQ Explorer*.

Related concepts:

“What's changed in IBM WebSphere MQ Version 7.5” on page 17

Review the list of changes carefully before upgrading queue managers to IBM WebSphere MQ Version 7.5. Decide whether you must plan to make changes to existing applications, scripts, and procedures before starting to migrate systems to Version 7.5.

“What's changed in IBM WebSphere MQ Version 7.5 Fix Packs” on page 18

Changes to functions and resources in Version 7.5 Fix Packs are described in this section.

IBM WebSphere MQ client for HP Integrity NonStop Server

Learn about the IBM WebSphere MQ client for the HP Integrity NonStop Server platform.

IBM WebSphere MQ now supports the client for the HP Integrity NonStop Server platform. The client is released in SupportPac MAT1, *IBM WebSphere MQ Clients for NSS*, for details, see http://www.ibm.com/support/docview.wss?rs=171&uid=swg24034802&loc=en_US&cs=utf-8&lang=en.

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 121.

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 130.

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 131.

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*.

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*.
- Planning your installation, see *Planning your installation on HP Integrity NonStop Server*.
 - File system
- Hardware and software requirements, see *Hardware and software requirements on HP Integrity NonStop Server*.
- Verifying that you have the correct software, see *Verifying system software prerequisites*.
- Preparing your system, see *Setting up the user and group on HP Integrity NonStop Server*.
- Installing the client, see *Installing IBM WebSphere MQ client on HP Integrity NonStop Server systems*.
- Verifying your installation, see *Verifying a client installation*.

- Uninstalling, see Uninstalling IBM WebSphere MQ on HP Integrity NonStop Server.

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
- endmqtrc
- mqrc (MQ return code)
- runmqras
- runmqtmc
- strmqtrc

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

- dspmqtrc

New Product Identifier, MQNC, added to the DISPLAY CHSTATUS 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.
- Setting up security on HP Integrity NonStop Server
 - OpenSSL
 - Entropy Daemon
- IBM WebSphere MQ support for SSL and TLS
- Working with SSL or TLS on HP Integrity NonStop Server
 - Certificate management
 - Personal certificate store
 - Certificate trust store
 - Pass phrase stash file
 - Certificate revocation list file

Transaction Management Facility

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

- Planning your IBM WebSphere MQ client environment on HP Integrity NonStop Server
 - Preparing the HP Integrity NonStop Server environment
 - IBM WebSphere MQ and HP NonStop TMF
 - Using HP NonStop TMF
 - Using global units of work
 - Avoiding long running transactions
 - Information about queue manager configuration to expire global units of work after a pre-configured interval of inactivity added to Expiring global units of work.
- Configuring HP Integrity NonStop Server

- Gateway process overview
- Configuring Gateway to run under Pathway
- TMF and TMF/Gateway stanzas
- Configuring the client initialization file
- Granting permissions to channels
- Administering HP Integrity NonStop Server
 - Manually starting the TMF/Gateway from Pathway
 - Stopping the TMF/Gateway from Pathway
- Troubleshooting IBM WebSphere MQ client for HP Integrity NonStop Server

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
 - OSS and Guardian headers and public libraries
 - Preparing C programs in HP Integrity NonStop Server
 - Preparing COBOL programs
 - Preparing pTAL programs
- Coding in pTAL.
- Preparing JMS programs for the IBM WebSphere MQ client for HP Integrity NonStop Server.

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
- 2355 (0933) (RC2355): MQRC_UOW_MIX_NOT_SUPPORTED
- 2072 (0818) (RC2072): MQRC_SYNCPOINT_NOT_AVAILABLE
- 2003 (07D3) (RC2003): MQRC_BACKED_OUT

Samples

For information about the techniques demonstrated by the sample programs, see Samples for IBM WebSphere MQ client for HP Integrity NonStop Server.

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
- Error logs on HP Integrity NonStop Server
- Using trace on HP Integrity NonStop Server
- FFST™: WebSphere MQ for HP Integrity NonStop Server

What's changed in IBM WebSphere MQ Version 7.5

Review the list of changes carefully before upgrading queue managers to IBM WebSphere MQ Version 7.5. Decide whether you must plan to make changes to existing applications, scripts, and procedures before starting to migrate systems to Version 7.5.

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 7.5” on page 12.

List of changes by version, release, and maintenance level

- V7.1 to V7.5 changes
- V7.0.1 to V7.5 changes
- V6.0 to V7.5 changes

Related concepts:

“What’s new in IBM WebSphere MQ Version 7.5” on page 12

Learn about the main new functions in IBM IBM WebSphere MQ Version 7.5.

“What’s changed in IBM WebSphere MQ Version 7.5 Fix Packs”

Changes to functions and resources in Version 7.5 Fix Packs are described in this section.

What's changed in IBM WebSphere MQ Version 7.5 Fix Packs

Changes to functions and resources in Version 7.5 Fix Packs are described in this section.

- **V7.5.0.1** “Version 7.5.0, Fix Pack 1: support for the MQTT over WebSockets” on page 19
- **V7.5.0.4** “Version 7.5.0, Fix Pack 4: Disable IBM WebSphere MQ Advanced Message Security at the client (Java clients)” on page 19
- **V7.5.0.5** “Version 7.5.0, Fix Pack 5: Disable IBM WebSphere MQ Advanced Message Security at the client (C clients)” on page 19
- **Windows V7.5.0.6** “Version 7.5.0, Fix Pack 6: userid and password for managed .NET client applications” on page 20
- **V7.5.0.6** “Version 7.5.0, Fix Pack 6: deprecated cipherspecs” on page 20
- **V7.5.0.6** “Version 7.5.0, Fix Pack 6: Serviceability enhancements for IBM WebSphere MQ Managed File Transfer” on page 20
- **V7.5.0.6** “Version 7.5.0, Fix Pack 6 New IBM WebSphere MQ Managed File Transfer agent property failTransferOnFirstFailure” on page 20
- **V7.5.0.7** “Version 7.5.0, Fix Pack 7: deprecated cipherspecs” on page 20
- **Windows V7.5.0.7** “Version 7.5.0, Fix Pack 7: support for JDBC connections to an Oracle 12c database” on page 21
- **Windows V7.5.0.7** “Version 7.5.0, Fix Pack 7: Removal of restriction on using .NET for MQCNO_CLIENT_BINDING and MQCNO_LOCAL_BINDING” on page 21
- **V7.5.0.8** “Version 7.5.0, Fix Pack 8: Restriction on the use of topic alias queues in distribution lists” on page 21
- **V7.5.0.8** “Version 7.5.0, Fix Pack 8: GSKit version updated” on page 21
- **V7.5.0.8** “Version 7.5.0, Fix Pack 8: Deprecated CipherSpecs” on page 21
- **V7.5.0.8** “Version 7.5.0, Fix Pack 8: New constant JMS_IBM_SUBSCRIPTION_USER_DATA added to the JmsConstants interface” on page 22
- **V7.5.0.8** “Version 7.5.0, Fix Pack 8: JMS exception listener updates” on page 22
- **V7.5.0.8** “Version 7.5.0, Fix Pack 8: Support for class name whitelisting in IBM WebSphere MQ classes for JMS ObjectMessage” on page 22
- **V7.5.0.8** “Version 7.5.0, Fix Pack 8: New IBM WebSphere MQ Managed File Transfer agent property additionalWildcardSandboxChecking ” on page 23
- **V7.5.0.8** “Version 7.5.0, Fix Pack 8: Change to behavior of IBM WebSphere MQ Managed File Transfer fteCleanAgent command” on page 23

V7.5.0.1

Version 7.5.0, Fix Pack 1: support for the MQTT over WebSockets

A new communication protocol parameter (PROTOCOL) has been added to the MQTT channel definition (DEFINE CHANNEL (MQTT)):

- If the parameter is set to MQTTV3, the channel only accepts connections from clients using Version 3 of the MQ Telemetry Transfer protocol. This was the only protocol supported before IBM WebSphere MQ Version 7.5.0, Fix Pack 1.
- If the parameter is set to HTTP, the channel only accepts HTTP requests for pages, or WebSockets connections to IBM WebSphere MQ Telemetry.
- If the parameter is set to MQTTV3,HTTP, the channel accepts connections from clients using either protocol. This is the default behavior for new MQTT channels created with IBM WebSphere MQ Version 7.5.0, Fix Pack 1 and later versions.

For more information, see [Connecting the MQTT messaging client for JavaScript over SSL and WebSockets](#).

When a client connects to an MQTT channel using SSL, the parameter SSLCAUTH determines whether IBM WebSphere MQ requires a certificate from the client (see DEFINE CHANNEL (MQTT)). Before IBM WebSphere MQ Version 7.5.0, Fix Pack 1, this parameter could be either REQUIRED or OPTIONAL for MQTT channels:

- REQUIRED means that IBM WebSphere MQ requests a certificate from the client and the client must supply a valid certificate.
- OPTIONAL means that IBM WebSphere MQ will request a certificate from the client but the client does not have to supply one. The client connection is allowed if the client supplies a valid certificate or if the client does not supply a certificate. The client connection is disallowed only if the client supplies an invalid certificate.

In IBM WebSphere MQ Version 7.5.0, Fix Pack 1 and later, the parameter SSLCAUTH can be set to NEVER for MQTT channels. NEVER means that IBM WebSphere MQ never requests a certificate from the client. The new value was added as part of the support for clients using the MQTT messaging client for JavaScript. It accommodates the behavior of some web browsers which treat the request for a client certificate as a protocol error.

V7.5.0.4

Version 7.5.0, Fix Pack 4: Disable IBM WebSphere MQ Advanced Message Security at the client (Java clients)

For Java clients, from IBM WebSphere MQ Version 7.5.0, Fix Pack 4, you can disable IBM WebSphere MQ Advanced Message Security at the client to prevent errors when they are connecting to queue managers that are running on earlier versions of the product. To disable IBM WebSphere MQ Advanced Message Security, you set an environment variable AMQ_DISABLE_CLIENT_AMS. For more information, see [Environment variable used to disable WebSphere MQ AMS at the client](#).

V7.5.0.5

Version 7.5.0, Fix Pack 5: Disable IBM WebSphere MQ Advanced Message Security at the client (C clients)

For C clients, from IBM WebSphere MQ Version 7.5.0, Fix Pack 5, at the client to prevent errors when they are connecting to queue managers that are running on earlier versions of the product. To disable IBM WebSphere MQ Advanced Message Security, you use the DisableClientAMS property, under the **Security** stanza in the mqclient.ini file. For more information, see [Environment variable used to disable WebSphere MQ AMS at the client](#).













Windows V7.5.0.6

Version 7.5.0, Fix Pack 6: userid and password for managed .NET client applications

From IBM WebSphere MQ Version 7.5.0, Fix Pack 6, the userid and password that are specified with the managed .NET client application are set in the IBM WebSphere MQ .NET MQChannelDefinition class that is passed to the client security exit. For more information, see Using channel exits in IBM WebSphere MQ .NET.

Version 7.5.0, Fix Pack 6: deprecated cipherspecs

The following cipherspecs are deprecated from IBM WebSphere MQ Version 7.5.0, Fix Pack 6:

- DES_SHA_EXPORT
-    DES_SHA_EXPORT1024
-    FIPS_WITH_DES_CBC_SHA
- NULL_MD5
- NULL_SHA
- TLS_RSA_WITH_DES_CBC_SHA
- ECDHE_ECDSA_NULL_SHA256
-    ECDHE_RSA_NULL_SHA256
-    TLS_RSA_WITH_NULL_NULL
- TLS_RSA_WITH_NULL_SHA256

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

V7.5.0.6

Version 7.5.0, Fix Pack 6: Serviceability enhancements for IBM WebSphere MQ Managed File Transfer

From IBM WebSphere MQ Version 7.5.0, Fix Pack 6, the following changes have been made for IBM WebSphere MQ Managed File Transfer:

- The default value for of the `commandMessagePriority` property in the `installation.properties` file has changed to 8. For more information, see [The installation.properties file](#).
- The default value for of the `logTransferRecovery` property in the `agent.properties` file has changed to true. For more information, see [The agent.properties file](#).
- The need for the `enableFunctionalFixPack` property to be set in the `installation.properties` file when using the `-d` parameter on the `fteShowAgentDetails` command is removed. for more information, see [fteShowAgentDetails](#) (display WebSphere MQ Managed File Transfer agent details).
- A first failure data capture (FDC) is generated if an agent encounters an unrecoverable error.

V7.5.0.6













Version 7.5.0, Fix Pack 6 New IBM WebSphere MQ Managed File Transfer agent property `failTransferOnFirstFailure`

From IBM WebSphere MQ Version 7.5.0, Fix Pack 6, you can configure an agent to fail a managed transfer as soon as a transfer item within that managed transfer fails by setting the `failTransferOnFirstFailure` property in the `agent.properties` file. For more information, see [The agent.properties file](#).

Version 7.5.0, Fix Pack 7: deprecated cipherspecs

The following cipherspecs are deprecated from IBM WebSphere MQ Version 7.5.0, Fix Pack 7:

- RC2_MD5_EXPORT
- RC4_MD5_EXPORT
- RC4_MD5_US

- RC4_SHA_US
-    RC4_56_SHA_EXPORT1024
-    ECDHE_ECDSA_RC4_128_SHA256
-    ECDHE_RSA_RC4_128_SHA256
-    TLS_RSA_WITH_RC4_128_SHA256

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

Version 7.5.0, Fix Pack 7: support for JDBC connections to an Oracle 12c database

From IBM WebSphere MQ Version 7.5.0, Fix Pack 7, a new file, `jdbcora12.dll`, is supplied with the IBM WebSphere MQ Windows server installation image to support JDBC connections to an Oracle 12c database (see [Configuring JTA/JDBC coordination on Windows](#)).

Version 7.5.0, Fix Pack 7: Removal of restriction on using .NET for MQCNO_CLIENT_BINDING and MQCNO_LOCAL_BINDING

From IBM WebSphere MQ Version 7.5.0, Fix Pack 7, the IBM WebSphere MQ custom channel for Microsoft Windows Communication Foundation (WCF) has been updated so that the correct client connection configuration is used when running from a client-only installation. For more information, see [Connecting to a queue manager using the MQCONN call](#).

Version 7.5.0, 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 7.5.0, 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`.

Version 7.5.0, Fix Pack 8: GSKit version updated

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 7.5.0, 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. For IBM WebSphere MQ Version 7.5, this is Version 7.5.0, Fix Pack 8.

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.

Version 7.5.0, Fix Pack 8: Deprecated CipherSpecs

From Version 7.5.0, Fix Pack 8, the following CipherSpecs are deprecated:

-    FIPS_WITH_3DES_EDE_CBC_SHA
- TRIPLE_DES_SHA_US

- TLS_RSA_WITH_3DES_EDE_CBC_SHA
- Linux Windows UNIX ECDHE_ECDSA_3DES_EDE_CBC_SHA256
- Linux Windows UNIX ECDHE_RSA_3DES_EDE_CBC_SHA256

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

V 7.5.0.8

Version 7.5.0, Fix Pack 8: New constant JMS_IBM_SUBSCRIPTION_USER_DATA added to the JmsConstants interface

From Version 7.5.0, Fix Pack 8, 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 user subscription data and DEFINE SUB](#).

V 7.5.0.8

Version 7.5.0, Fix Pack 8: JMS exception listener updates

From IBM WebSphere MQ Version 7.5.0, Fix Pack 8, to maintain behavior for current JMS applications that configure a JMS MessageListener and a JMS ExceptionListener, the default value for the ASYNC_EXCEPTIONS JMS ConnectionFactory property has been changed to ASYNC_EXCEPTIONS_CONNECTIONBROKEN for the IBM WebSphere MQ classes for JMS for Version 7.5. As a result, by default, only exceptions corresponding to broken connection error codes are delivered to an application's JMS ExceptionListener. Connection broken exceptions are delivered to the exception listener when consuming messages synchronously or asynchronously.

The IBM WebSphere MQ classes for JMS for Version 7.5 have also been updated such that JMS exceptions relating to non-connection broken errors that occur during message delivery to asynchronous message consumers, are delivered to a registered ExceptionListener when the JMS ConnectionFactory used by the application has the ASYNC_EXCEPTIONS property set to the value ASYNC_EXCEPTIONS_ALL.

For more information, see [JMS: Exception listener changes in Version 7.5](#) and [Exceptions in IBM WebSphere MQ classes for JMS](#).

V 7.5.0.8

Version 7.5.0, Fix Pack 8: Support for class name whitelisting in IBM WebSphere MQ classes for JMS ObjectMessage

From IBM WebSphere MQ Version 7.5.0, Fix Pack 8, 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 [Class name whitelisting in JMS ObjectMessage](#) and [Running IBM WebSphere MQ classes for JMS applications under the Java Security Manager](#).

V 7.5.0.8

Version 7.5.0, Fix Pack 8: New IBM WebSphere MQ Managed File Transfer agent property `additionalWildcardSandboxChecking`

From IBM WebSphere MQ Version 7.5.0, Fix Pack 8, if an agent has been configured with a user or agent sandbox in order to restrict the locations that the agent can transfer files to and from, you can specify that additional checks are to be made on wildcard transfers for that agent by setting the `additionalWildcardSandboxChecking` property to true. For more information, see [Additional checks for wildcard transfers](#) and [The agent.properties file](#).

V7.5.0.8

Version 7.5.0, Fix Pack 8: Change to behavior of IBM WebSphere MQ Managed File Transfer `fteCleanAgent` command

From IBM WebSphere MQ Version 7.5.0, Fix Pack 8, the `fteCleanAgent` command has been updated so that you must specify which IBM WebSphere MQ Managed File Transfer state to clear by passing the appropriate parameters to the command, as well as providing an agent name. This change in behavior ensures that, by default, `fteCleanAgent` does not clear all in-progress and pending transfers, resource monitor definitions and scheduled transfer definitions for the agent specified.

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

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

Related concepts:

[“What's new in IBM WebSphere MQ Version 7.5” on page 12](#)

[Learn about the main new functions in IBM WebSphere MQ Version 7.5.](#)

[“What's changed in IBM WebSphere MQ Version 7.5” on page 17](#)

[Review the list of changes carefully before upgrading queue managers to IBM WebSphere MQ Version 7.5. Decide whether you must plan to make changes to existing applications, scripts, and procedures before starting to migrate systems to Version 7.5.](#)

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](#)

Mappings between earlier IBM WebSphere MQ publications and the current information structure

The information in the knowledge center is structured according to a set of generic activities. For example installing, developing, administering, configuring, securing. Earlier publications (the "MQ books") were structured somewhat differently. This section provides a mapping between these earlier IBM WebSphere MQ publications, and the current information structure.

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 7.5
- Checking requirements
- Preparing the system
- Installing IBM WebSphere MQ server on AIX
- Verifying a server installation
- Installing a IBM WebSphere MQ client on AIX systems
- Verifying a client installation
- AIX: Applying maintenance level upgrades on IBM WebSphere MQ Version 7.5
- Uninstalling IBM WebSphere MQ on AIX

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 7.5
- Checking requirements
- Preparing the system
- Installing IBM WebSphere MQ server on HP-UX
- Verifying a server installation
- Installing a IBM WebSphere MQ client on HP-UX systems
- Verifying a client installation
- HP-UX: Applying maintenance level updates on IBM WebSphere MQ Version 7.5
- Uninstalling IBM WebSphere MQ on HP-UX

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 7.5
- Checking requirements
- Preparing the system
- Installing IBM WebSphere MQ server on Linux
- Verifying a server installation
- Installing WebSphere MQ client on Linux
- Verifying a client installation
- Linux: Applying maintenance level updates on IBM WebSphere MQ Version 7.5
- Uninstalling IBM WebSphere MQ on Linux

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 7.5
- Checking requirements
- Preparing the system
- Installing IBM WebSphere MQ server on Solaris
- Verifying a server installation
- Installing a IBM WebSphere MQ client on Solaris
- Verifying a client installation
- Solaris: Applying maintenance level updates on IBM WebSphere MQ Version 7.5
- Uninstalling IBM WebSphere MQ on Solaris

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 7.5
- Checking requirements
- Preparing the system
- Installing IBM WebSphere MQ server on Windows
- Verifying a server installation
- Installing a IBM WebSphere MQ client on Windows systems
- Verifying a client installation
- Windows: Applying maintenance level upgrades on IBM WebSphere MQ Version 7.5
- Uninstalling IBM WebSphere MQ on Solaris

Application programming guide

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

- Application development concepts
- Designing IBM WebSphere MQ applications
- Writing a queuing application
- Sample WebSphere MQ programs
- C language examples
- COBOL examples
- System/390[®] assembler-language examplesSystem/390 assembler-language examples
- WebSphere MQ data definition files
- Coding standards on 64-bit platforms

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
- Function calls
- Attributes of objects
- Return codes
- Rules for validating MQI options
- Report options and message flags
- Report options and message flags
- Data conversion
- Properties specified as MQRFH2 elements
- Code page conversion

Clients

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

- “Overview of IBM WebSphere MQ MQI clients” on page 121
- “Platform support for WebSphere MQ clients” on page 123
- Installing a IBM WebSphere MQ client
- Configuring connections between the server and client
- Configuring an extended transactional client
- Verifying a client installation
- Setting up WebSphere MQ MQI client security
- “Channels” on page 107
- MQI client: Default behavior of client-connection and server-connection
- Defining MQI channels
- Creating server-connection and client-connection definitions on different platforms
- Creating server-connection and client-connection definitions on the server
- Channel-exit programs for MQI channels
- Connecting a client to a queue-sharing group
- Configuring a client using a configuration file
- Using WebSphere MQ environment variables
- Using the message queue interface (MQI) in a client application
- Building applications for WebSphere MQ MQI clients
- Running applications in the WebSphere MQ MQI client environment
- Preparing and running CICS® and Tuxedo applications
- Preparing and running Microsoft Transaction Server applications
- Preparing and running WebSphere MQ JMS applications
- Resolving problems with IBM WebSphere MQ MQI clients
- Referencing connection definitions using a preconnect exit from a repository

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
- Constants

Intercommunication

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

Introduction

- “Concepts of intercommunication” on page 39
- Connecting applications using distributed queuing
- Networks and Network Planning
- WebSphere MQ distributed-messaging techniques
- Introduction to distributed queue management
- Channel attributes
- Example configuration information

Windows

UNIX

Linux

Distributed queue management in WebSphere MQ for Windows and UNIX platforms

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

Further intercommunication considerations

- Channel-exit programs for messaging channels
- Channel-exit calls and data structures
- Queue name resolution

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
- API completion and reason codes
- PCF reason codes
- Secure Sockets Layer (SSL) and Transport Layer Security (TLS) return codes
- WCF custom channel exceptions

Migration

The structure of the migration section in Version 7.5 remains the same as in Version 7.1.

Topics have been added for migration to Version 7.5, and removed for migration to Version 7.1. To refer to migration to Version 7.1 and earlier releases, refer to *Where to find a topic about a specific migration path*.

Monitoring

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

- Event monitoring
- Message monitoring
- Accounting and statistics messages
- Real-time monitoring
- Structure data types
- Object attributes for event data

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
- Introduction to the WebSphere MQ Administration Interface (MQAI)

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
- Distributed publish/subscribe
- Writing publish/subscribe applications
- Publish/subscribe security
- Publish/Subscribe migration from Version 6.0
- Migration of the publish/subscribe broker in WebSphere Event Broker and WebSphere Message Broker

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 54
- Configuring a queue manager cluster
- Managing IBM WebSphere MQ clusters
- Routing messages to and from clusters
- Using clusters for workload management
- Keeping clusters secure
- Working with the MQI and clusters
- WebSphere MQ cluster commands
- Resolving problems with queue manager clusters

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
- Generic values and characters with special meanings
- Building command scripts
- “Rules for naming IBM WebSphere MQ objects” on page 113
- Syntax diagrams
- The MQSC commands

Security

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

- Security

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 31
- Administering IBM WebSphere MQ
- Administering local WebSphere MQ objects
- Administration using the IBM WebSphere MQ Explorer
- Using the WebSphere MQ Taskbar application (Windows only)
- WebSphere MQ Control commands

Configuration and management

- Changing IBM WebSphere MQ and queue manager configuration information
- Planning file system support
- Setting up security on Windows, UNIX and Linux systems
- Transactional support
- Handling undelivered messages with the WebSphere MQ dead-letter queue handler
- Availability, recovery and restart
- Troubleshooting and support
- WebSphere MQ and UNIX System V IPC resources

- WebSphere MQ and UNIX Process Priority
- User exits, API exits, and WebSphere MQ installable services

Using .NET

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

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

Using C++

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

- Using C++
- WebSphere MQ C++ classes

Using Java

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

- Should I use WebSphere MQ classes for Java or WebSphere MQ classes for JMS?
- Using WebSphere MQ classes for Java
- WebSphere MQ classes for JMS
- Using WebSphere MQ classes for JMS
- 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 bridge for HTTP

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)

IBM WebSphere MQ Version 7.5, IBM i and z/OS

IBM WebSphere MQ Version 7.5 is not available for IBM i and z/OS.

These platforms are available in later versions of the product.

For information about the latest versions of IBM MQ for IBM i and z/OS, see the IBM MQ website.

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

Related concepts:

“Introduction to IBM WebSphere MQ” on page 1

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

Related reference:

“Main features and benefits of message queuing” on page 33

This information highlights some features and benefits of message queuing. It describes features such as security and data integrity of message queuing.

Related information:

Designing a WebSphere MQ architecture

WebSphere MQ Managed File Transfer

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 35). The physical nature of a queue depends on the operating system on which the queue manager is running. A queue can either be a volatile buffer area in the memory of a computer, or a data set on a permanent storage device (such as a disk). The physical management of queues is the responsibility of the queue manager and is not made apparent to the participating application programs.

Programs access queues only through the external services of the queue manager. They can open a queue, put messages on it, get messages from it, and close the queue. They can also set, and inquire about, the attributes of queues.

Different styles of message queuing

Point-to-point

One message is placed on the queue and one application receives that message.

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

Publish/Subscribe

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

Publish/subscribe messaging allows you to decouple the provider of information from the consumers of that information. The sending application and receiving application do not need to know as much about each other for the information to be sent and received. For more information about publish/subscribe messaging, see Introduction to WebSphere MQ publish/subscribe messaging

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.

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.

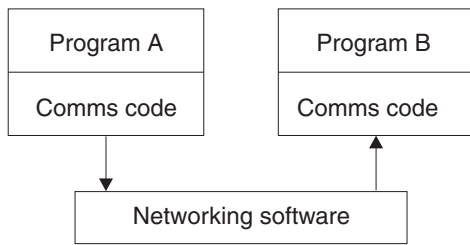
No direct connections between programs

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 4 on page 34).

Traditional communication between programs



Communication by message queuing

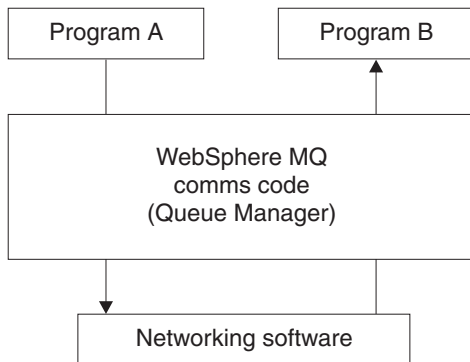


Figure 4. Message queuing compared with traditional communication

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.

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

- Topic
- Subscription

Message

In message queuing, a message is a collection of data sent by one program and intended for another program. See IBM WebSphere MQ messages. For information about message types, see *Types of message*.

Message descriptor

A IBM 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 IBM 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 IBM 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. IBM 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* 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 in IBM WebSphere MQ Version 7.0 and later versions.

In a IBM 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 54.

IBM WebSphere MQ MQI client

IBM WebSphere MQ MQI *clients* are independently installable components of IBM WebSphere MQ. An MQI client allows you to run IBM 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 IBM WebSphere MQ MQI client components, see *Installing a IBM WebSphere MQ MQI client and Configuring connections between the server and client*.

Point-to-point messaging

In point-to-point messaging, each message travels from one producing application to one consuming application. Messages are transferred through the producing application putting messages onto a queue and the consuming application gets them from that queue.

Publish/subscribe messaging

In publish/subscribe messaging, a copy of each message published by a publishing application is delivered to every interested application. There might be many, one or no interested applications. In publish/subscribe an interested application is known as a subscriber and the messages are queued on a queue identified by a subscription. For more information about publish/subscribe, see *Introduction to IBM WebSphere MQ publish/subscribe messaging*.

Topic

A topic is a character string that describes the subject of the information that is published in a publish/subscribe message.

Topics are key to the successful delivery of messages in a publish/subscribe system. Instead of including a specific destination address in each message, a publisher assigns a topic to each message. The queue manager matches the topic with a list of subscribers who have subscribed to that topic, and delivers the message to each of those subscribers.

Subscription

A publish/subscribe application can register an interest in messages about specific topics. When an application does this it is known as a subscriber and the term subscription defines how matching messages are queued for processing.

A subscription contains information about the identity of the subscriber and the identity of the destination queue on to which publications are to be placed. It also contains information about how a publication is to be placed on the destination queue.

Messages and queues

Messages and queues are the basic components of a message queuing system.

What is a message?

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

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.

Message lengths

The default maximum message length is 4 MB, although you can increase this to a maximum length of 100 MB (where 1 MB equals 1 048 576 bytes). In practice, the message length might be limited by:

- The maximum message length defined for the receiving queue
- The maximum message length defined for the queue manager
- The maximum message length defined by the queue
- The maximum message length defined by either the sending or receiving application
- The amount of storage available for the message

It might take several messages to send all the information that an application requires.

How do applications send and receive messages?

Application programs send and receive messages using **MQI calls** .

For example, to put a message onto a queue, an application:

1. Opens the required queue by issuing an MQI **MQOPEN** call
2. Issues an MQI **MQPUT** call to put the message onto the queue

Another application can retrieve the message from the same queue by issuing an MQI **MQGET** call

For more information about MQI calls, see MQI calls.

What is a queue?

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

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

Predefined queues and dynamic queues

Queues can be characterized by the way they are created:

- **Predefined queues** are created by an administrator using the appropriate MQSC or PCF commands. Predefined queues are permanent; they exist independently of the applications that use them and survive IBM 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 5 on page 40 shows an overview of the components of distributed queuing.

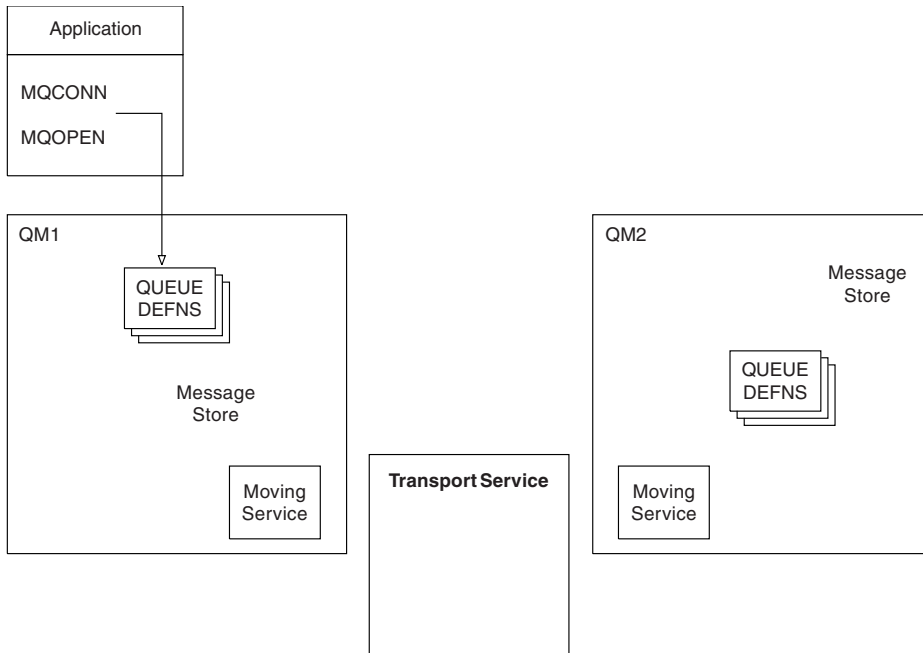


Figure 5. Overview of the components of distributed queuing

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

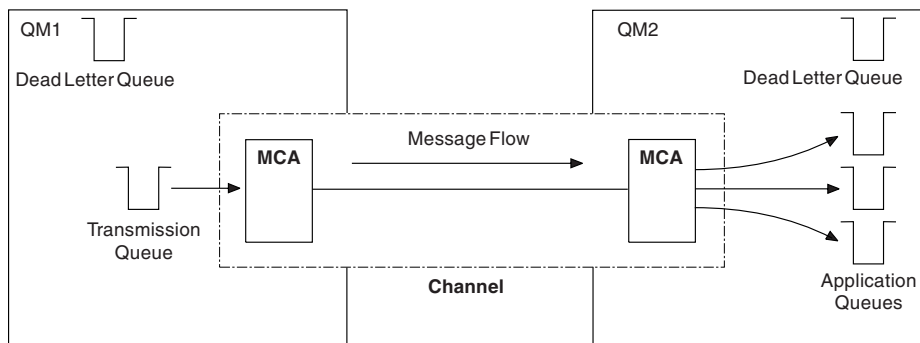


Figure 6. Sending messages

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 7 on page 42.

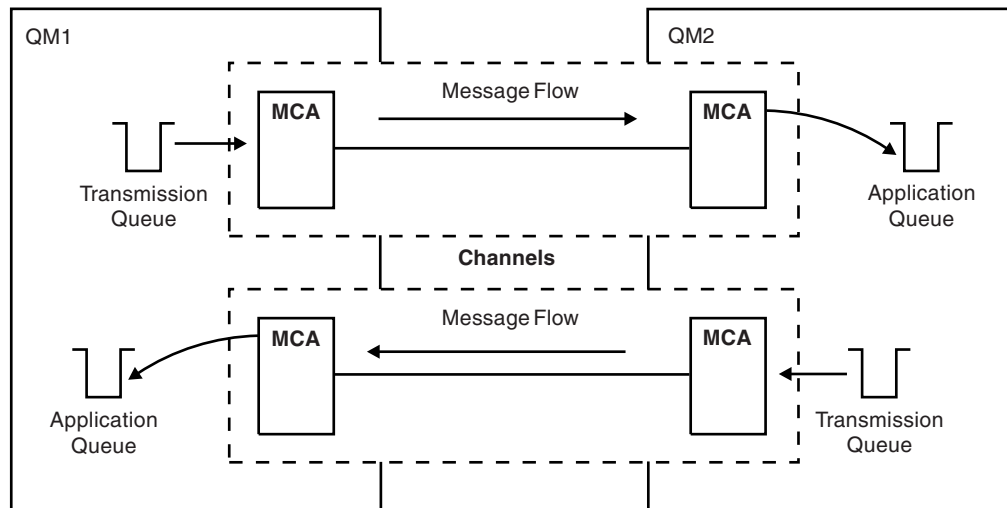


Figure 7. Sending messages in both directions

For more information about Distributed Queue Management, see Introduction to distributed queue management.

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 8 on page 43 shows the components of a cluster called CLUSTER:

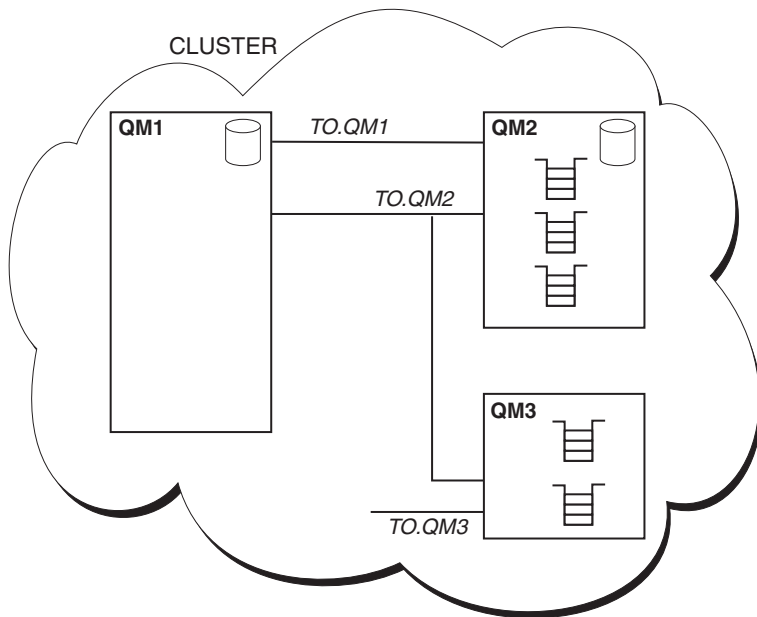


Figure 8. A cluster of queue managers

- 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 29.

Related concepts:

“Distributed queuing components” on page 44

These are objects you need for enabling intercommunication.

“Dead-letter queues” on page 47

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.

“Remote queue definitions” on page 47

Remote queue definitions are definitions for queues that are owned by another queue manager.

“How to get to the remote queue manager” on page 47

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.

“Addressing information” on page 49

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*.

“What are aliases?” on page 50

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 alias definitions” on page 51

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.

“Reply-to queue alias definitions” on page 52

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.

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 (not applicable to z/OS). For examples of the parameters needed to set up sender-receiver channels, see Example configuration information applicable to your platform. For the parameters needed to define a channel of any type, see DEFINE CHANNEL.

Sender-receiver channels

A sender in one system starts the channel so that it can send messages to the other system. The sender requests the receiver at the other end of the channel to start. The sender sends messages from its transmission queue to the receiver. The receiver puts the messages on the destination queue. Figure 9 illustrates this.

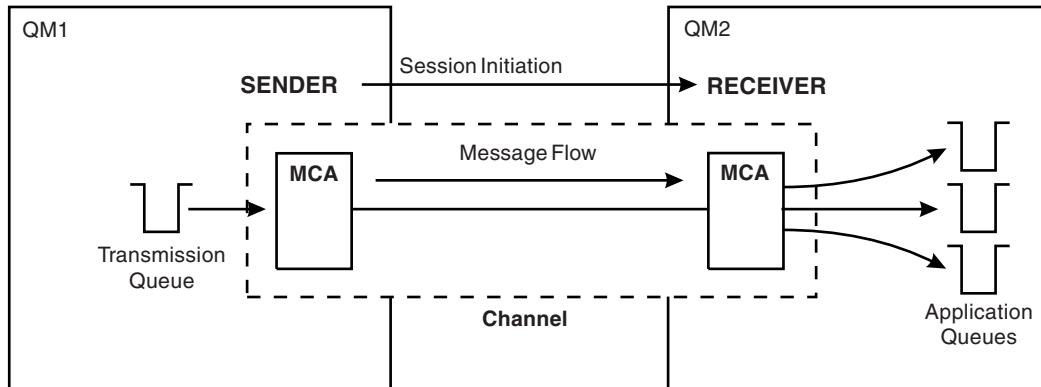


Figure 9. A sender-receiver channel

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.

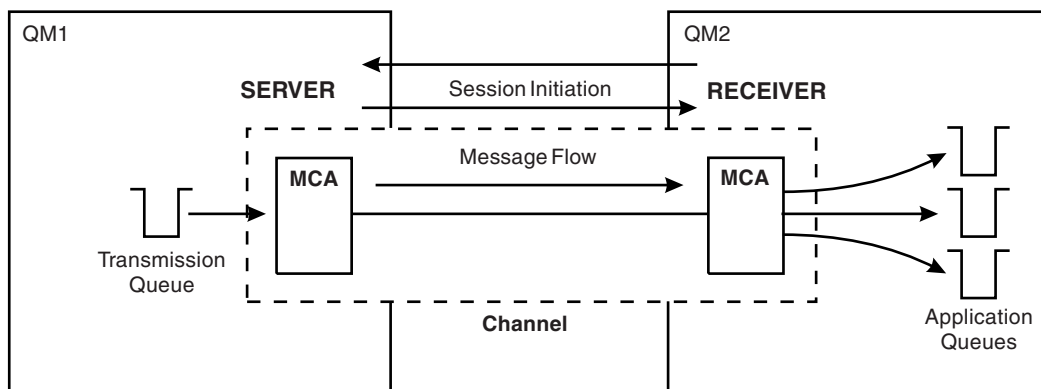


Figure 10. A requester-server channel

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.

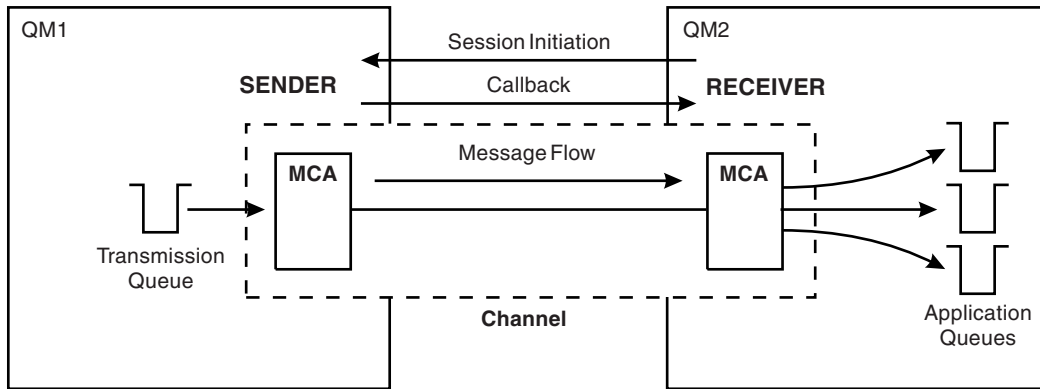


Figure 11. A requester-sender channel

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 9 on page 45.

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.

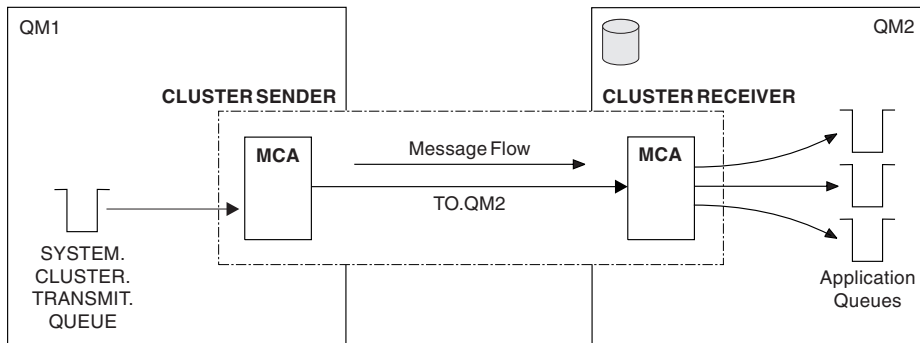


Figure 12. A cluster-sender channel

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 12.

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 USEDQL 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 USEDQL channel attribute on different MQSC commands, see `DEFINE CHANNEL`, `DISPLAY CHANNEL`, `ALTER CHANNEL`, and `DISPLAY CLUSQMGR`.

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 13 on page 48.

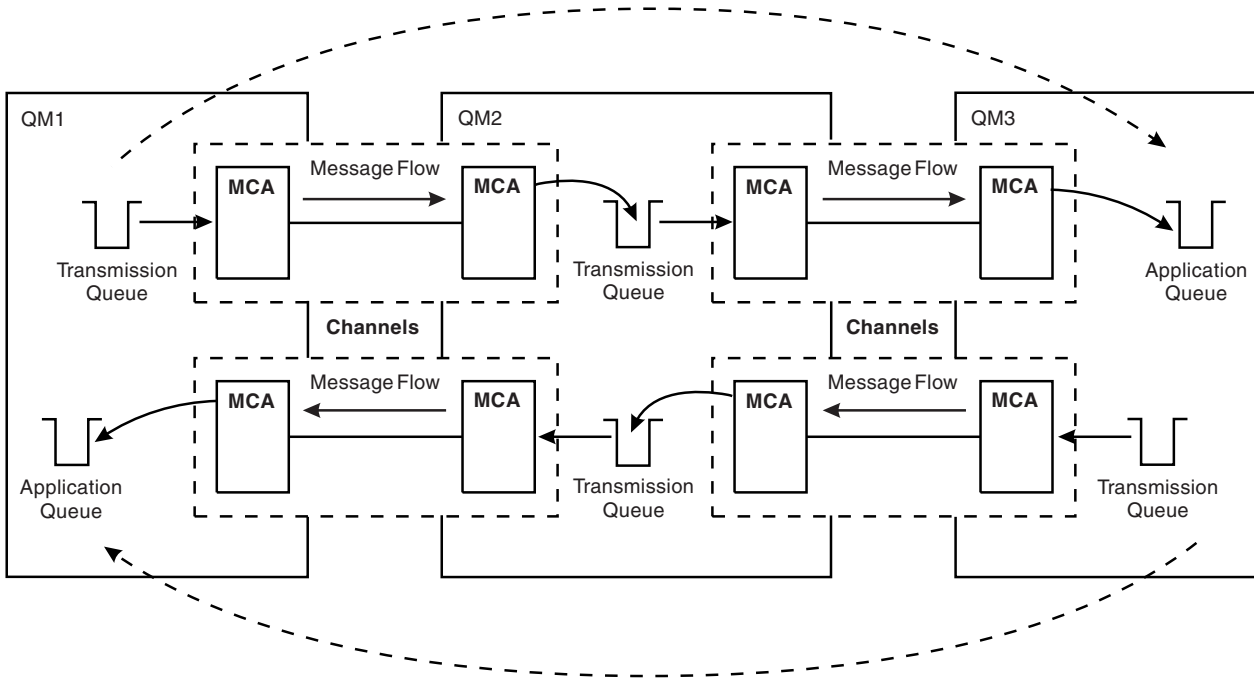


Figure 13. Passing through intermediate queue managers

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 14.

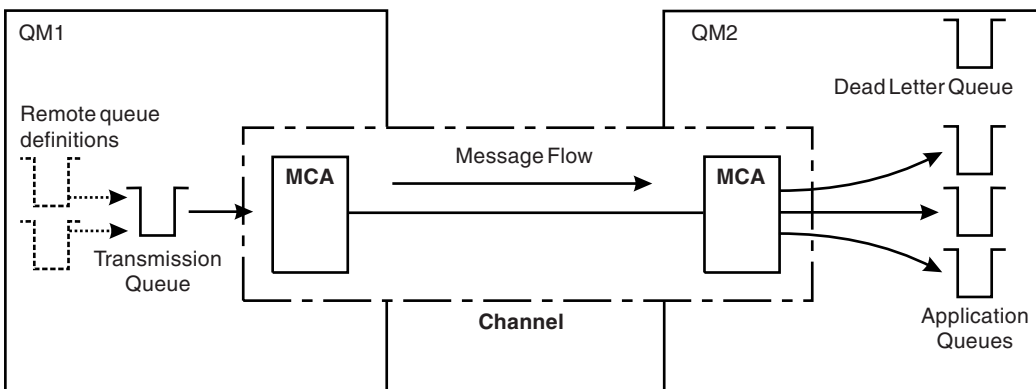


Figure 14. Sharing a transmission queue

Figure 14 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 15.

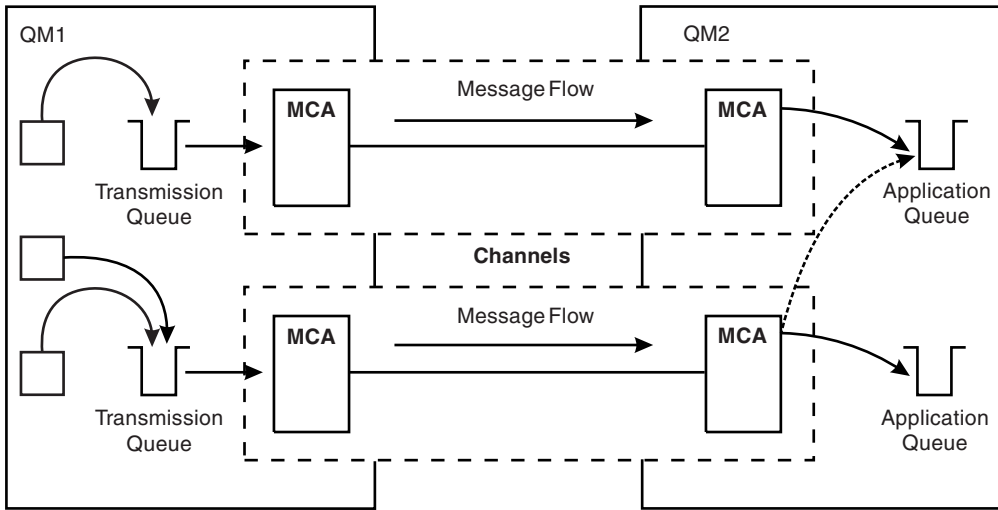


Figure 15. Using multiple channels

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 117.

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](#).

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 16 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.

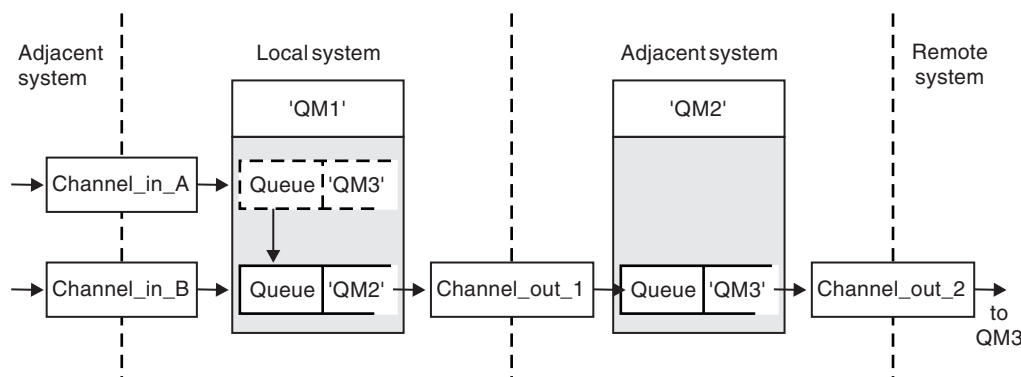


Figure 16. Queue manager alias

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.

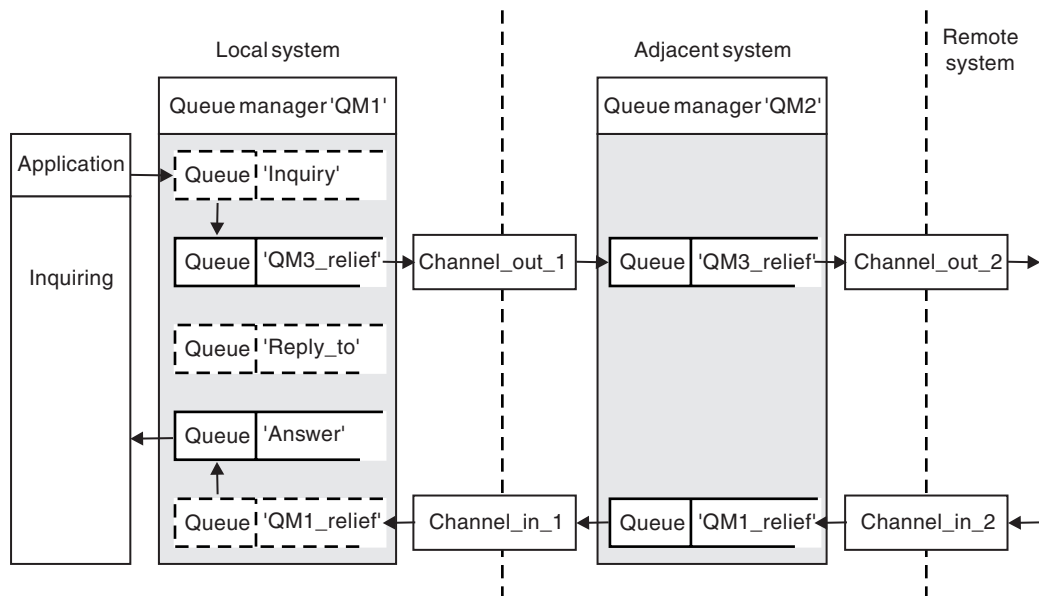


Figure 17. Reply-to queue alias used for changing reply location

In the example in Figure 17:

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](#).

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. You are not limited to using a single transmission queue. A queue manager can use multiple transmission queues to separate the messages going to each queue manager in a cluster. Typically, a queue manager uses a single cluster transmission queue. You can change the queue manager attribute DEFCLXQ, so that a queue manager uses a different cluster transmission queues for each queue manager in a cluster. You can also define cluster transmission queues manually.

All the queue managers that join a cluster agree to work in this way. They send out information about themselves and about the queues they host, and they receive information about the other members of the cluster.

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 18 on page 56 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.
-

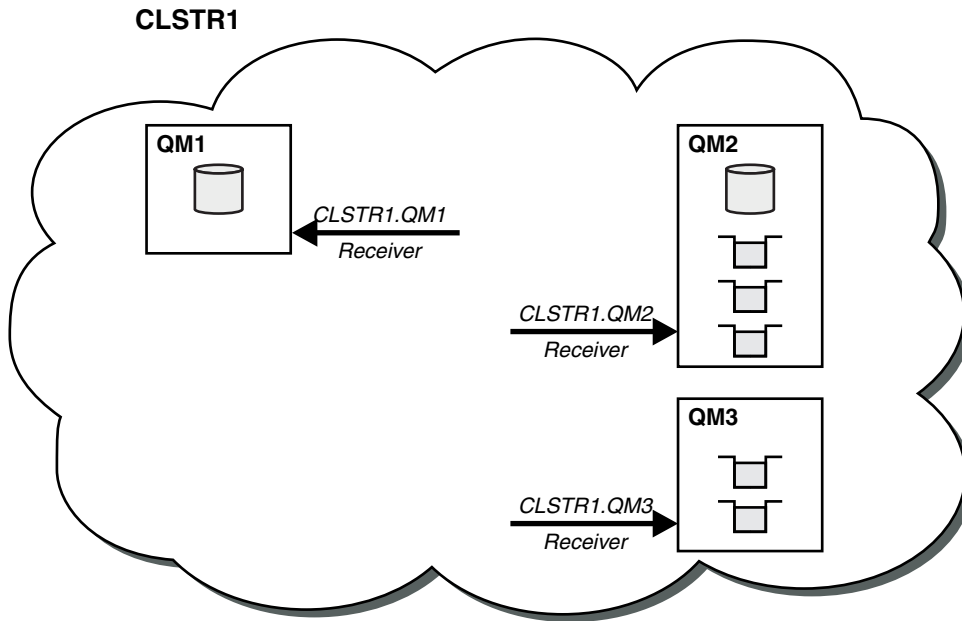


Figure 18. A cluster of queue managers

- In Figure 19 on page 57 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.

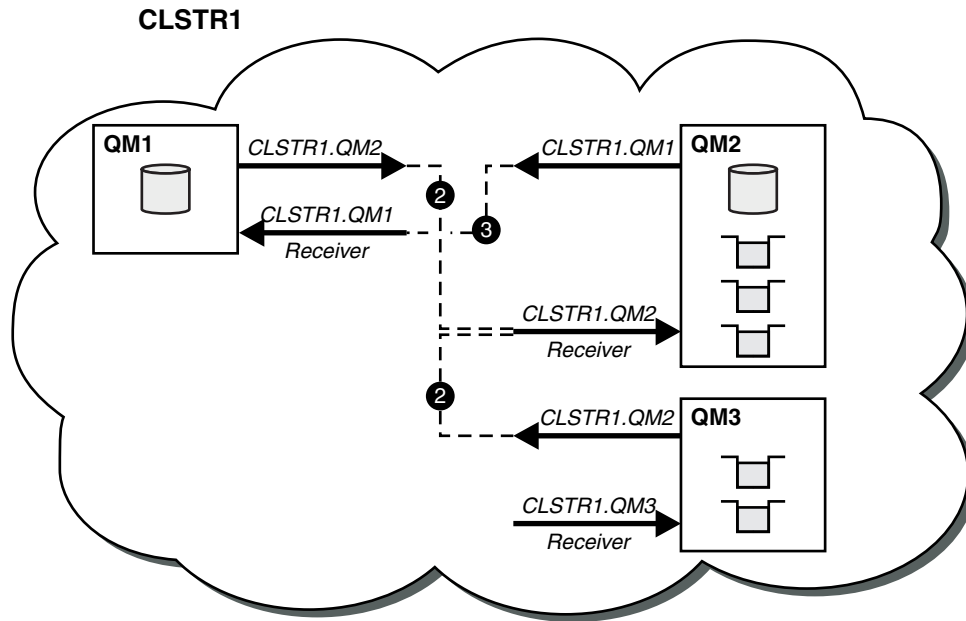


Figure 19. A cluster of queue managers with sender channels

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 18 on page 56. 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 20 on page 58 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.

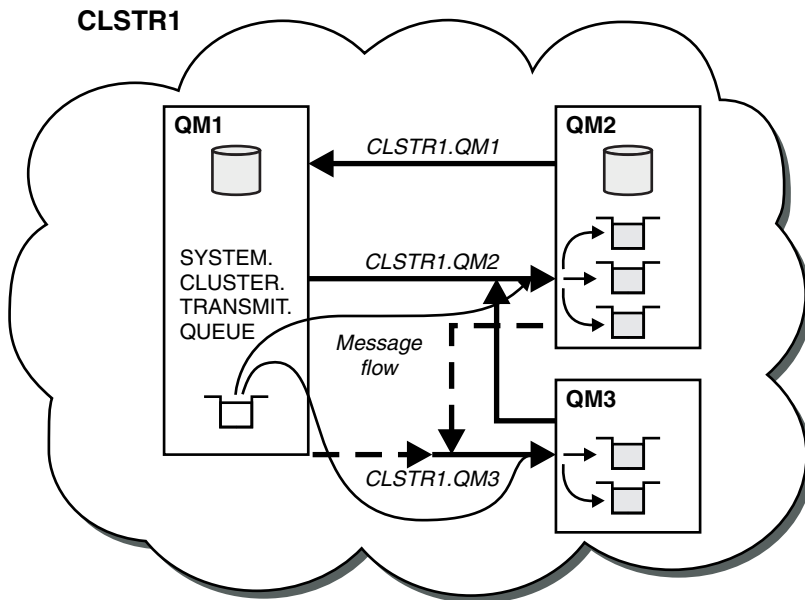


Figure 20. A cluster of queue managers, showing auto-defined channels

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

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.

Related information:

- Configuring a queue manager cluster
- Comparison of clustering and distributed queuing
- Components of a cluster
- Setting up a new cluster
- Managing WebSphere MQ clusters

IBM WebSphere MQ Telemetry

IBM WebSphere MQ Telemetry comprises a telemetry (MQXR) service that is part of a queue manager, telemetry clients that you can write yourself, or use one of the clients that are provided, and command line and explorer administrative interfaces. Telemetry refers to collecting data from and administering a wide range of remote devices. With IBM WebSphere MQ Telemetry you can integrate the collection of data and control of devices with web applications.

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 7.5 or

Migration of telemetry applications from using WebSphere Message Broker version 6 to use IBM WebSphere MQ Telemetry and WebSphere Message Broker version 7.0.

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 (MQXR) 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. 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.”

Related information:

Installing WebSphere MQ Telemetry

Administering WebSphere MQ Telemetry

Migration of telemetry applications from using WebSphere Message Broker version 6 to use WebSphere MQ Telemetry and WebSphere Message Broker version 7.0

Migrating WebSphere MQ Telemetry from version 7.0.1 to version 7.5

Developing applications for WebSphere MQ Telemetry

WebSphere MQ Telemetry Reference

Troubleshooting for WebSphere MQ Telemetry


Telemetry concepts and scenarios for monitoring and control

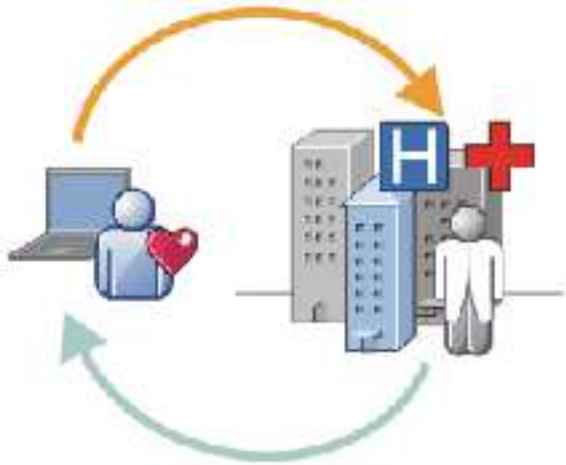
Telemetry is the automated sensing, measurement of data, and control of remote devices. The emphasis is on the transmission of data from devices to a central control point. Telemetry also includes sending configuration and control information to devices.


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	
	<ul style="list-style-type: none"> • 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 64

Telemetry: Smart Health Services	
<ul style="list-style-type: none"> • 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 62 	

Telemetry: One in a Crowd	
	<ul style="list-style-type: none"> • 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”

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.

“Telemetry scenario: Home energy monitoring and control” on page 64

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.

“Telemetry scenarios: Radio Frequency Identification (RFID)” on page 65

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.

“Telemetry scenarios: Environment sensing” on page 66

Environment sensing uses telemetry to collect information about river water levels and quality, atmospheric pollutants, and other environmental data.

“Telemetry scenarios: Mobile applications” on page 67

Mobile applications are applications that run on wireless devices. The devices are either generic application platforms or custom devices.

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 and Java 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 patient and the MQTT device, a

short-range low-power RF network is used. The transmitter connects to the hospital using a VPN TCP/IP connection over a low-bandwidth phone-line.

It is often impractical to find a way to connect every device directly to an Internet Protocol network. Using two networks, connected by a hub, is a common solution. The MQTT device is a simple hub, storing information from the patient, and forwarding it to the hospital.

Security

The physician must be able to trust the authenticity of the patient data, and the patient wants the privacy of their data to be respected.

In some 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 be 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 appliances, and the meter, use a home area network, such as Zigbee. MQTT for sensor networks (MQTT-S), is designed to work with Zigbee and other low bandwidth network protocols. 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 losing data.

Scalability

If the RFID application requires immediate responses, in the order of a second or two, then the RFID readers must stay connected.

Telemetry 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.

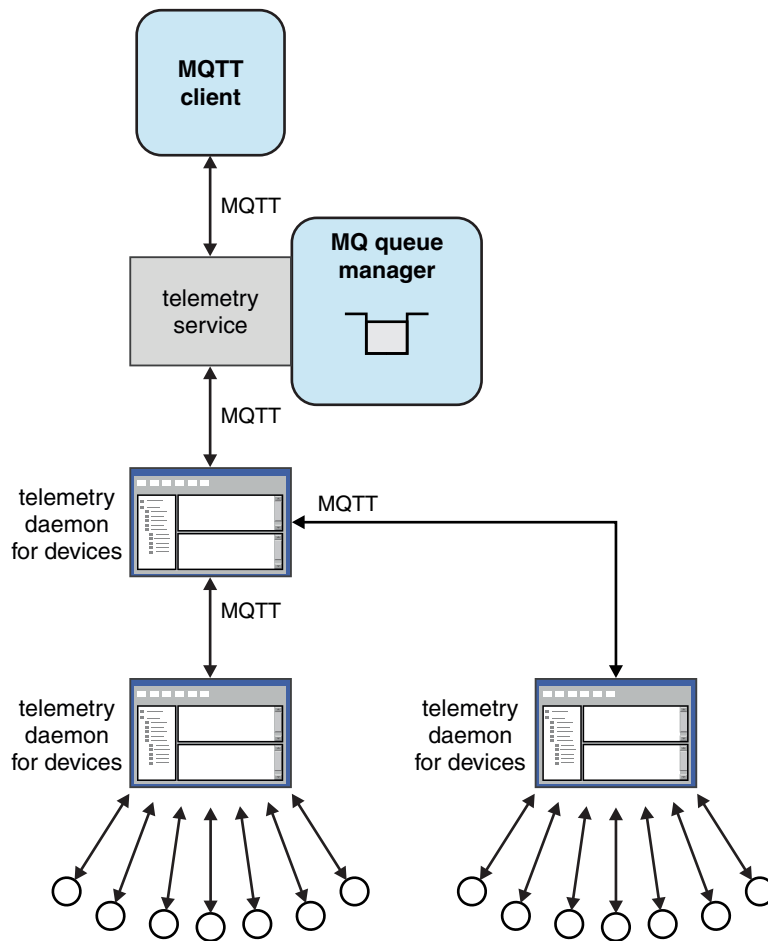
Scalability

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 (MQXR) 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 (MQXR) 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 (MQXR) service. Each queue manager runs a maximum of one telemetry (MQXR) service.

The telemetry (MQXR) service uses the remote socket address set by each client in the `MqttClient.connect` method to allocate the connection to a telemetry channel. A socket address is the combination of TCP/IP host name and port number. Multiple clients that use the same remote socket address are connected to the same telemetry channel by the telemetry (MQXR) service.

If there are multiple queue managers on a server, split the telemetry channels between the queue managers. Allocate the remote socket addresses between the queue managers. Define each telemetry channel with a unique remote socket address. Two telemetry channels must not use the same socket address.

If the same remote socket address is configured for telemetry channels on multiple queue managers, the first telemetry channel to connect, wins. Subsequent channels connecting on the same address fail, 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](#) and [Configuring a queue manager for telemetry on Windows](#) to configure telemetry manually.

Related information:

MQXR properties

Telemetry connection protocols

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

Telemetry (MQXR) service

The telemetry (MQXR) 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 (MQXR) 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](#).

The telemetry (MQXR) 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 IBM WebSphere MQ Telemetry supplement for IBM WebSphere MQ Explorer. Configure a channel, using the wizard, to accept connections from MQTT clients on a particular TCP/IP port. Since Version 7.1, you can configure IBM WebSphere MQ Telemetry using the command line program, **runmqsc**.

Create multiple telemetry channels, on different ports, to make large numbers of client connections easier to manage, by splitting the clients into groups. Each telemetry channel has a different name.

You can configure telemetry channels with different security attributes to create different types of connection. Create multiple channels to accept client connections on different TCP/IP addresses. Use SSL to encrypt messages and authenticate the telemetry channel and client; see [SSL configuration of MQTT clients and telemetry channels](#). 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](#).

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. Version 3 of the protocol uses publish/subscribe, and supports three qualities of service: "fire and forget", "at least once", and "exactly once".

The small size of the protocol headers, and the byte array message payload, keeps messages small. The headers comprise a 2 byte fixed header, and up to 12 bytes of additional variable headers. The protocol uses 12 byte variable headers to subscribe and connect, and only 2 byte variable headers for most publications.

With three qualities of service, you can trade off between low-latency and reliability; see Qualities of service provided by an MQTT client. "Fire and forget" uses no persistent device storage, and only one transmission to send or receive a publication. "At least once", and "exactly once" require persistent storage on the device to maintain the protocol state and save a message until it is acknowledged.

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.

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. See PubSync.java.

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. See Configure distributed queuing to send messages to MQTT clients.

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 73 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 21 on page 72, the MQTT client subscribes using the topic string "MQTT Examples". A WebSphere MQ C application, Figure 22 on page 72, publishes to the topic using the topic string "MQTT Examples". In the code snippet Figure 23 on page 72, 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 77 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 (MQXR) 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 24 on page 73 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 78, 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 21 on page 72 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](#).

```
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 21. MQTT v3 client subscriber

The code snippet in Figure 22 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

```
/* 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 22. 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.

```
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 23. messageArrived method

Figure 24 on page 73 shows an MQTT v3 publishing a message to the subscription created in Figure 21. The code snippet is extracted from the task, Creating your first MQ Telemetry Transport publisher application using Java.

```

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 24. 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 (MQXR) service yet defined” on page 74.
- 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 (MQXR) service yet defined” on page 74.
- You want to do the task using an existing queue manager that has no telemetry resources defined. You do not want to run the **Define sample configuration** wizard.
 1. Do one of the following tasks to set up telemetry:
 - Configuring a queue manager for telemetry on Linux and AIX
 - Configuring a queue manager for telemetry on Windows
 2. Do the task: “Start task with a running telemetry (MQXR) service” on page 74
- If you want to do the task using an existing queue manager that already has telemetry resources defined, do the task: “Start task with a running telemetry (MQXR) service” on page 74.

What to do next

Do “Sending a message to an MQTT client using WebSphere MQ Explorer” on page 77 to send a message directly to the client utility.

Start task with no telemetry (MQXR) service yet defined:

Create a queue manager and run the **Define sample configuration** to define sample telemetry resources for the queue manager. Publish a message using 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 76. 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, MQTQMGR.
2. Create and start the telemetry (MQXR) service and create a sample telemetry channel.
 - a. Open the Queue Managers*QmgrName*\Telemetry folder.
 - b. Click **Define sample configuration...** > **Finish**
Leave the **Launch MQTT Client Utility** check box checked.
3. Create a subscription for MQTT Example using the MQTT client utility.
 - a. Click **Connect**.
The **Client history** records a Connected event.
 - b. Type MQTT Example into the **Subscription\Topic** field > **Subscribe**.
The **Client history** records a Subscribed event.
4. Create MQTTExampleTopic in 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 (MQXR) service:

Create a telemetry channel and a topic. Authorize the user to use the topic and the telemetry transmit queue. Publish a message using 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 (MQXR) service is defined and running. The telemetry (MQXR) service might have been created manually, or by running the **Define sample configuration** wizard.

About this task

In this task you configure an existing queue manager to send a publication to the MQTT client utility.

Step 1 of the task sets the default transmission queue to `SYSTEM.MQTT.TRANSMIT.QUEUE`, which might interfere with applications running on an existing queue manager. It is possible, but laborious, to configure telemetry and not use the default transmission queue; do the follow on task: "Using a queue manager alias" on page 76.

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 (MQXR) service yet defined” on page 74 or the task, “Start task with a running telemetry (MQXR) service” on page 74.

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 73. Leave the MQTT client utility connected.

About this task

The task demonstrates sending a message to an MQTT client using queue rather than publishing to a topic. You do not create a subscription in the client. Step 2 of the task demonstrates that the previous subscription has been deleted.

Procedure

1. Discard any existing subscriptions by disconnecting and reconnecting the MQTT client utility.

The subscription is discarded because, unless you change the defaults, the MQTT client utility connects with a clean session; see Clean sessions.

To make it easier to do the task, type your own `ClientIdentifier`, rather than use the generated `ClientIdentifier` created by the MQTT client utility.

a. Click **Disconnect** to disconnect the MQTT client utility from the telemetry channel.

The **Client History** records a Disconnected event

b. Change the **Client Identifier** to MyClient.

c. Click **Connect**.

The **Client History** records a Connected event

2. Check that the MQTT client utility no longer receives publication for the MQTTExampleTopic.

a. Click the Queue Managers*QmgrName*\Topics folder in the 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 73. 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 80, 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 `Clean sessions`.

To make it easier to do the task, type your own `ClientIdentifier`, rather than use the generated `ClientIdentifier` created by the MQTT client utility.

- a. Click **Disconnect** to disconnect the MQTT client utility from the telemetry channel.

The **Client History** records a Disconnected event

- b. Change the **Client Identifier** to `MyClient`.
- c. Click **Connect**.

The **Client History** records a Connected event

2. Create a subscription to the topic, `MyClient`

`MyClient` is the `ClientIdentifier` of this client.

- a. Type `MyClient` into the **Subscription\Topic** field > **Subscribe**.

The **Client history** records a Subscribed event.

3. Start another MQTT client utility.

- a. Open the Queue Managers*QmgrName*\Telemetry\channels folder.
- b. Right-click the **PlainText** channel > **Run MQTT Client Utility...**
- c. Click **Connect**.

The **Client History** records a Connected event

4. Publish `Hello MyClient!` to the topic `MyClient`.

- a. Copy the subscription topic, `MyClient`, from the MQTT client utility running with the `ClientIdentifier`, `MyClient`.
- b. Paste `MyClient` into the **Publication\Topic** field of each of the MQTT client utility instances.
- c. Type `Hello MyClient!` into the **Publication\message** field.
- d. Click **Publish** in both instances.

Results

The **Client history** in the MQTT client utility with the `ClientIdentifier`, `MyClient`, records two **Received** events and one **Published** event. The other MQTT client utility instance records one **Published** event.

If 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 (MQXR) service.

What to do next

Do the next task, “Using the WebSphere MQ Telemetry daemon for devices as the publish/subscribe broker” on page 80. 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.

- a. Create a file called `amqtdd.cfg` in the same directory as the daemon.
- b. Add a line to the file to configure a different default port for the daemon.
port 1884
- c. 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, 2010 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.

- 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. Change the **Client Identifier** 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 receive 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 73. 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 74 of “Publishing a message to the MQTT client utility from WebSphere MQ Explorer” on page 73.
 - 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 73.

 - 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 78 and “Using the WebSphere MQ Telemetry daemon for devices as the publish/subscribe broker” on page 80 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 amqtdc.cfg, which is stored in the same directory as the daemon, amqtdc, 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 (MQXR) service, and then exchange messages with the daemon.

Tip: The update file, `amqtdc.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 `amqtdc.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, `amqtdc.upd`, in the same directory as the daemon, `amqtdc`, 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 (MQXR) 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 25 shows the resulting console log.

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

Figure 25. 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 (MQXR) service forwards the publications to the client that created the subscription. It uses `ClientIdentifier`, which has been set as the remote queue manager name to identify the client.

Typically, `SYSTEM.MQTT.TRANSMIT.QUEUE` must be defined as the default transmission queue. It is possible, but onerous, to configure MQTT not to use the default transmission queue; see *Configure distributed queuing to send messages to MQTT clients*.

An MQTT client can create a persistent session; see “MQTT stateless and stateful sessions” on page 88. Subscriptions created in a persistent session are durable. Publications that arrive for a client with a persistent session are stored in `SYSTEM.MQTT.TRANSMIT.QUEUE`, and forwarded to the client when it reconnects.

An MQTT client can also publish and subscribe to retained publications; see *Retained publications and MQTT clients*. A subscriber to a retained publication topic receives the latest publication to the topic. The subscriber receives the retained publication when it creates a subscription, or when it reconnects to its earlier session.

Telemetry applications

Write telemetry applications using 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 (MQXR) 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 `json-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 85.

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 WebSphere Message Broker version 6 to use IBM WebSphere MQ Telemetry and WebSphere Message Broker version 7.0* 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 86 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 (MQXR) 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 3.

Table 3. 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 (MQXR) 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.

RFH2 values are set as shown in Table 4. The Format property is set in the RFH2 fixed header and the other values are set in RFH2 folders.

Table 4. RFH2

RFH2 property	Type/Folder	Header
Format	MQCHAR8	MQFMT_NONE
ClientIdentifier	mqtt/clientId	Copy MqttClient.ClientIdentifier with a length of 1...23 bytes.
QoS	mqtt/qos	Copy QoS from incoming MQTT message.
Message ID	mqtt/msgid	Copy Message ID from incoming MQTT message, if QoS is 1 or 2.
MQIsRetained	mqs/Ret	Set if the original MQTT publication was sent with the RETAIN property set and the message is received as a retained publication.
MQTopicString	mqs/Top	The topic to which the MQTT message was published.

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

Table 5. 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 6 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 (MQXR) service.

Table 6. MQTT fixed header properties

MQTT field	Type	Value
DUP	boolean	Set if QoS = 1 or 2, and the message was sent to this client in a previous transmission, and the message has not been acknowledged after a time.
QoS	int	<p>The way the value of QoS in an outgoing publication from the publish/subscribe broker in 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.</p> <p>MQTT Lower value of the QoS in the incoming publication, and in the QoS requested by the subscriber.</p> <p>WebSphere MQ Lower value of the QoS derived from the incoming publication: MQPER_NOT_PERSISTENT→QoS=0 MQPER_PERSISTENT→QoS=2</p> <p>and the QoS requested by the subscriber. If the message is sent to the client without a subscription, QoS is set by default to 2. A client can alter this value by subscribing to DEFAULT.QoS with a different QoS.</p>
RETAIN	boolean	Set if the incoming publication has the retained property set.

Table 7 describes how the variable message headers are set in the MQTT message that is sent to the MQTT client.

Table 7. 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.

Concentrate multiple MQTT clients into one connection to the telemetry server.

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 (MQXR) 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 Clean sessions.

If the client resumes an existing session, it continues as if the connection had not been broken. Publications awaiting delivery are sent to the client, and any message transfers that had not been committed, are completed. When a client in a persistent session disconnects from the telemetry (MQXR) service, any subscriptions the client created remain. Publications for the subscriptions are sent to the client when it reconnects. If it reconnects without resuming the old session, the publications are discarded by the telemetry (MQXR) service.

Session state information is saved by the queue manager in the `SYSTEM.MQTT.PERSISTENT.STATE` queue.

The 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 Last will and testament publication. It is sent by the telemetry (MQXR) service, if it detects the connection to the client has broken without the client requesting it.

A client can publish a retained publication at any time; see Retained publications and MQTT clients. A new subscription to a topic can request to be sent any retained publication associated with topic. If you create the last will and testament as a retained publication, you can use it to monitor the status of a client.

For example, the client publishes a retained publication, when it connects, advertising its availability. At the same time, it creates a retained last will and testament publication that announces its unavailability. In addition, just before it makes a planned disconnection, it publishes its unavailability as a retained publication. To find out whether the client is available, you would subscribe to the topic of the retained publication. You would always receive one of the three publications.

If the client is to receive messages published when it is disconnected, then reconnect the client to its previous session; see “MQTT stateless and stateful sessions.” 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 Delivery tokens.

After passing a message to the MQTT client, the application has the choice of waiting on the delivery-token. Rather than waiting, the client can provide a callback method that is called when the publication is delivered to 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 (MQXR) 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. You can also use SSL to check that the client device is permitted to connect to the server; see MQTT client authentication using SSL.

JAAS is principally used to check that the user of the device is permitted to use a server application; see MQTT client authentication using a password. JAAS can be used with LDAP to check a password using a single sign-on directory.

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.

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.getBytes` method that converts a string into a byte array encoded using the local platform default character set. Two MQTT Java programs exchanging text in the message payload, between platforms with the same default character set do so easily and efficiently in UTF-8.

If the default character set of one of the platforms is not UTF-8, then the applications must establish a convention for exchanging messages. For example, the publisher specifies conversion from a string to UTF-8 using the `getBytes("UTF8")` method. To receive the text of a message, the subscriber assumes that the message is encoded in the UTF-8 character set.

The telemetry (MQXR) service describes the encoding of all incoming publications from MQTT clients messages as being UTF-8. It sets `MQMD.CodedCharSetId` to UTF-8, and `RFH2.CodedCharSetId` to `MQCCSI_INHERIT`; see "Integration of WebSphere MQ Telemetry with queue managers" on page 85. 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 (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 `Client identifier`.

Throughput

The length of topic names affects the number of bytes that flow across the network. When publishing or subscribing, the number of bytes in a message might be important. Therefore limit the number of characters in a topic name. When an MQTT client subscribes for a topic 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`.

Improving scalability by avoiding subscriptions.

When an MQTT V3 client subscribes to a topic, a subscription is created by the telemetry (MQXR) 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 (MQXR) service by setting the JVM parameters `-Xms` and `-Xmx`. Follow these steps:

1. Find the `java.properties` file in the telemetry service configuration directory; see [Telemetry \(MQXR\) service configuration directory on Windows](#) or [Telemetry service configuration directory on Linux](#).
2. Follow the directions in the file; a heap of 1 GB is sufficient for 50,000 concurrently connected clients.

```
# Heap sizing options - uncomment the following lines to set the heap to 1G
#-Xmx1024m
#-Xms1024m
```
3. Add other command-line arguments to pass to the JVM running the telemetry (MQXR) service in the `java.properties` file; see [Passing JVM parameters to the telemetry \(MQXR\) service](#).

To increase the number of open file descriptors on Linux, add the following lines to `/etc/security/limits.conf/`, and log in again.

```
@mqm soft nofile 65000
@mqm hard nofile 65000
```

Each socket requires one file descriptor. The telemetry service require some additional file descriptors, so this number must be larger than the number of open sockets required.

The queue manager uses an object handle for each nondurable subscription. To support many active, nondurable subscriptions increase the maximum number of active handles in the queue manager; for example:

```
echo ALTER QMGR MAXHANDS(999999999) | runmqsc qMgrName
```

Figure 26. Alter maximum number of handles on Windows

```
echo "ALTER QMGR MAXHANDS(999999999)" | runmqsc qMgrName
```

Figure 27. Alter maximum number of handles on Linux

Other considerations

When planning your system requirements, consider the length of time taken to restart the system. The planned downtime might have implications for the number of messages that queue up, waiting to be processed. Configure the system so that the messages can be successfully processed in an acceptable time. Review disk storage, memory, and processing power. With some client applications, it might be possible to discard messages when the client reconnects. To discard messages, set `CleanSession` in the client connection parameters; see [Clean sessions](#). Alternatively, publish and subscribe using the best effort Quality of Service, 0, in an MQTT client; see [Quality of service](#). Use non-persistent messages when sending messages from 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 run on a range of devices, from sensors and actuators, to hand held devices and vehicle systems.

MQTT clients are small, and run on devices constrained by little memory and low processing power. The MQTT protocol is reliable and has small headers, which suits networks constrained by low bandwidth, high cost, and intermittent availability.

WebSphere MQ Telemetry provides three clients, which all 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.
- The advanced client, WebSphere MQ Telemetry daemon for devices, which is written in C, and can run on any suitable Linux or Windows platform.

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). For further information, see IBM Support & downloads.

Related concepts:

“Telemetry daemon for devices” on page 87

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.

Administering objects

Queue managers define the properties, or *attributes*, of IBM WebSphere MQ 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 objects. Objects are identified by an MQ *object descriptor* (MQOD) when addressed from a program.

When you use a WebSphere MQ command to carry out an object administration operation, such as defining, modifying, or deleting an object, the queue manager checks that you have the required level of authority to perform the operation. Similarly, when an application uses the MQOPEN call to open an object, the queue manager checks that the application has the required level of authority before it allows access to that object. The checks are made on the name of the object being opened.

The administration of objects includes the following tasks:

- 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 an overview of methods about how to create and manage WebSphere MQ objects, see “Managing objects” on page 116.

For further details of the object administration tasks, see the following subtopics:

Objects

Many of the administration tasks involve manipulating various types of IBM WebSphere MQ *objects*.

For information about naming IBM WebSphere MQ objects, see “Naming IBM WebSphere MQ objects” on page 112.

For information about the default objects created on a queue manager, see “System default objects” on page 117.

For information about the different types of IBM WebSphere MQ objects, see the following subtopics:

Related concepts:

“Introduction to message queuing” on page 31

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.

“Object attributes” on page 117

The properties of an object are defined by its attributes. Some you can specify, others you can only view.

Related information:

The MQSC commands

Queues:

Introduction to 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.

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 100.

Before using a queue, you must open the queue, specifying what you want to do with it. For example, you can open a queue for:

- Browsing messages only (not retrieving them)
- Retrieving messages (and either sharing the access with other programs, or with exclusive access)
- Putting messages on the queue
- Inquiring about the attributes of the queue
- Setting the attributes of the queue

For a complete list of the options that you can specify when you open a queue, see MQOPEN - Open object.

Attributes of queues

Some of the attributes of a queue are specified when the queue is defined, and cannot be changed afterward (for example, the type of the queue). Other attributes of queues can be grouped into those that can be changed:

- By the queue manager during the processing of the queue (for example, the current depth of a queue)
- Only by commands (for example, the text description of the queue)
- By applications, using the MQSET call (for example, whether put operations are allowed on the queue)

You can find the values of all the attributes using the MQINQ call.

The attributes that are common to more than one type of queue are:

QName Name of the queue

QType Type of the queue

QDesc Text description of the queue

InhibitGet

Whether programs are allowed to get messages from the queue (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.

Related concepts:

“Remote queues” on page 97

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.

“Alias queues” on page 98

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.

“Defining queues” on page 102

You define queues to IBM WebSphere MQ by using the MQSC command DEFINE or the PCF Create Queue command.

“Queues used by IBM WebSphere MQ” on page 103

IBM WebSphere MQ uses some local queues for specific purposes related to its operation.

Related reference:

“Local queues”

Transmission, initiation, dead-letter, command, default, channel, and event queues are types of local queue.

“Shared and cluster queues” on page 99

This information defines and explains the terms shared queues and cluster queues, as well as providing a comparison between the two.

“Dynamic and Model queues” on page 100

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.

Related information:

The MQSC commands

Developing applications reference

Local queues:

Transmission, initiation, dead-letter, command, default, channel, and event queues are types of local queue.

A queue is known to a program as *local* if it is owned by the queue manager to which the program is connected. You can get messages from, and put messages on, local queues.

The queue definition object holds the definition information of the queue as well as the physical messages put on the queue.

Each queue manager can have some local queues that it uses for special purposes:

Transmission queues

When an application sends a message to a remote queue, the local queue manager stores the message in a special local queue, called a *transmission queue*.

A *message channel agent* is a channel program is associated with the transmission queue and it delivers the message to its next destination. The next destination is the queue manager to which the message channel is connected. It is not necessarily the same queue manager as the final destination of the message. When the message is delivered to its next destination, it is deleted from the transmission queue. The message might have to pass through many queue managers on its journey to its final destination. You must define a transmission queue at each queue manager along the route, each holding messages waiting to be transmitted to the next destination. A normal transmission queue holds messages for the next destination, although the messages might have different eventual destinations. A cluster transmission queue holds messages for multiple destinations. The *correlID* of each message identifies the channel that the message is placed on to transfer it to its next destination.

You can define several transmission queues at a queue manager. You might define several transmission queues for the same destination, with each one being used for a different class of service. For example, you might want to create different transmission queues for small messages and large messages going to the same destination. You can then transfer the messages using different messages channels, so that the large messages do not hold up the smaller messages. On platforms other than z/OS, messages go onto the single cluster transmission queue, `SYSTEM.CLUSTER.TRANSMIT.QUEUE` by default. As an option on the other platforms, you can change the default, and separate the message traffic going to different cluster queue managers onto different cluster transmission queues. If you set the queue manager attribute `DEFCLXQ` to `CHANNEL`,

each cluster-sender channel creates a separate cluster transmission queue. Another option on the other platforms, is to manually define cluster transmission queues for cluster-sender channels to use.

Transmission queues can trigger a message channel agent to send messages onward; see Starting WebSphere MQ applications using triggers.

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.

Dead-letter (undelivered message) queue

A *dead-letter (undelivered message) queue* is a local queue on which the queue manager puts messages that it cannot deliver.

When the queue manager puts a message on the dead-letter queue, it adds a header to the message. The header information includes the reason that the queue manager put the message on the dead-letter queue. It also contains the destination of the original message, the date, and the time that the queue manager put the message on the dead-letter queue.

Applications can also use the queue for messages that they cannot deliver. For more information, see Using the dead-letter (undelivered message) queue .

System command queue

The *system command queue* is a queue to which suitably authorized applications can send 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.

System default queues

The *system default queues* contain the initial definitions of the queues for your system. When you create a queue definition, the queue manager copies the definition from the appropriate system default queue. Creating a queue definition is different from creating a dynamic queue. The definition of the dynamic queue is based upon the model queue you choose as the template for the dynamic queue.

Event queues

Event queues hold event messages. These messages are reported by the queue manager or a channel.

Remote queues:

To a program, a queue is *remote* if it is owned by a different queue manager to the one to which the program is connected.

Where a communication link has been established, a program can send a message to a remote queue. A program can never get a message from a remote queue.

The queue definition object, created when you define a remote queue, only holds the information necessary for the local queue manager to locate the queue to which you want your message to go. This object is known as the *local definition of a remote queue*. All the attributes of the remote queue are held by the queue manager that owns it, because it is a local queue to that queue manager.

When opening a remote queue, to identify the queue you must specify either:

- 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 95. 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.

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 94, alias queues have a *BaseQName* attribute. This is the name of the base queue to which the alias name resolves. For a fuller description of this attribute, see *BaseQName* (MQCHAR48).

The *InhibitGet* and *InhibitPut* attributes (see “Queues” on page 94) 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.

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 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*).

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*.

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*.

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, runmqtrm, and Starting WebSphere MQ applications using triggers*

Transmission queues

Transmission queues are queues that temporarily store messages that are destined for a remote queue manager. You must define at least one transmission queue for each remote queue manager to which the local queue manager is to send messages directly. These queues are also used in remote administration; see *Remote administration from a local queue manager*. For information about the use of transmission queues in distributed queuing, see *WebSphere MQ distributed-messaging techniques*.

Each queue manager can have a default transmission queue. If a queue manager that is not part of a cluster puts a message onto a remote queue, the default action is to use the default transmission queue. If there is a transmission queue with the same name as the destination queue manager, the message is placed on that transmission queue. If there is a queue manager alias definition, in which the **RQMNAME** parameter matches the destination queue manager, and the **XMITQ** parameter is specified, the message is placed on the transmission queue named by **XMITQ**. If there is no **XMITQ** parameter, the message is placed on the local queue named in the message.

Cluster transmission queues

Each queue manager within a cluster has a cluster transmission queue called **SYSTEM.CLUSTER.TRANSMIT.QUEUE**, and a model cluster transmission queue, **SYSTEM.CLUSTER.TRANSMIT.MODEL.QUEUE**. Definitions of these queues are created by default when you define a queue manager. If the queue manager attribute, **DEFCLXQ**, is set to **CHANNEL**, a permanent dynamic cluster transmission queue is automatically created for each cluster-sender channel that is created. The queues are called **SYSTEM.CLUSTER.TRANSMIT.Channe1Name**. You can also define cluster transmission queues manually.

A queue manager that is part of the cluster sends messages on one of these queues to other queue managers that are in the same cluster.

During name resolution, a cluster transmission queue takes precedence over the default transmission queue, and a specific cluster transmission queue takes precedence over **SYSTEM.CLUSTER.TRANSMIT.QUEUE**.

Dead-letter queues

A dead-letter (undelivered-message) queue is a queue that stores messages that cannot be routed to their correct destinations. A message cannot be routed when, for example, the destination queue is full. The supplied dead-letter queue is called **SYSTEM.DEAD.LETTER.QUEUE**.

For distributed queuing, define a dead-letter queue on each queue manager involved.

Command queues

The command queue, **SYSTEM.ADMIN.COMMAND.QUEUE**, is a local queue to which suitably authorized applications can send MQSC commands for processing. These commands are then retrieved by 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 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, Windows)
- The level of system control commands that the queue manager supports
- The maximum priority that you can assign to messages processed by the queue manager
- The name of the queue to which programs can send WebSphere MQ commands
- The maximum length of messages the queue manager can process
- 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 name of the queue manager's dead-letter (undelivered message) queue
- The name of the queue manager's default transmission queue
- The maximum number of open handles for any one connection
- The enabling and disabling of various categories of event reporting
- The maximum number of uncommitted messages within a unit of work

Queue managers and workload management

You can set up a cluster of queue managers that has more than one definition for the same queue (for example, the queue managers in the cluster could be clones of each other). Messages for a particular queue can be handled by any queue manager that hosts an instance of the queue. A workload-management algorithm decides which queue manager handles the message and so spreads the workload between your queue managers; see The cluster workload management algorithm for further information.

Process definitions:

Process definition objects allow applications to be started without the need for operator intervention by defining the attributes of the application for use by the queue manager.

The process definition object defines an application that starts in response to a trigger event on a IBM WebSphere MQ queue manager. The process definition attributes include the application ID, the application type, and data specific to the application. For more information, see the "Initiation queues" entry under "Queues used by IBM WebSphere MQ" on page 103.

To allow an application to be started without the need for operator intervention (described in Starting WebSphere MQ applications using triggers), the attributes of the application must be known to the queue manager. These attributes are defined in a *process definition object*.

The *ProcessName* attribute is fixed when the object is created; you can change other attributes by using the IBM WebSphere MQ commands.

You can inquire about the values of *all* the attributes using MQINQ - Inquire object attributes.

For a full description of the attributes of process definitions, see Attributes for process definitions.

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 by using 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.

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.

For more information about SSL and TLS, see Secure Sockets Layer (SSL) and Transport Layer Security (TLS) concepts.

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 Authentication information objects.

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. For more information about creating a COMMINFO object, see Getting started with multicast.

Related concepts:

“WebSphere MQ Multicast” on page 118

WebSphere MQ Multicast offers low latency, high fan out, reliable multicast messaging.

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-exit calls and data structures 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:

“Concepts of intercommunication” on page 39

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).

Related reference:

“Communications” on page 110

WebSphere MQ MQI clients use MQI channels to communicate with the server.

Related information:

Administering remote WebSphere MQ objects

Channel-exit calls and data structures

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”

A message channel is a one-way link. It connects two queue managers by using *message channel agents* (MCAs).

“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.

“Stopping channels” on page 109

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.

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.

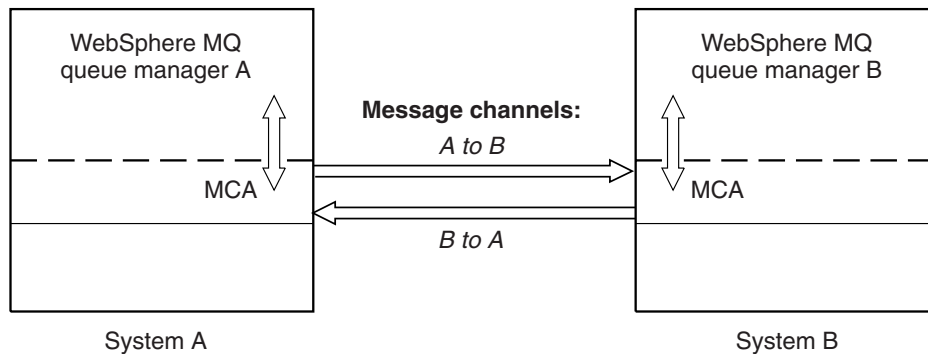


Figure 28. Message channels between two queue managers

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).

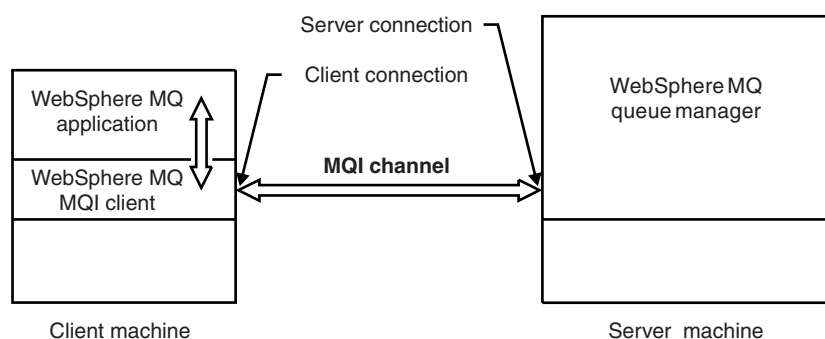


Figure 29. Client-connection and server-connection on an MQI channel

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).

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.

The transmission protocols possible are shown in the following table:

Table 8. Transmission protocols for MQI channels

Client platform	LU 6.2	TCP/IP	NetBIOS	SPX
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 27.

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 **runmqtsr** 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.

Services:

Service objects are a way of defining programs to be run when a queue manager starts or stops.

Programs can be one of the following types:

Servers

A *server* is a service object that has the parameter SERVTYPE specified as SERVER. A server service object is the definition of a program that will be executed when a specified queue manager is started. Only one instance of a server process can be executed concurrently. While running, the status of a server process can be monitored using the MQSC command, DISPLAY SVSTATUS. Typically server service objects are definitions of programs such as dead letter handlers or trigger monitors, however the programs that can be run are not limited to those supplied with 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.

Related information:

Working with services

Topic objects:

A *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 a topic object at any node in a topic tree by specifying that node's topic string in the TOPICSTR attribute of the topic object. You can also define other attributes for the administrative topic node. For more information about these attributes, see the The MQSC commands, or the Automating administration tasks. Each topic object will, by default, inherit its attributes from its closest parent administrative topic node.

topic objects can also be used to hide the full topic tree from application developers. If a topic object named FOOTBALL.US is created for the topic /Sport/American Football, an application can publish or subscribe to the object named FOOTBALL.US instead of the string /Sport/American Football with the same result.

If you enter a #, +, /, or * character within a topic string on a topic object, the character is treated as a normal character within the string, and is considered to be part of the topic string associated with a topic object.

For more information about topic objects, see Introduction to WebSphere MQ publish/subscribe messaging.

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" on page 113.

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 115

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.

Related reference:

“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.

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. The maximum length of channel object names and client connection channel names is 20 characters. See Defining the channels for more information about channels.
2. Topic strings can be a maximum of 10240 bytes. All IBM WebSphere MQ object names are case-sensitive.
3. The maximum length of storage class names is 8 characters.
4. 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
Lowercase a - z	<ul style="list-style-type: none"> • In MQSC scripts, names with lowercase characters must be enclosed in single quotation marks. This prevents the lowercase characters being folded into uppercase. • Systems using EBCDIC Katakana cannot use lowercase a- z characters in object names.
Numerics 0 - 9	• None
Period (.)	• None
Underscore (_)	• None
Forward slash (/)	<ul style="list-style-type: none"> • Windows On Windows systems, the first character of a queue manager name cannot be a forward slash.
Percent sign (%)	• None

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.

To refer to a remote queue, a program must include the name of the queue manager in the full queue name, or there must be a local definition of the remote queue.

When an application uses a queue name, that name can be either the name of a local queue (or an alias to one) or the name of a local definition of a remote queue, but the application does not need to know which, unless it needs to get a message from the queue (when the queue must be local). When the application opens the queue object, the MQOPEN call performs a name resolution function to determine on which queue to perform subsequent operations. The significance of this is that the application has no built-in dependency on particular queues being defined at particular locations in a network of queue managers. Therefore, if a system administrator relocates queues in the network, and changes their definitions, the applications that use those queues do not need to be changed.

Reserved object names

Object names that start with SYSTEM. are reserved for objects defined by the queue manager. You can use the **Alter**, **Define**, and **Replace** commands to change these object definitions to suit your installation. The names that are defined for IBM WebSphere MQ are listed in full in Queue names.

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.

Managing objects

An overview of how to create, alter, display, and delete objects.

For further information, see “Objects” on page 94.

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 and Automating administration tasks
- The MQSC commands described in The MQSC commands
- 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.
- IBM WebSphere MQ Administration Interface (MQAI) calls in a program. See WebSphere MQ Administration Interface (MQAI).
- **Windows** 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.

Windows **UNIX** **Linux** 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.

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
Understand what clusters are and how they work.

Related information:

Configuring a queue manager cluster
Setting up a new cluster

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 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 .

The only COMMINFO field that **MUST** be set is the name of the COMMINFO object. This name is then used to identify the COMMINFO object to a topic. The **GRPADDR** field of the COMMINFO object must be checked to ensure that the value is a valid multicast group address.

TOPIC object

A topic is the subject of the information that is published in a publish/subscribe message, and a topic is defined by creating a TOPIC object. For more information about the TOPIC object parameters, see DEFINE TOPIC.

Existing topics can be used with multicast by changing the values of the following TOPIC object parameters: **COMMINFO** and **MCAST**.

- **COMMINFO** This parameter specifies the name of the multicast communication information object.
- **MCAST** This parameter specifies whether multicast is allowable at this position in the topic tree. By default, **MCAST** is set to ASPARENT meaning that the multicast attribute of the topic is inherited from the parent. Setting **MCAST** to ENABLED allows multicast traffic at this node.

Multicast networks and topics

The following information is an overview of what happens to subscriptions with different types of subscription and topic definition. These examples all assume that the TOPIC object **COMMINFO** parameter is set to the name of a valid COMMINFO object:

Topic set to multicast enabled

If the topic string **MCAST** parameter is set to **ENABLED**, subscriptions from multicast capable clients are allowed and a multicast subscription is made unless:

- It is a durable subscription from a multicast capable client.
- It is a non-managed subscription from a multicast capable client.
- It is a subscription from a non-multicast capable client.

In these cases a non-multicast subscription is made and subscriptions are downgraded to normal publish/subscribe.

Topic set to multicast disabled

If the topic string **MCAST** parameter is set to **DISABLED**, a non-multicast subscription is always made and subscriptions are downgraded to normal publish/subscribe.

Topic set to multicast only

If the topic string **MCAST** parameter is set to **ONLY**, subscriptions from multicast capable clients are allowed and a multicast subscription is made unless:

- It is a durable subscription: Durable subscriptions are rejected with reason code 2436 (0984) (RC2436): MQRC_DURABILITY_NOT_ALLOWED
- It is a non-managed subscription: Non-managed subscriptions are rejected with reason code 2046 (07FE) (RC2046): MQRC_OPTIONS_ERROR
- It is a subscription from a non-multicast capable client: These subscriptions are rejected with reason code 2560 (0A00) (RC2560): MQRC_MULTICAST_ONLY
- It is a subscription from a locally bound application: These subscriptions are rejected with reason code 2560 (0A00) (RC2560): MQRC_MULTICAST_ONLY

Security

In IBM WebSphere MQ, there are several methods of providing security: the authorization service interface; user-written, or third party, channel exits; channel security using Secure Sockets Layer (SSL), channel authentication records, and message security.

Authorization service interface

Authorization for using MQI calls, commands, and access to objects is provided by the **object authority manager (OAM)**, which by default is enabled. Access to IBM 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](#).

User-written or third party channel exits

Channels can use user-written or third party channel exits. For more information, see [Channel-exit programs for messaging channels](#).

Channel security using 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. For an overview of SSL, including pointers to the commands described in this section, see Cryptographic security protocols: SSL and TLS.

Channel authentication records

Use channel authentication records to exercise precise control over the access granted to connecting systems at a channel level. For more information, see Channel authentication records.

Message security

Use WebSphere MQ Advanced Message Security, which is a separately installed and licensed component of IBM WebSphere MQ, to provide cryptographic protection to messages sent and receive using IBM WebSphere MQ. See WebSphere MQ Advanced Message Security.

Related information:

Security

Planning for your security requirements

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.

For information about clients in general, see “Overview of IBM WebSphere MQ MQI clients” on page 121.

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.

Related concepts:

“Transaction management and support” on page 127

An introduction to transaction management and how WebSphere MQ supports transactions.

“Extending queue manager facilities” on page 129

You can extend queue manager facilities by using user exits, API exits, or installable services.

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.

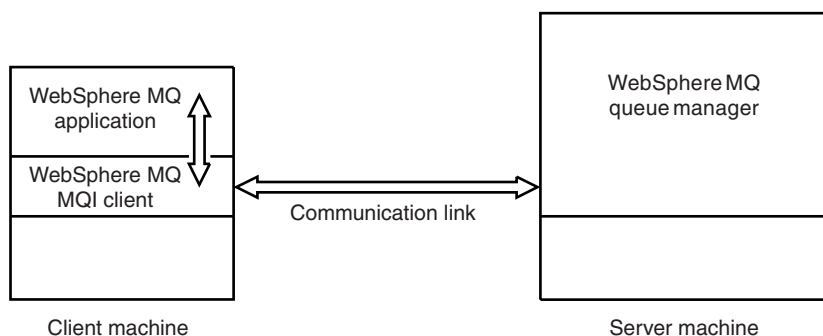


Figure 30. Link between a client and server

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 123.

IBM WebSphere MQ MQI client

UNIX and Linux
Windows

HP Integrity NonStop Server

IBM WebSphere MQ server

UNIX and Linux
Windows

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 Message Service (JMS). You can use Java and JMS clients on UNIX, Linux and Windows platforms. 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](#) and [Using WebSphere MQ classes for JMS](#).

Related concepts:

“Why use WebSphere MQ clients?”

Using WebSphere MQ clients is an efficient way of implementing WebSphere MQ messaging and queuing.

“How do I set up a IBM WebSphere MQ MQI client?” on page 124

Follow these instructions to set up a client.

“What is an extended transactional client?” on page 124

A WebSphere MQ extended transactional client can update resources managed by another resource manager, under the control of an external transaction manager.

“How the client connects to the server” on page 126

A client connects to a server using MQCONN or MQCONNX, and communicates through a channel.

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 IBM WebSphere MQ MQI client?”

The full MQI is supported in the client environment.

“Platform support for WebSphere MQ clients” on page 123

WebSphere MQ on all server platforms accepts client connections from WebSphere MQ MQI clients on UNIX or Linux systems, and Windows.

What applications run on a IBM 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 IBM WebSphere MQ MQI client system by linking the application on the IBM 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 9. MQGET options permitted when read ahead is enabled

	Permitted when read ahead is enabled and can be altered between MQGET calls	Permitted when read ahead is enabled but cannot be altered between MQGET calls ¹	MQGET options that are not permitted when read ahead is enabled ²
MQGET MD values	MsgId ³ CorrelId ³	Encoding CodedCharSetId	
MQGET MQGMO options	MQGMO_WAIT MQGMO_NO_WAIT MQGMO_FAIL_IF QUIESCING MQGMO_BROWSE_FIRST ⁴ MQGMO_BROWSE_NEXT ⁴ MQGMO_BROWSE_MESSAGE _UNDER_CURSOR ⁴	MQGMO_SYNCPOINT_IF_PERSISTENT MQGMO_NO_SYNCPOINT MQGMO_ACCEPT_TRUNCATED_MSG MQGMO_CONVERT MQGMO_LOGICAL_ORDER MQGMO_COMPLETE_MSG MQGMO_ALL_MSGS_AVAILABLE MQGMO_ALL_SEGMENTS_AVAILABLE MQGMO_MARK_BROWSE_HANDLE MQGMO_MARK_BROWSE_CO_OP MQGMO_UNMARK_BROWSE_CO_OP MQGMO_UNMARK_BROWSE_HANDLE MQGMO_UNMARKED_BROWSE_MSG MQGMO_PROPERTIES_FORCE_MQRFH2 MQGMO_NO_PROPERTIES MQGMO_PROPERTIES_IN_HANDLE MQGMO_PROPERTIES_COMPATIBILITY	MQGMO_SET_SIGNAL MQGMO_SYNCPOINT MQGMO_MARK_SKIP _BACKOUT MQGMO_MSG_UNDER _CURSOR ⁴ MQGMO_LOCK MQGMO_UNLOCK
MQGMO values		MsgHandle	

1. If these options are altered between MQGET calls an MQRC_OPTIONS_CHANGED reason code is returned.
2. If these options are specified on the first MQGET call then read ahead is disabled. If these options are specified on a subsequent MQGET call a reason code MQRC_OPTIONS_ERROR is returned.
3. The client applications need to be aware that if the MsgId and CorrelId values are altered between MQGET calls messages with the previous values might have already been sent to the client and remain in the client read ahead buffer until consumed (or automatically purged).
4. The first MQGET call determines whether messages are to be browsed or got from a queue when read ahead is enabled. If the application attempts to use a combination of browse and get an MQRC_OPTIONS_CHANGED reason code is returned.
5. MQGMO_MSG_UNDER_CURSOR is not possible with read ahead. Messages can be browsed or got when read ahead is enabled but not a combination of both.

An application running on a IBM 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).

Platform support for WebSphere MQ clients:

WebSphere MQ on all server platforms accepts client connections from WebSphere MQ MQI clients on UNIX or Linux systems, and Windows.

WebSphere MQ installed as a *Base product and Server* can accept connections from the WebSphere MQ MQI clients on the following platforms:

- HP Integrity NonStop Server
- 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 123.
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.
3. Ensure that your communication links are configured and connected. Configuration of communication links is described in Configuring connections between the server and client.
4. Check that your installation is working correctly. Verifying your installation is described in Verifying a client installation.
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.
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. There are some additional considerations if you are using SSL. These considerations are described in Specifying that an MQI channel uses SSL. 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.
7. IBM WebSphere MQ applications are fully described in the Developing applications.
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
 - Building applications for WebSphere MQ MQI clients
 - Connecting IBM WebSphere MQ MQI client applications to queue managers
 - Resolving problems with IBM WebSphere MQ MQI clients

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 127.

Note that the XA transactional client is now supplied as part of WebSphere MQ.

A client application can participate in a unit of work that is managed by a queue manager to which it is connected. Within the unit of work, the client application can put messages to, and get messages from, the queues that are owned by that queue manager. The client application can then use the **MQCMIT** call to commit the unit of work or the **MQBACK** call to back out the unit of work. However, within the same unit of work, the client application cannot update the resources of another resource manager, the tables of a DB2® database, for example. Using 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.

Related reference:

“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.

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 7.5 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 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 V7.5 - AIX
HP-UX	Minimum Requirements for WebSphere MQ V7.5 - HP-UX
HP Integrity NonStop Server	../com.ibm.mq.pla.doc/q113320_.dita
Linux	Minimum Requirements for WebSphere MQ V7.5 - Linux
Solaris	Minimum Requirements for WebSphere MQ V7.5 - Solaris
Windows	Minimum Requirements for WebSphere MQ V7.5 - 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”

You can also run an application in the WebSphere MQ MQI client environment when your machine also has a queue manager installed.

“Clients on different platforms” on page 127

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.

“Using different versions of client and server software” on page 127

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.

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.

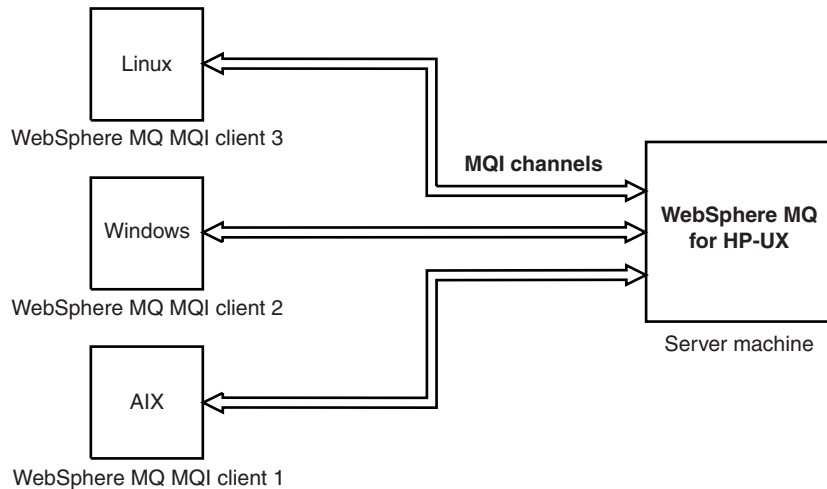


Figure 31. WebSphere MQ server connected to clients on different platforms

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 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 also provides support for the Microsoft Transaction Server (COM+). Using the Microsoft Transaction Server (COM+) 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.

Data conversion exits

These exits create source code fragments that can be put into application programs to convert data from one format to another. Data conversion exits are described in the Writing data-conversion exits .

The cluster workload exit

The function performed by this exit is defined by the provider of the exit. Call definition information is given in MQ_CLUSTER_WORKLOAD_EXIT - Call description.

API exits

API exits let you write code that changes the behavior of 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.

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*.

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

Describes where you can find the IBM WebSphere MQ client for HP Integrity NonStop Server SupportPac, and gives 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, *IBM WebSphere MQ Clients for NSS*, for details, see http://www.ibm.com/support/docview.wss?rs=171&uid=swg24034802&loc=en_US&cs=utf-8&lang=en.

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/76385-0-0-121.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 10.

	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.*`

Scenarios

This section provides information about scenarios that explains how to use and combine new WebSphere MQ version 7.5 function. The scenarios include useful links to infocenter content to help you to gain a better understanding of the area in which you are interested.

The available scenarios are described in the following subtopics:

Getting started with WebSphere MQ V 7.5

This scenario explains how to get started with IBM WebSphere MQ Version 7.5 on a Windows platform. Use this scenario if you have never used IBM WebSphere MQ and want to get started quickly.

This scenario contains the following sections. It is possible to complete these steps by using either the graphical or command-line interfaces, as shown in this scenario.

Basic concepts and key terms

Description of the basic concepts and key terms you must know about before using the Getting started with IBM WebSphere MQ Version 7.5 scenario.

Basic concepts

IBM WebSphere MQ enables applications to read and write messages to a queue. The application that reads the message is independent of the application that writes the message. It is not a requirement to have the two applications running at the same time. If no application is available to read the message it is queued on the IBM WebSphere MQ queue until an application reads it.

Key terms

Here is a list of key terms about message queuing.

Term	Description
Queue managers	The queue manager is responsible for maintaining the queues it owns, and for storing all the messages it receives onto the appropriate queues.
Messages	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. The applications can be running on the same or on different computers.
Local queues	A local queue is a data structure used to store messages. The queue can be a normal queue or a transmission queue. A normal queue holds messages that are to be read by an application that is reading the message directly from the queue manager. A transmission queue holds messages that are in transit to another queue manager.
Remote queues	A remote queue is used to address a message to another queue manager.
Channels	Channels are use to send and receive message between queue managers.
Listeners	Listeners are processes that accept network requests from other queue managers, or client applications, and start associated channels.

Creating a queue manager called QM1

Create a queue manager, called QM1, for use with the Getting started with IBM WebSphere MQ Version 7.5 scenario, by using either the command-line interface or the WebSphere MQ Explorer. Queue managers are the main components in a WebSphere MQ messaging network.

Before you begin

You must have IBM WebSphere MQ Version 7.5 installed. If you do not, see [Installing and uninstalling](#) for information about how to do so.

About this task

In this example, all names are typed in uppercase and because IBM WebSphere MQ names are case-sensitive, you must type all names in uppercase too.

Creating the queue manager by using the command-line interface:

To create and start a queue manager by using the command-line interface, complete the following steps:

Procedure

1. Create a queue manager with the name QM1 by typing the following command:

```
crtmqm QM1
```

When the system creates the queue manager, the following output is displayed:

```
C:\>crtmqm QM1
WebSphere MQ queue manager created.
Creating or replacing default objects for QM1.
Default objects statistics : 61 created. 0 replaced. 0 failed.
Completing setup.
Setup completed.
```

The queue manager is created, and is stopped. You must start the queue manager before you can administer it, and before you can read and write messages from its queues.

2. Start the queue manager by entering the following command:

```
strmqm QM1
```

When the queue manager successfully starts, the following output is displayed:

```
C:\>strmqm QM1
WebSphere MQ queue manager 'QM1' starting.
5 log records accessed on queue manager 'QM1' during the log replay phase.
Log replay for queue manager 'QM1' complete.
Transaction manager state recovered for queue manager 'QM1'.
WebSphere MQ queue manager 'QM1' started.
```

The queue manager is started.

What to do next

To create a queue, see [“Creating a queue called LQ1”](#) on page 134.

Creating the queue manager by using WebSphere MQ Explorer:

To create and start a queue manager by using the WebSphere MQ Explorer, complete the following steps

Procedure

1. Start WebSphere MQ Explorer.

2. In the Navigator view, right-click the Queue Managers folder, then click **New > Queue Manager**. The Create Queue Manager wizard starts.
3. In the **Queue Manager name** field, type QM1.
4. Select the Make this the default queue manager check box.
5. In the **Dead-letter queue** field type SYSTEM.DEAD.LETTER.QUEUE. This is the name of the dead-letter queue that is automatically created when you create the queue manager.
6. Leave the other fields empty and click **Finish**, or if that button is disabled, click **Next**. The **Finish** button is disabled if the port number conflicts with an existing queue manager, for example the queue manager that is created as part of the default configuration. You must continue through the wizard to change the default port number.
7. If you clicked **Next**, continue to accept the defaults and click **Next** on each page until you get to the final page of the wizard, when the **Finish** button becomes available. Change the specified port number, for example to 1415, and click **Finish**.
WebSphere MQ displays a Creating Queue Manager dialog window while the queue manager is created and started.

What to do next

To create a queue, see “Creating a queue called LQ1.”

Creating a queue called LQ1

Create a queue for use with the Getting started with WebSphere IBM WebSphere MQ Version 7.5 scenario, by using either the command-line interface or the WebSphere MQ Explorer. Queues are data structures that are used to store messages and are IBM WebSphere MQ queue manager objects.

About this task

There are three ways to create IBM WebSphere MQ objects:

- Command-line.
- IBM WebSphere MQ Explorer.
- Using a programmable interface.

In this task you can create IBM WebSphere MQ objects using the command-line or the IBM WebSphere MQ Explorer.

Creating a queue by using the command-line interface:

The command-line interface has a scripting language called IBM WebSphere MQ Script Commands (MQSC). The scripting tool, **runmqsc**, is used to run the script against a queue manager. To create and start a queue by using the command-line interface, complete the following steps:

Procedure

1. Start the scripting tool by typing the following command:

```
runmqsc QM1
```

When the scripting tool starts, the following output is displayed:

```
C:\>runmqsc QM1
5724-H72 (C) Copyright IBM Corp. 1994, 2008. ALL RIGHTS RESERVED.
Starting MQSC for queue manager QM1.
```

The tool is ready to accept MQSC commands.

2. Create a local queue called LQ1 by typing the following MQSC command:
define qlocal(LQ1)

When the queue is created, the following output is displayed:

```
define qlocal(LQ1)
  2 : define qlocal(LQ1)
AMQ8006: WebSphere MQ queue created.
```

3. Stop the scripting tool by typing the following MQSC command:

```
end
```

When the scripting tool ends, the following output is displayed:

```
One MQSC command read.
No commands have a syntax error.
All valid MQSC commands were processed.
```

```
C:\>
```

What to do next

You are ready to put a message on to your queue. To put a message in a queue, see “Putting a message to the queue LQ1.”

Creating a queue by using WebSphere MQ Explorer:

To create and start a queue by using the WebSphere MQ Explorer, complete the following steps:

Procedure

1. In the Navigator view, expand the **Queue Managers** folder.
2. Expand queue manager **QM1**.
3. Right-click the **Queues** folder, then click **New > Local Queue...** The New Local Queue wizard starts.
4. In the **Name** field, type LQ1.
5. Click **Finish**.

The new queue LQ1, is displayed in the Content view. If the queue is not displayed in the Content view, click on the **Refresh** button, at the top of the Content view.

What to do next

You are ready to put a message on to your queue. To put a message in a queue, see “Putting a message to the queue LQ1.”

Putting a message to the queue LQ1

Put a message on to the queue LQ1, for use with the Getting started with IBM WebSphere MQ Version 7.5 scenario, by using either the command-line interface or the IBM WebSphere MQ Explorer.

About this task

IBM WebSphere MQ comes with a sample application called **amqsput**. This application puts a message to a predefined queue.

Putting a message to the queue by using the command-line interface:

To put a message on to the queue by using the command-line interface, complete the following steps:

Procedure

1. Use the **amqsput** sample application to put a message to queue LQ1, by typing the following command:

```
amqsput LQ1 QM1
```

When the sample application starts, the following output is displayed:

```
C:\>amqsput LQ1 QM1
Sample AMQSPUT0 start
target queue is LQ1
```

2. Type Hello World and press Enter. You placed a message that contains the text "Hello World" on the queue LQ1 managed by the queue manager called QM1.
3. To end **amqsput**, press Enter. The following output is displayed:

```
C:\>amqsput LQ1 QM1
Sample AMQSPUT0 start
target queue is LQ1
Hello World
```

```
Sample AMQSPUT0 end
```

What to do next

To get a message from the queue, see "Getting a message from the queue LQ1."

Putting a message to the queue by using IBM WebSphere MQ Explorer:

To put a message on to the queue by using the IBM WebSphere MQ Explorer, complete the following steps:

Procedure

1. In the Navigator view, expand the **Queue Managers** folder.
2. Expand queue manager QM1, which you created.
3. Click the **Queues** folder. The queue manager's queues are listed in the Content view.
4. In the Content view, right-click the local queue LQ1, then click **Put Test Message...** The **Put test message** dialog opens.
5. In the **Message data** field, type some text, for example Hello World, then click **Put message**. The **Message data** field is cleared and the message is put on the queue.
6. Click **Close**. In the Content view, notice that the LQ1 **Current queue depth** value is now 1. If the **Current queue depth** column is not visible, you might need to scroll to the right of the Content View.

What to do next

To get a message from the queue, see "Getting a message from the queue LQ1."

Getting a message from the queue LQ1

Get a message from the queue LQ1, for use with the Getting started with IBM WebSphere MQ Version 7.5 scenario, by using either the command-line interface or IBM WebSphere MQ Explorer.

About this task

IBM WebSphere MQ comes with a sample application called **amqsget**. This application reads messages from a queue.

Getting a message from the queue using the command-line interface:

To get a message from the queue by using the command-line interface, complete the following steps:

Procedure

Use the **amqsget** sample application to read a message on the queue LQ1, by typing the following command:

```
amqsget LQ1 QM1
```

When the sample application starts, the following output is displayed:

```
C:\>amqsget LQ1 QM1
Sample AMQSGET0 start
message <Hello World>
no more messages
Sample AMQSGET0 end
```

The **amqsget** application ends 30 seconds after reading the message.

What to do next

To learn about writing queuing applications, connecting to and disconnecting from a queue manager, publish/subscribe, and opening and closing objects, see [Writing a queuing application](#).

Getting a message from the queue by using IBM WebSphere MQ Explorer:

To get a message from the queue by using the IBM WebSphere MQ Explorer, complete the following steps:

Procedure

1. In the Navigator view, expand the **Queue Managers** folder, then expand QM1.
2. Click the **Queues** folder.
3. In the Content view, right-click QM1, then click **Browse Messages....** The Message browser opens to show the list of the messages that are currently on QM1.
4. Double-click the last message to open the properties dialog.

On the **Data** page of the properties dialog, the **Message data** field displays the content of the message in human-readable form.

What to do next

To learn about writing queuing applications, connecting to and disconnecting from a queue manager, publish/subscribe, and opening and closing objects, see [Writing a queuing application](#).

What to do next

What to do next on completion of the Getting started with IBM WebSphere MQ Version 7.5 scenario.

IBM WebSphere MQ provides role-based training paths to assist you by defining a path to acquiring skills for specific WebSphere product offerings. There are two training paths for IBM WebSphere MQ:

- Application Developer

These users are responsible for creating the applications that use the queue manager. In this scenario, they write the applications **amqsput** and **amqsget**.

- System Administrator

These users are responsible for creating the queue manager and its objects, they typically carry out similar tasks to that you covered in this scenario.

For more information about IBM WebSphere MQ training paths, see: <http://www.ibm.com/software/websphere/education/paths/>.

To view the full list of IBM WebSphere MQ courses, see: <http://www.ibm.com/software/websphere/education/curriculum/appint/wmq/>.

A certification program is available which demonstrates you achieve a skill level in IBM WebSphere MQ. For more information, see: http://www.ibm.com/certify/certs/ws_index.shtml.

Conferences are available for you to attend, for a list, see: <http://www-304.ibm.com/jct03001c/services/learning/ites.wss?pagetype=page&c=a0015064>.

You can collaborate with other users, for example, for:

- E-mail based community of IBM WebSphere MQ professionals, see <https://listserv.meduniwien.ac.at/archives/mqser-l.html>.
- Discussion forums focusing on the IBM WebSphere MQ family of products, see <http://www.mqseries.net/>.
- A developerWorks® blog by the developers of the various IBM messaging products, see <http://www.ibm.com/developerworks/blogs/page/messaging/>.
- The official IBM-hosted forum for IBM WebSphere MQ, see <http://www.ibm.com/developerworks/forums/forum.jspa?forumid=280>.
- IBM WebSphere MQ tagged questions and answers on [stackoverflow.com](http://stackoverflow.com/questions/tagged/websphere-mq), see <http://stackoverflow.com/questions/tagged/websphere-mq>.

There are additional topics for you to view in this product documentation. You might want to look at the following sections:

- Administering IBM WebSphere MQ

IBM WebSphere MQ provides control commands that you can use. You use two of these commands in this scenario: **crtmqm** and **strmqm**. This section also provides a good overview about message queuing.

- MQSC reference

In this scenario, you use the `define qlocal('LQ1')` command to define a local queue called LQ1; this command is an MQSC command. IBM WebSphere MQ System Administrators use these commands to manage their queue managers. This section introduces the commands and shows you how to use them, before describing the commands in detail, in alphabetical order.

- Configuring a queue manager cluster

This section describes how to organize, use, and manage queue managers in virtual groups known as clusters. Clustering ensures that each queue manager within a cluster knows about all the other queue managers in the same cluster. Clustering also makes the management of complex queue manager networks simpler.

The Product Connectivity Scenarios product documentation provides information that leads you through the key tasks required to connect WebSphere Application Server to WebSphere MQ in a variety of scenarios. Each scenario contains the instructions for implementing a solution in a business context, allowing you to learn as you go without needing to make use of other information resources.

<http://publib.boulder.ibm.com/infocenter/prodconn/v1r0m0/index.jsp>

Basic file transfer using the scripts

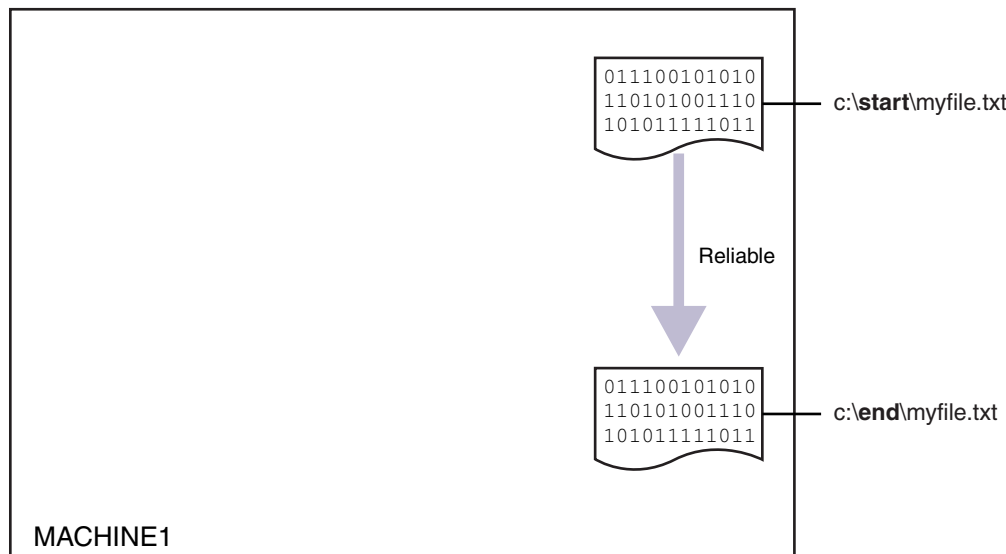
You can transfer files in a number of different ways using IBM WebSphere MQ Version 7.5. Review the topics in this section to understand what is covered in this scenario, the reasons why a business might want to follow the scenario, the user roles involved, and an overview of the solution proposed by the scenario.

Transferring files with control, reliably, and with auditing can be a fundamental requirement in your enterprise. IBM WebSphere MQ Version 7.5 provides a Managed File Transfer capability as part of its integrated messaging platform. You can use Managed File Transfer capability to integrate files seamlessly into your messaging infrastructure, either through basic file transfers, or fully fledged participants in messaging. For more details on this capability, see WebSphere MQ Managed File Transfer introduction.

This scenario provides you with a basic understanding of how to integrate files into the simplest IBM WebSphere MQ messaging topology. To do this, you work through a basic IBM WebSphere MQ scenario designed to move a file from one location to another. Although this initial scenario is restricted to a single computer, it gives you experience of configuring the environment and forms an important foundation for later scenarios. Later scenarios demonstrate how you can use IBM WebSphere MQ to transfer files across a network, and then begin to show how the Managed File Transfer component can address real business problems.

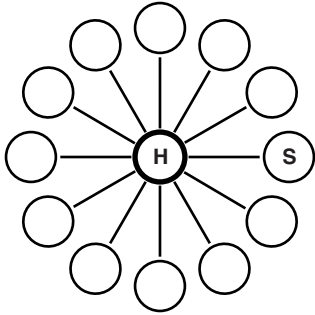
You should have a basic understanding of IBM WebSphere MQ, specifically, the notion of a queue manager and basic configuration and administration of IBM WebSphere MQ through to using commands such as `runmqsc` and the IBM WebSphere MQ Explorer.

In this scenario, you explore how IBM WebSphere MQ can be used to initiate and track a transfer of a file from one location to another on a single computer, giving you experience of installing, configuring, and using the reliable managed file transfer capability in IBM WebSphere MQ Version 7.5.



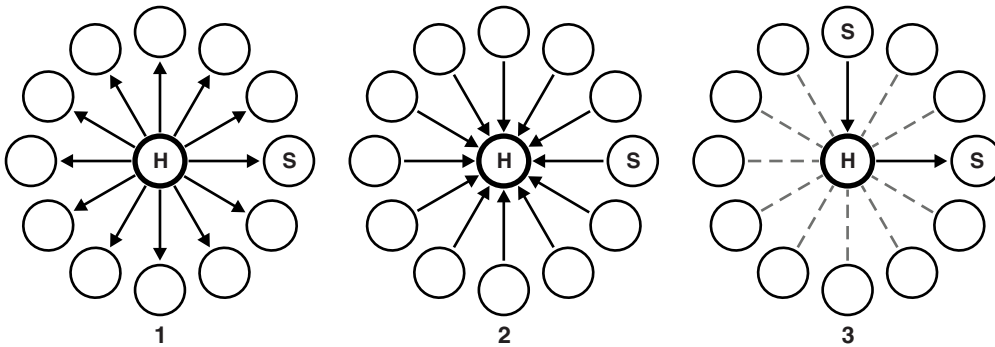
Example file transfer topology

This and subsequent scenarios are based around a hub and spoke topology. The diagram shows conceptual hub and spoke topology that comprises the hub (H) and multiple spokes (S).



Such a topology finds applications in many business scenarios, for example, you might want to:

1. Send files from a centralized HQ (the hub) to many outlying nodes (the spokes).
2. Collate files from many nodes (the spokes) to a single location (the hub).
3. Transfer files from one spoke to another.



Your topology might differ from this example, but the principles and commands in this scenario can be readily extended to cater for any IBM WebSphere MQ network.

Planning the solution

Planning the transfer of files to or from a computer using the scripts. Includes understanding the security model, prerequisites and licenses, installing and configuring IBM WebSphere MQ, and preparing your users and groups.

To transfer a file from point A to point B, create a hub and spoke topology on a single computer. This topology comprises the hub, a IBM WebSphere MQ queue manager, and two spokes each a file transfer agent.

File transfer agents are Java processes that run on the computer and transfer files to and from other agents. In this scenario, define a file transfer that uses these file transfer agents to move a sample file from one location to another through two mechanisms:

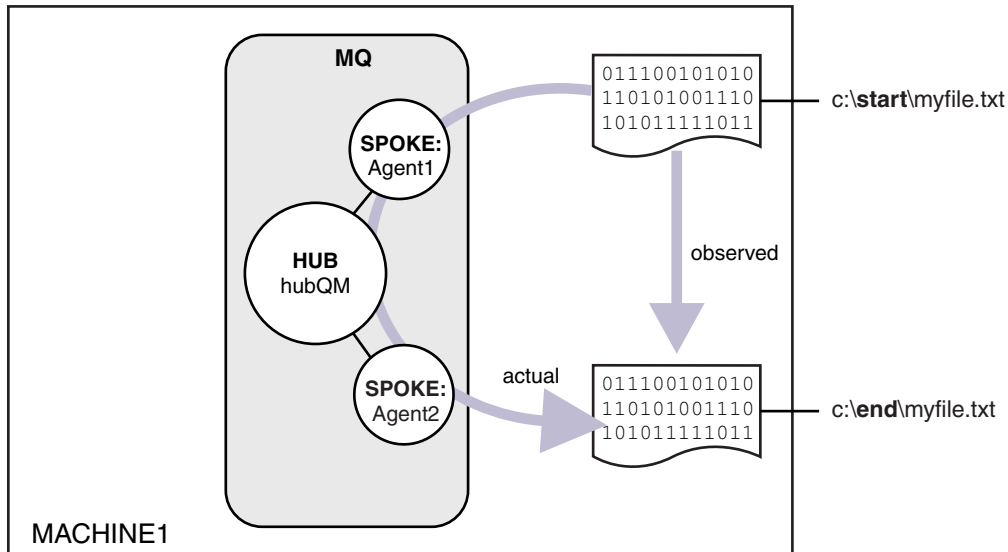
1. By using the graphical IBM WebSphere MQ Explorer.
2. By way of the command line.

The scenario assumes that you have a Windows system. For a UNIX system, substitute appropriate paths and commands, and ensure that you have read and write access to all relevant directories. The scenario also assumes that you have a basic understanding of what a queue manager does.

To transfer files to or from a computer, you must have a file transfer agent running on that computer. Every agent connects to a IBM WebSphere MQ queue manager and uses IBM WebSphere MQ to communicate with other agents. For more information, see WebSphere MQ Managed File Transfer topology overview.

IBM WebSphere MQ Version 7.5 has two sample scripts createHub and addSpoke that help you build up a file transfer topology quickly and easily. Use these scripts to construct this topology from the hub outwards. The scripts are samples, and you can modify them to meet your own requirements. If you do want to modify the scripts, copy them to a location in your own user directory first.

Sample scripts createHub and addSpoke are used to construct the topology.



When a transfer is started, you observe the file copied from one location to another. This simple scenario can be achieved through a single file transfer agent that manages both the source and destination files. However, two file transfer agents are used to provide you with a better example of how this setup works before moving to a multiple computer topology.

Understanding the security model

The createHub and addSpoke scripts configure a file transfer topology with the following security characteristics:

- Access to IBM WebSphere MQ is partitioned between three roles:
 1. The IBM WebSphere MQ administrator who configures IBM WebSphere MQ and runs the createHub and addSpoke scripts.
 2. FTAGENTS who start, stop, and interact with agents.
 3. FTUSERS who initiate file transfers.

For more information about configuring access, see Authorities for resources specific to WebSphere MQ Managed File Transfer.

- All incoming connections from agents are mapped to a single user FTAGENT at the hub.
- Weak IP-based authentication is used to authenticate individual agents.

For clarity, the security model in the script is not fully hardened. You must understand your own topology needs and security threats before using these scripts in production. You must therefore consider whether to address the following potential vulnerabilities and recommendations:

- Any user can impersonate any other. Consider finer granularity in the object access model for file transfer resources.
- Any agent can impersonate any other. Consider stronger authentication, for example, TLS/SSL and finer granularity in the object access model for file transfer resources.
- The interface between file system versus IBM WebSphere MQ security is not considered. Consider implementing file sandboxing, and understand the impact of permissions of the agent's configuration files. For more information about sandboxing, see Sandboxes.
- The interface between the agent and the operating system is not described. Consider implementing file sandboxing. For more details, see Sandboxes.

For more information about security, IBM WebSphere MQ, and file transfers, see [What to do next](#).

Prerequisites and licenses

You need the following items:

- A test computer that satisfies the hardware and operating system prerequisites for IBM WebSphere MQ, for details, see <http://www-01.ibm.com/support/docview.wss?uid=swg27006467>, with no existing installation of IBM WebSphere MQ or IBM WebSphere MQ data.
- IBM WebSphere MQ Version 7.5. You can download a trial version from <http://www.ibm.com/developerworks/downloads/ws/wmq/>.

Preparing your computer

Ensure your test computer satisfies the requirements for installation of IBM WebSphere MQ Version 7.5, see [Checking requirements](#).

Ensure that your computer is prepared appropriately for installation, see [Verifying a local installation using the command line](#).

Installing IBM WebSphere MQ Version 7.5

Install IBM WebSphere MQ Version 7.5 server with the following components: Server, IBM WebSphere MQ Explorer, IBM WebSphere MQ Advanced Managed File Transfer Agent, and IBM WebSphere MQ Managed File Transfer Command Line Tools. For details, see [Choosing what to install](#).

Decide how you want to administer IBM WebSphere MQ. You can administer IBM WebSphere MQ by:

- Setting up an appropriate environment using the `setmqenv` command. For more information, see [setmqenv](#).
- Calling fully qualified IBM WebSphere MQ administrative commands.

The scenario assumes that you are using a clean computer with no previous installations of IBM WebSphere MQ or IBM WebSphere MQ File Transfer Edition installed. If not, you must determine whether coexistence is supported or adjust the installation mechanism and configuration of environments. For details, see [Multiple installations](#).

Creating your users and groups

This security model assumes that you have created the following users and groups:

Users

- `mqmAdmin`
IBM WebSphere MQ administrator, that is, a member of the `mqm` group, or on Windows a member of the Administrators group. You must create this user, and make it a member of the `mqm` group.

- ftuser

You must create this user, and make it a member of the FTUSERS group. To avoid the potential for administrative level security acts on the queue manager, do not add this user to the mqm group.

- ftagent

You must create this user, and make it a member of the FTAGENTS group. To avoid the potential for administrative level security acts on the queue manager, do not add this user to the mqm group.

Groups

- mqm

Automatically created as part of the IBM WebSphere MQ installation. Members of this group can administer IBM WebSphere MQ and its resources.

- FTUSERS

You must create this group. Members of this group can initiate file transfers.

- FTAGENTS

You must create this group. Members of this group can start and stop file transfer agents which are the endpoints that handle the transfers of files in your network.

Configuring IBM WebSphere MQ for file transfers

Configure IBM WebSphere MQ for file transfers by using the sample scripts createHub and addSpoke to build the topology for the basic file transfer using scripts scenario.

Procedure

1. Identify a free port that remote agents (defined in later scenarios) can connect to, for example, 1414.
2. Identify a suitable name for a queue manager to act as the hub of the file transfer network, for example, hubQM.
3. As the user mqmAdmin, from the IBM WebSphere MQ bin directory, <MQ_INSTALL_ROOT>/bin:

- a. Create the hub.

Enter the following command:

```
<MQ_INSTALL_ROOT>\mqft\samples\scripts\createHub hubQmgr=hubQM hubPort=1414
```

- b. Add the first agent spoke (in BINDINGS because it is local to the queue manager) calling the agent AGENT1.

```
<MQ_INSTALL_ROOT>\mqft\samples\scripts\addSpoke agentName=AGENT1
hubQmgr=hubQM connectionMode=BINDINGS
```

- c. Add the second agent spoke (in BINDINGS because it is local to the queue manager), calling the agent AGENT2.

```
<MQ_INSTALL_ROOT>\mqft\samples\scripts\addSpoke agentName=AGENT2
hubQmgr=hubQM connectionMode=BINDINGS
```

4. As the user ftagent:

- a. Start AGENT1 by entering the following command:

```
fteStartAgent -p hubQM AGENT1
```

If you configured more than one hub, the -p hubQM part of the command ensures that you pick up the correct one. It is not needed for a single hub.

- b. Start AGENT2 by entering the following command:

```
fteStartAgent -p hubQM AGENT2
```

- c. List the agents to confirm they are running correctly.

```
fteListAgents -p hubQM
```

You see the following output:

Agent Name:	Queue Manager Name:	Status:
AGENT1	hubQM	READY
AGENT2	hubQM	READY

Implementing the solution

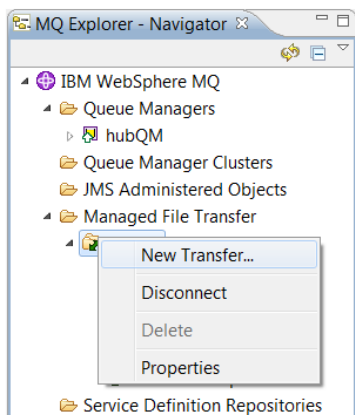
Implementing the solution in this scenario involves the use of IBM WebSphere MQ Explorer to define and initiate a file transfer. You can monitor transfer progress, and confirm its success by inspecting the file system.

About this task

To avoid the need to configure IBM WebSphere MQ Explorer for the non-administrative user ftuser, run this part of the scenario as the user mqmAdmin . If you want to enable this capability, see Security.

Procedure

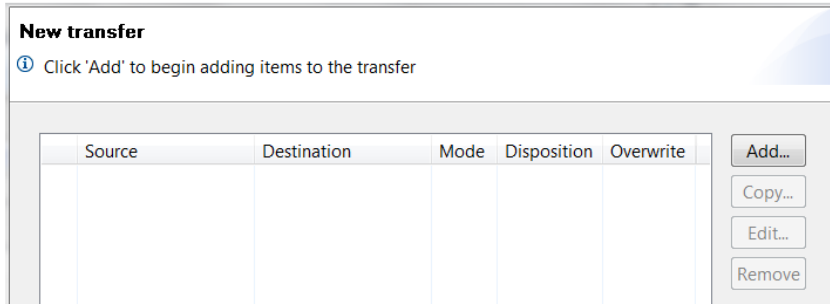
1. Identify source file and target directory:
 - a. Create a sample file to transfer, for example, C:\start\myfile.txt
 - b. Identify an existing directory to transfer this file to, for example, C:\end\
2. As an administrator, mqmAdmin in this example, start IBM WebSphere MQ Explorer. Start the program from the Start menu (or equivalent), or run the command **MQExplorer**. For more details, see [Launching WebSphere MQ Explorer](#) .
3. In the **Managed file transfer** section, right click the configuration named hubQM and select **Connect**.
4. Under the **Managed file transfer** section in the IBM WebSphere MQ navigator, right-click on hubQM and select **New Transfer** to start the New Transfer wizard



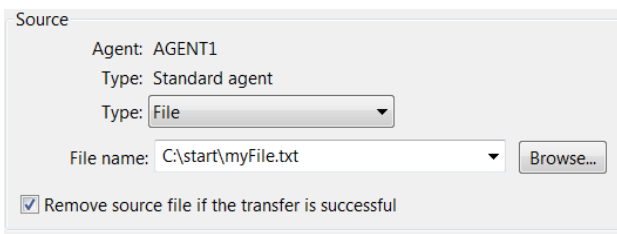
5. Use the menu to select AGENT1 as the source agent and AGENT2 as the destination agent.

Source agent	
Name:	AGENT1
Type:	Standard
Destination agent	
Name:	AGENT2
Type:	Standard

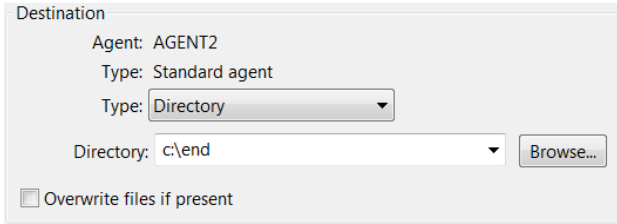
6. Click **Next**.



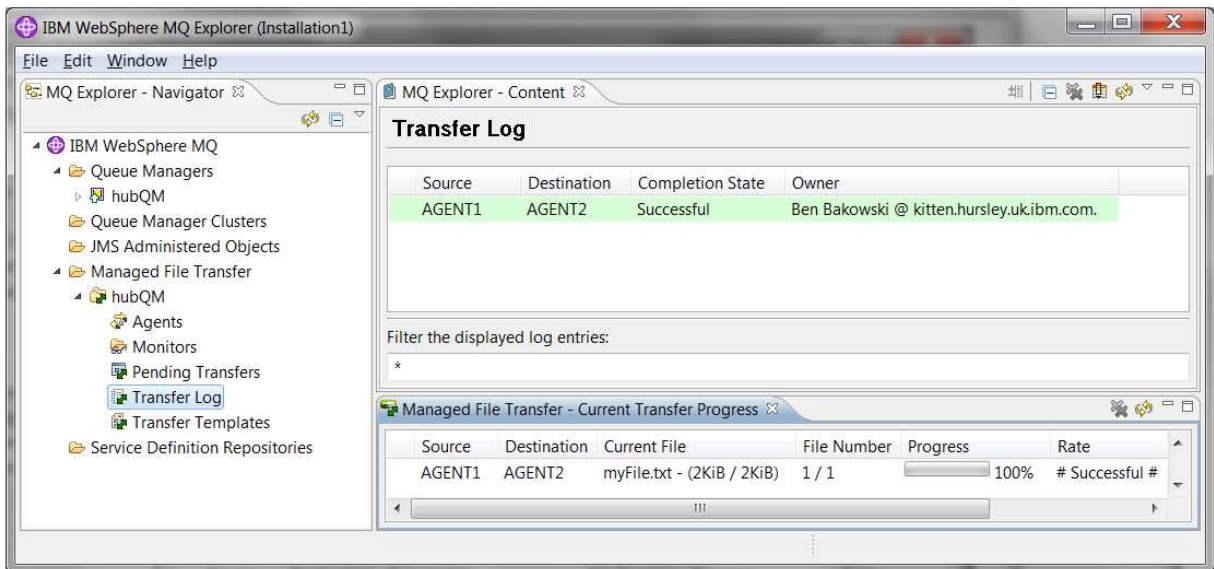
7. Click **Add...** to start selecting the files to transfer from AGENT1 to AGENT2 .
8. In the **Source** frame, click **Browse** and browse to the file identified earlier, for example, C:\start\myFile.txt. Ensure the **Remove source file if the transfer is successful checkbox** is checked.



9. In the **Destination** frame, select a **Type** of Directory, and then enter the destination directory you identified earlier, for example, C:\end\.



10. Click **OK** and then **Finish** and the transfer starts.
11. You can monitor transfer progress in the Current Transfer Progress tab at the bottom of IBM WebSphere MQ Explorer:



12. You can also inspect the file system manually to confirm that the new file exists, for example, C:\end\myfile.txt .

Using the command line to transfer a file:

In this section, the flexibility of file transfers is demonstrated by showing you how to use the command line to initiate a transfer. Although out of scope for this scenario, you can build on these principles and the Ant scripting technology to define and implement much more powerful file transfer scenarios.

Procedure

1. Return the file you transferred to its original location.
2. As the user ftuser, enter the **fteCreateTransfer** command to initiate the transfer of your file from C:\start\myfile.txt to C:\end\myfile.txt :

```
fteCreateTransfer -sa AGENT1 -sm hubQM -sd delete -da AGENT2 -dm hubQM -w -dd C:\end\ C:\start\myfile.txt
```

- -sa AGENT1 defines the source agent (that is, the agent from which the file is transferred) to be AGENT1.
- -sm hubQM defines the queue manager to which the source agent, AGENT1, connects.
- -sd delete specified that the source file is deleted after a successful transfer.
- -da AGENT2 defines the destination agent (that is, the agent to which the file is transferred) to be AGENT2.
- -w requests the **fteCreateTransfer** command to wait for confirmation of its success.
- -dd C:\end\ defines the destination directory to be C:\end\.
- C:\start\myfile.txt defines the file to transfer.

For more information, see `fteCreateTransfer` (create new file transfer).

3. Confirm the file transfer is successful by inspecting the file transfer log in IBM WebSphere MQ Explorer, or by manually inspecting the file system.

What to do next

You might want to explore more features of file transfer capability through external media. See:

- “Two computer file transfer using the scripts” on page 152

- Securing your environment further. Your own requirements might mandate a different access model to the one used in this scenario. For more information about best practices in this area, see *Securing WebSphere MQ File Transfer Edition V7* .

Basic file transfer in detail

You can transfer files in a number of different ways using IBM WebSphere MQ Version 7.5. Read the topics in this section to understand what is covered in this scenario, the reasons why a business might want to follow the scenario, the user roles involved, and an overview of the solution proposed by the scenario.

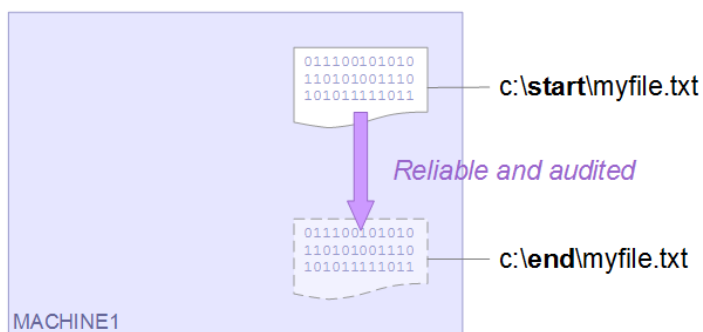
Transferring files with control, reliably, and with auditing can be a fundamental requirement in your enterprise. IBM WebSphere MQ Version 7.5 provides a Managed File Transfer capability as part of its integrated messaging platform. You can use Managed File Transfer capability to integrate files seamlessly into your messaging infrastructure, either through basic file transfers, or fully fledged participants in messaging.

This scenario provides you with a basic understanding of how to integrate files into the simplest IBM WebSphere MQ messaging topology. Work through this basic IBM WebSphere MQ scenario designed to move a file from one location to another. Although this initial scenario is restricted to a single computer, it gives you experience of configuring the environment and forms an important foundation for later scenarios. This scenario demonstrates how to use IBM WebSphere MQ to transfer files across a network, and then shows how the Managed File Transfer component can address real business problems.

You should have a basic understanding of IBM WebSphere MQ, specifically, the notion of a queue manager and basic configuration and administration of IBM WebSphere MQ through to using commands such as `runmqsc` and the IBM WebSphere MQ Explorer.

Overview

IBM WebSphere MQ can be used to initiate and track the transfer of a file from one location to another on a single computer. It gives you the experience of installing, configuring, and using managed file transfer capability in IBM WebSphere MQ Version 7.5, and therefore an understanding of how you can use it to begin to address real business problems with file transfers.



For more details about planning file transfer capability, see *WebSphere MQ Managed File Transfer introduction*.

Planning the solution

Transferring files to or from a computer, file transfer agents, prerequisites, licenses, preparing your computer, and installing and configuring IBM WebSphere MQ for file transfers for the basic file transfer scenario.

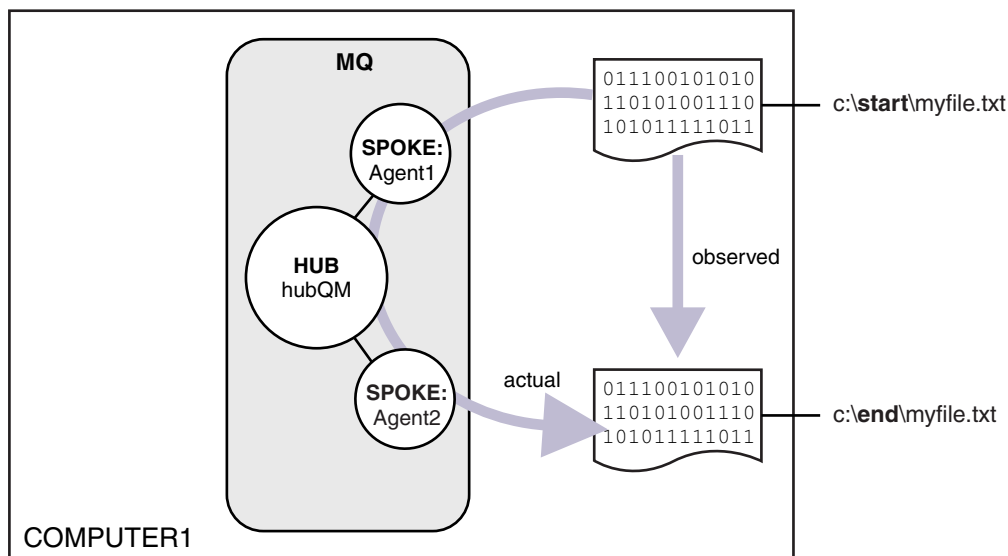
To transfer a file from point A to point B, define a IBM WebSphere MQ queue manager and two file transfer agents. You already understand what a queue manager is, however file transfer agents might be new to you. File transfer agents are Java processes that run on the computer and transfer files to and from other agents. In this scenario, define a file transfer that uses these file transfer agents to move a sample file from one location to another through two mechanisms:

1. By using the graphical IBM WebSphere MQ Explorer.
2. By way of the command line.

The walkthrough assumes that you have a Windows system. For a UNIX system, substitute appropriate paths and commands, and ensure that you have read and write access to all relevant directories. It is also assumed you have a basic understanding of what a queue manager does.

To transfer files to or from a computer, you must have a file transfer agent running on that computer. Every agent connects to a IBM WebSphere MQ queue manager and uses IBM WebSphere MQ to communicate with other agents. For more information, see WebSphere MQ Managed File Transfer topology overview.

This scenario can use a single file transfer agent managing both the source and destination files. However, this example uses two file transfer agents to provide you with a more realistic understanding of how file transfers work in realistic situations that typically span networks.



This diagram shows how IBM WebSphere MQ is used to demonstrate a basic file transfer. Observe the file copied from one location to another.

Prerequisites and licenses

You require the following items:

- A test computer that satisfies the hardware and operating system prerequisites for IBM WebSphere MQ, for details, see , with no existing installation of IBM WebSphere MQ or IBM WebSphere MQ data.
- IBM WebSphere MQ Version 7.5. You can download a trial version from .

Prepare your computer

Ensure your test computer satisfies the requirements for installation of IBM WebSphere MQ Version 7.5, see [Checking requirements](#) .

Ensure that your computer is prepared appropriately for installation, see [Verifying a local installation using the command line](#).

Configure WebSphere MQ for file transfers

1. Create a queue manager hubQM.

Enter the command `crtmqm hubQM`. For more details, see [crtmqm](#) .

2. Start the queue manager hubQM.

Enter the command `strmqm hubQM`. For more details, see [strmqm](#) .

3. Configure your queue manager hubQM to coordinate file transfers

- a. Create the properties files and the coordination queue manager directory for IBM WebSphere MQ by entering the following command:

```
fteSetupCoordination -coordinationQMgr hubQM
```

This command creates properties files and the coordination queue manager directory for IBM WebSphere MQ. In this case, hubQM acts as the coordination queue manager broadcasting audit and file transfer information. If the **fteSetupCoordination** command is not available, it means that you installed WebSphere MQ with the default settings. Installing the extra packages (for example, Managed File Transfer) fixes this issue.

For more information, see [WebSphere MQ Managed File Transfer topology overview and fteSetupCoordination \(set up coordination details\)](#).

- b. Configure hubQM to act as the coordination queue manager by entering the following command:

```
runmqsc hubQM < <filepath from previous statement>
```

- c. Define which queue manager handles file transfer commands, in this case, hubQM.

```
fteSetupCommands -connectionQMgr hubQM
```

For more information, see [fteSetupCommands \(create the command.properties file\)](#).

4. Create your first file transfer agent AGENT1.

- a. Prepare a file transfer agent AGENT1, including MQSC scripts that you must run against the queue manager that the agent connects to, in this case, hubQM by entering the following command:

```
fteCreateAgent -agentName AGENT1 -agentQMgr hubQM
```

For more information, see [fteCreateAgent \(create a WebSphere MQ Managed File Transfer agent\)](#).

- b. Configure hubQM to handle the agent you created.

```
runmqsc hubQM < <location of AGENT1_create.mqsc>
```

The location of the file depends on where you installed IBM WebSphere MQ.

5. Create your second file transfer agent AGENT2.

```
fteCreateAgent -agentName AGENT2 -agentQMgr hubQM
```

```
runmqsc hubQM < <location of AGENT2_create.mqsc>
```

6. Start AGENT1.

```
fteStartAgent AGENT1
```

For more information, see [fteStartAgent \(start a WebSphere MQ Managed File Transfer agent\)](#).

7. Start AGENT2.

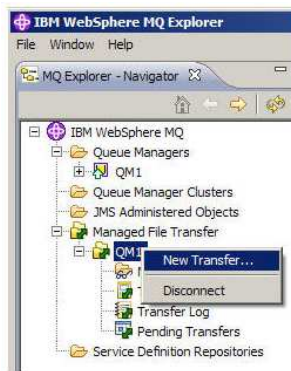
```
fteStartAgent AGENT2
```

Implementing the solution

Implementing the solution in this scenario involves using IBM WebSphere MQ Explorer to define and initiate a file transfer. You can monitor transfer progress, and confirm its success by inspecting the file system.

Procedure

1. Identify source file and target directory:
 - a. Create a sample file to transfer, for example, C:\start\myfile.txt
 - b. Identify an existing directory to transfer this file to, for example, C:\end\
2. Start IBM WebSphere MQ Explorer. Start the program from the Start menu (or equivalent), or run the command **MQExplorer**. For more details, see [Launching IBM WebSphere MQ Explorer](#).
3. Click **Managed file transfer** in the IBM WebSphere MQ Explorer navigator, right-click on **QM**, and select **New Transfer** to start the New Transfer wizard.



4. Select **AGENT1** as the source agent in the **From** section:

From:

Agent: AGENT1

Type: File

File:

Include subdirectories

5. Enter the path to the file you created earlier, for example, C:\start\myfile.txt.

From:

Agent: AGENT1

Type: File

File: C:\Users\Ben Bakowski\Desktop\FTETEST\START\myDemoTransfer.txt

Include subdirectories

6. Select **AGENT2** as the destination agent in the **To** section.
7. Enter the destination directory you identified earlier, for example, C:\end\.

To:

Agent: AGENT2

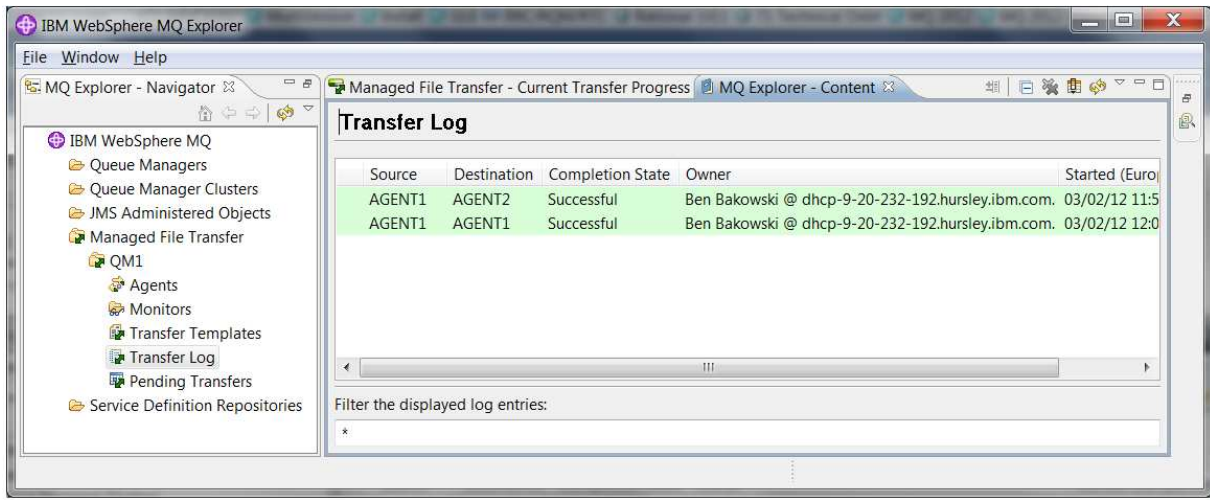
Type: File

Directory: C:\Users\Ben Bakowski\Desktop\FTETEST\END

File name: myDemoTransfer.txt

Overwrite files on the destination file system that have the same name

8. Click **Finish now** and the transfer starts.
9. You can monitor transfer progress in the **Current transfer progress** tab in IBM WebSphere MQ Explorer:



10. You can also inspect the file system manually to confirm that the new file exists, for example, C:\end\myfile.txt.

Using the command line to transfer a file:

The flexibility of file transfers can be demonstrated by using the command line to initiate a transfer. Although out of scope for this scenario, you can build on these principles and the Ant scripting technology to define and implement much more powerful file transfer scenarios.

Procedure

1. Delete the transferred file from the earlier demonstration, for example, C:\end\myfile.txt.
2. Use the **fteCreateTransfer** command to initiate the transfer of your file from C:\start\myfile.txt to C:\end\myfile.txt:

```
fteCreateTransfer -sa AGENT1 -sm hubQM -da AGENT2 -dm hubQM -w -dd C:\end\ C:\start\myfile.txt
```

- -sa AGENT1 defines the source agent (that is, the agent from which the file is transferred) to be AGENT1.
- -sm hubQM defines the queue manager to which the source agent, AGENT1, connects.
- -da AGENT2 defines the destination agent (that is, the agent to which the file is transferred) to be AGENT2.
- -w requests the **fteCreateTransfer** command to wait for confirmation of its success.
- -dd C:\end\ defines the destination directory to be C:\end\.
- C:\start\myfile.txt defines the file to transfer.

For more information, see **fteCreateTransfer** (create new file transfer).

3. Confirm the file transfer is successful by inspecting the file transfer log in IBM WebSphere MQ Explorer, or by manually inspecting the file system.

What to do next

You might want to explore more features of file transfer capability through external media. See:

- “Two computer file transfer using the scripts” on page 152
- Securing your environment further. Your own requirements might mandate a different access model to the one used in this scenario. For more information about best practices in this area, see http://www.ibm.com/developerworks/websphere/library/techarticles/0902_wyatt/0902_wyatt.html.

Two computer file transfer using the scripts

Extends the basic file transfer using the scripts scenario to integrate file transfers into a multiple-computer IBM WebSphere MQ messaging topology.

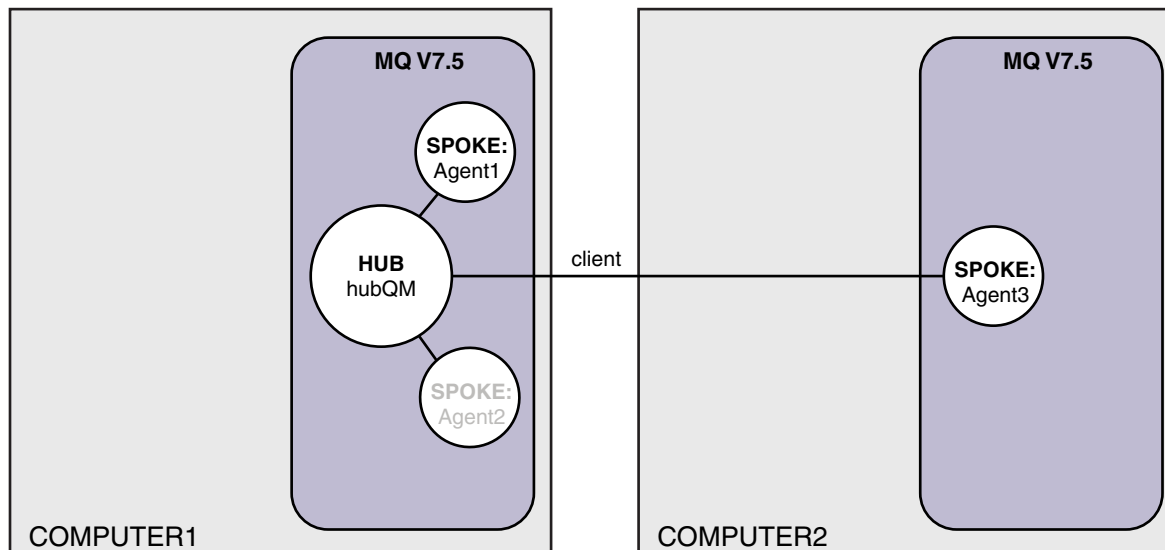
After building a simple demonstration of the Managed File Transfer capability (see “Basic file transfer using the scripts” on page 139), you are now familiar with the basic principles behind managed file transfers. You can recognize that on a single computer this capability offers little benefit, so now you extend the scenario to explore how to integrate file transfers into a multiple-computer IBM WebSphere MQ messaging topology.

Begin to access the benefits offered by the underpinning IBM WebSphere MQ technology, that is, reliable once-and-only delivery of files. To achieve this, the topology is extended to include a second computer that participates in file transfers. In this scenario, you install and configure the separately available Managed File Transfer Agent, and you begin to understand security considerations that apply to a multiple computer file transfer topology. This scenario concludes by demonstrating a file transfer from one computer to the next, providing strong foundations for the next scenario in which you can add auditing capabilities, thus showing why this is a managed file transfer capability.

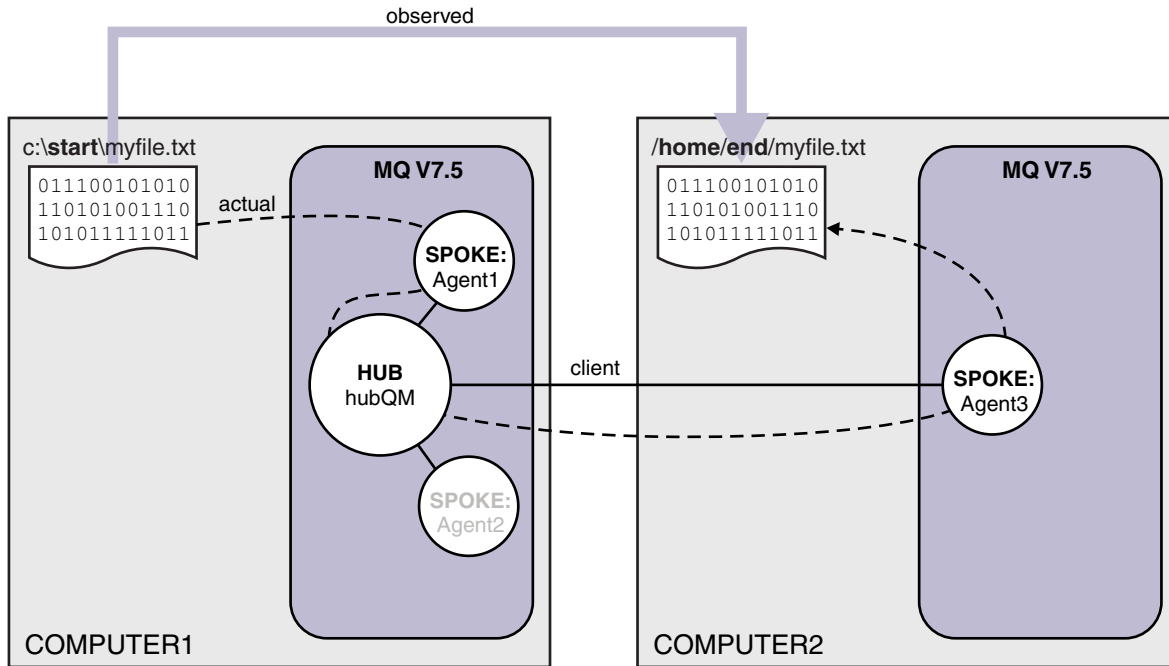
It is assumed that you completed the tasks given in “Basic file transfer using the scripts” on page 139.

Overview

In this scenario, the hub and spoke topology configured in the “Basic file transfer using the scripts” on page 139 scenario is extended to include a second computer, computer 2. Computer 2 has the Managed File Transfer Agent and Command Line Tools installed; there is not the prerequisite of a local IBM WebSphere MQ server installation on this second computer. Such a model is popular in hub and spoke solutions, where multiple Managed File Transfer Agents interact over client connections to a centralized IBM WebSphere MQ queue manager without the need for IBM WebSphere MQ server installations, and associated licenses, on each spoke. Other topologies are supported, and your own topology depends on your specific needs and licensing or entitlement. AGENT2 is retired in favor of using AGENT1 alone to handle transfers on this computer.



This diagram shows hub and spoke topology spanning two computers. AGENT1 and AGENT2 are configured to use bindings to the (local) hub, while AGENT3 connects over a client connection. AGENT2 is disabled because it plays no further role in the scenario. AGENT1 handles all file transfer activity on computer 1. Ensure that you understand the security model and its limitations, for details, see the “Basic file transfer using the scripts” on page 139 scenario. When configured, this topology is used to transfer a file from computer 1 to computer 2.



This diagram shows the file transfer route that is demonstrated. Again, the underlying file transfer takes place over reliable and active IBM WebSphere MQ connections. In this example, assume that computer 1 is a Windows computer and that computer 2 is a Linux computer. You might want to use alternative platforms and architectures, for a full list of supported platforms, see <http://www.ibm.com/support/docview.wss?uid=swg27006467#7.1>.

Planning the solution

Describes transferring files to or from a computer, file transfer agents, prerequisites, licenses, preparing your computer, and configuring IBM WebSphere MQ for the two computer file transfers using the scripts scenario.

Prerequisites

You need the following items:

- Computer 1, a working configuration from “Basic file transfer using the scripts” on page 139.
- Computer 2, a second test computer that satisfies the hardware and operating system prerequisites for IBM WebSphere MQ Version 7.5. For more information, see <http://www-01.ibm.com/support/docview.wss?uid=swg27006467>.
- IBM WebSphere MQ Version 7.5. You can download a trial version from <http://www.ibm.com/developerworks/downloads/ws/wmq/>.
- Knowledge of the IP addresses of computer 1 and computer 2, in nnn.nnn.nnn.nnn format.

Install IBM WebSphere MQ Version 7.5

Install IBM WebSphere MQ Version 7.5 server with the following components: Server, IBM WebSphere MQ Explorer, IBM WebSphere MQ Advanced Managed File Transfer Agent, and IBM WebSphere MQ Managed File Transfer Command Line Tools. For details, see *Choosing what to install*.

Decide how you want to administer IBM WebSphere MQ. You can administer IBM WebSphere MQ by:

- Setting up an appropriate environment using the **setmqenv** command. For more information, see *setmqenv*.
- Calling fully qualified IBM WebSphere MQ administrative commands.

Prepare your users and groups

This security model assumes that you have the following groups and users:

Groups

- **mqm**
Automatically created as part of the IBM WebSphere MQ installation. Members of this group can administer IBM WebSphere MQ and its resources.
- **FTUSERS**
You must create this group. Members of this group can initiate file transfers.
- **FTAGENTS**
You must create this group. Members of this group can start and stop file transfer agents which are the endpoints that handle the transfers of files in your network.

Users

- **mqmAdmin**
IBM WebSphere MQ administrator, that is, a member of the **mqm** group, or on Windows a member of the Administrators group.
- **ftuser**
You must create this user, and make it a member of the **FTUSERS** group. Do not add this user to the **mqm** group, to avoid the potential for administrative security acts on the queue manager.
- **ftagent**
You must create this user, and make it a member of the **FTAGENTS** group. Do not add this user to the **mqm** group, to avoid the potential for administrative security acts on the queue manager.

Add a new agent AGENT3 as a spoke on computer 2

Prepare computer 2 to support the extended topology for the IBM WebSphere MQ two computer file transfers using the *scripts* scenario.

About this task

Use the *addSpoke* sample script again from computer 1, defining an agent spoke that attaches over a client connection. The **addSpoke** command finishes by presenting you with a set of commands to run from the IBM WebSphere MQ installation on computer 2.

Procedure

1. On computer 1, as user **mqmAdmin**, run the **addspoke** command from IBM WebSphere MQ bin directory
`<MQ_INSTALL_ROOT>\bin.`
`<MQ_INSTALL_ROOT>\mqft\samples\scripts\addSpoke agentName=AGENT3`
`hubQmgr=hubQM connectionMode=CLIENT agentIPAddress=<IP address of computer 2>`
`hubIPAddress=<IP address of computer 1> hubPort=1414`

2. On computer 2, as user mqmAdmin, run the commands output by the **addSpoke** command from computer 1, for example:
 - a. `fteSetupCoordination -coordinationQMGr hubQM-coordinationQMGrHost <IP address of computer 1> -coordinationQMGrPort 1414 -coordinationQMGrChannel FTE.USER.SVRCONN -f`
 - b. `fteSetupCommands -p hubQM -connectionQMGr hubQM -connectionQMGrHost <IP address of computer 1> -connectionQMGrPort 1414 -connectionQMGrChannel FTE.USER.SVRCONN -f`
 - c. `fteCreateAgent -p hubQM -agentName AGENT3 -agentQMGr hubQM -agentQMGrHost <IP address of computer 1> -agentQMGrPort 1414 -agentQMGrChannel FTE.AGENT.SVRCONN -f`

Note you do not need to run the generated MQSC scripts.

3. On computer 2, as user ftagent, start AGENT3
`fteStartAgent -p hubQM AGENT3`
4. On computer 2, as user ftagent, confirm the three agents are available (starting the agents on computer 1 under user ftagent if needed).

`fteListAgents -p hubQM`

You see the following output:

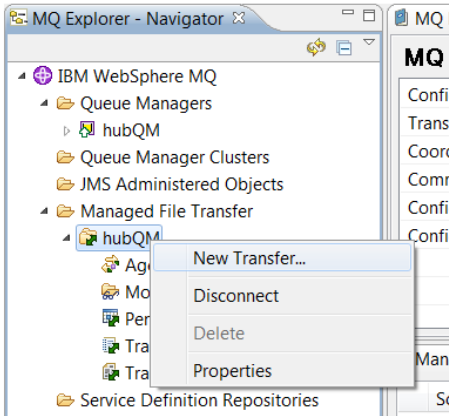
Agent Name:	Queue Manager Name:	Status:
AGENT1	hubQM	READY
AGENT2	hubQM	READY
AGENT2	hubQM	READY

Implementing the solution

Start the demonstration for the two computer file transfer using the scripts scenario by using IBM WebSphere MQ to transfer a file. Monitor the transfer progress, and confirm its success by inspecting the file system.

Procedure

1. Identify source file and target directory:
 - a. Create a sample file on computer 1 to transfer, for example, `C:\start\myfile.txt`
 - b. Identify an existing directory on computer 2 to transfer this file to, for example, `C:\end\`. Ensure user ftagent has write access to the directory.
2. Start IBM WebSphere MQ Explorer on computer 1 as user mqmAdmin. Note as with the earlier scenario, do this as an IBM WebSphere MQ administrator in the interests of focusing the scenario around file transfers, rather than configuring IBM WebSphere MQ Explorer. Start the program from the Start menu (or equivalent), or run the command **MQExplorer**. For more details, see *Launching WebSphere MQ Explorer*.
3. Under the **Managed file transfer** section in the IBM WebSphere MQ navigator, right-click on hubQM and select **New Transfer** to start the New Transfer wizard.



4. Use the menu to select AGENT1 as the source agent and AGENT3 as the destination agent:

 A screenshot of the 'New Transfer' dialog box. It has two main sections: 'Source agent' and 'Destination agent'. In the 'Source agent' section, the 'Name' dropdown is set to 'AGENT1' and the 'Type' is 'Standard'. In the 'Destination agent' section, the 'Name' dropdown is set to 'AGENT3' and the 'Type' is 'Standard'.

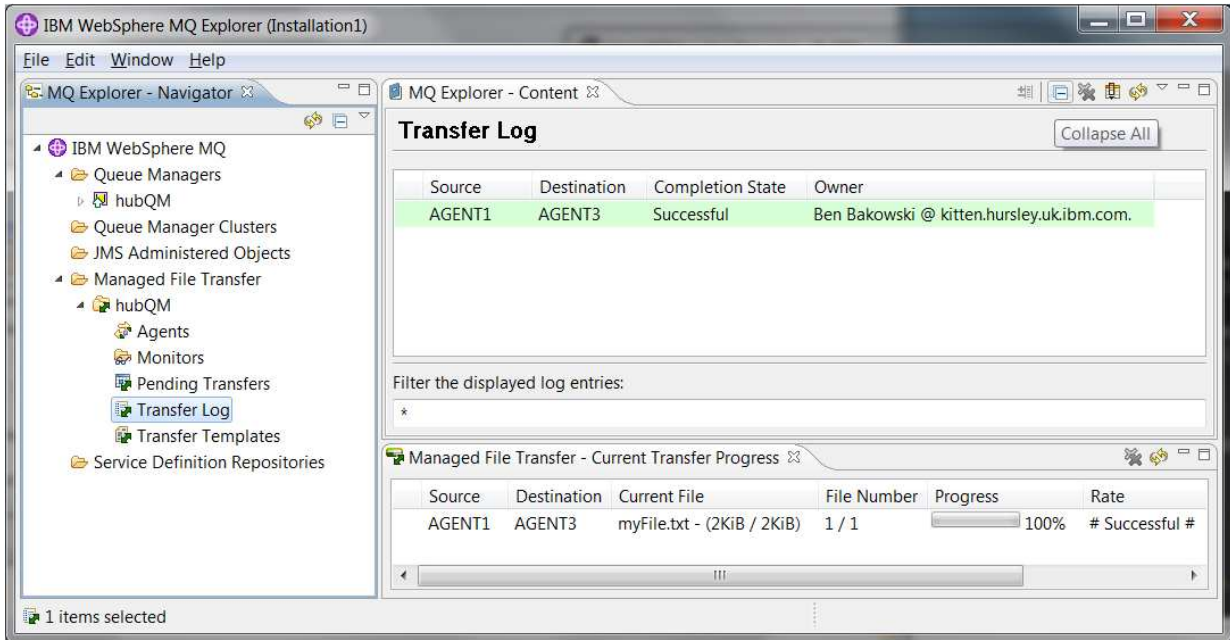
5. Click **Next** and then click **Add...** and for the source, enter the path to the file you want to transfer, for example, C:\start\myfile.txt. Ensure the **Remove source file if the transfer is successful** check box is checked.

 A screenshot of the 'Source' configuration dialog box. It shows the following settings: 'Agent: AGENT1', 'Type: Standard agent', and a 'Type' dropdown set to 'File'. The 'File name' field contains 'C:\start\myfile.txt' and has a 'Browse...' button next to it. At the bottom, the checkbox 'Remove source file if the transfer is successful' is checked.

6. Enter the destination directory. In this scenario, this destination is on a Linux platform, so use the appropriate notation of /home/end/.

 A screenshot of the 'Destination' configuration dialog box. It shows the following settings: 'Agent: AGENT3', 'Type: Standard agent', and a 'Type' dropdown set to 'Directory'. The 'Directory' field contains '/home/end' and has a 'Browse...' button next to it. At the bottom, the checkbox 'Overwrite files if present' is unchecked.

7. Click **OK** and then **Finish** and the transfer starts.
8. You can monitor transfer progress in the **Current Transfer Progress** tab in IBM WebSphere MQ Explorer.



- :
9. You can also inspect the file system manually to confirm that the new file exists, for example, `/home/end/myfile.txt`.

Using the command line to transfer a file:

The flexibility of file transfers can be demonstrated by using the command line to initiate a transfer. Although out of scope for this scenario, you can build on these principles and the Ant scripting technology to define and implement much more powerful file transfer scenarios.

Procedure

1. Delete the transferred file from the earlier demonstration, for example, `/home/end/myfile.txt`, and recreate it in its original location.
2. As use `ftuser`, use the **`fteCreateTransfer`** command from to initiate the transfer of your file from `C:\start\myfile.txt` on computer 1 to `/home/end/myfile.txt` on computer 2:


```
fteCreateTransfer -sa AGENT1 -sm hubQM -da AGENT3 -dm hubQM -w -dd "/home/end/" "C:\start\myfile.txt"
```
3. Confirm the file transfer is successful by inspecting the file transfer log in IBM WebSphere MQ Explorer, or by manually inspecting the file system.

What to do next

For more details about IBM WebSphere MQ security, see [Security](#) .

You might want to explore more features of file transfer capability through external media.

- For details about adding audit capability to provide the managed aspect of managed file transfer, see “Adding audit capability to managed file transfer” on page 165.
- For more details about:
 - Triggering: Moving new files when they appear.
 - Triggering: Configuring a single file's appearance to initiate transfer of multiple files.
 - Scripting transfers using Apache Ant.

See .

- Your own requirements might mandate a different access model to the one used in this scenario. For more information about best practices in securing your environment further, see .

Two computer file transfer in detail

Extends the basic file transfer scenario to integrate file transfers into a multiple-computer IBM WebSphere MQ messaging topology.

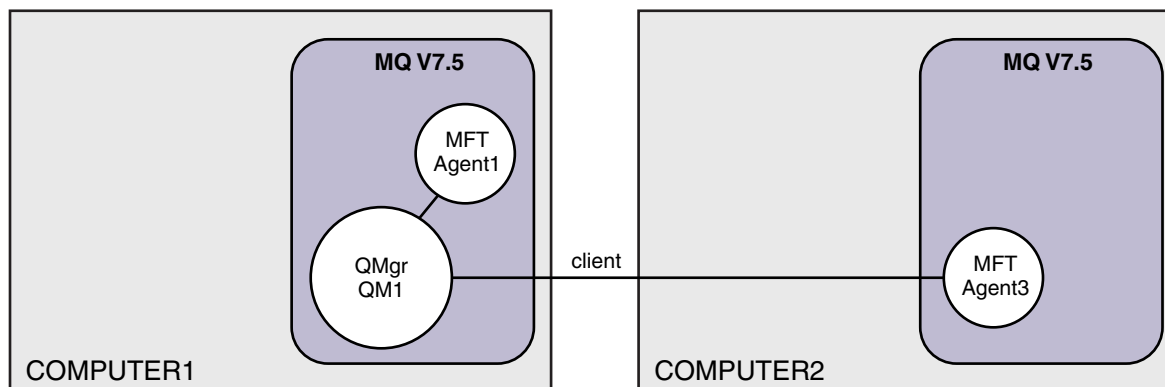
After completing the simple demonstration of the Managed File Transfer capability (see “Basic file transfer in detail” on page 147), you are now familiar with the basic principles behind managed file transfers. You can recognize that on a single computer this capability offers little benefit, so now you extend the scenario to explore how to integrate file transfers into a multiple-computer IBM WebSphere MQ messaging topology.

Begin to access the benefits offered by the underpinning IBM WebSphere MQ technology, that is, reliable once-and-only delivery of files. To achieve this, the topology is extended to include a second computer that participates in file transfers. In this scenario, you install and configure the separately available Managed File Transfer Agent, and you begin to understand security considerations that apply to a multiple computer file transfer topology. This scenario concludes by demonstrating a file transfer from one computer to the next, providing strong foundations for the next scenario in which you can add auditing capabilities, thus showing why this is a managed file transfer capability.

It is assumed that you completed the tasks given in “Basic file transfer in detail” on page 147.

Overview

In this scenario, you continue with the existing Windows computer configured in the basic file transfer scenario. Agent2 is disabled because you use the single agent as the file transfer endpoint on this first computer. On a second computer, you install the Managed File Transfer Agent; a local IBM WebSphere MQ server installation is not a prerequisite on this second computer. Such a model is popular in hub and spoke solutions, where multiple Managed File Transfer Agents interact over client connections to a centralized IBM WebSphere MQ queue manager without the need for IBM WebSphere MQ server installations, and associated licenses, on each spoke. Other topologies are supported, and your own topology depends on your specific needs and licensing or entitlement.

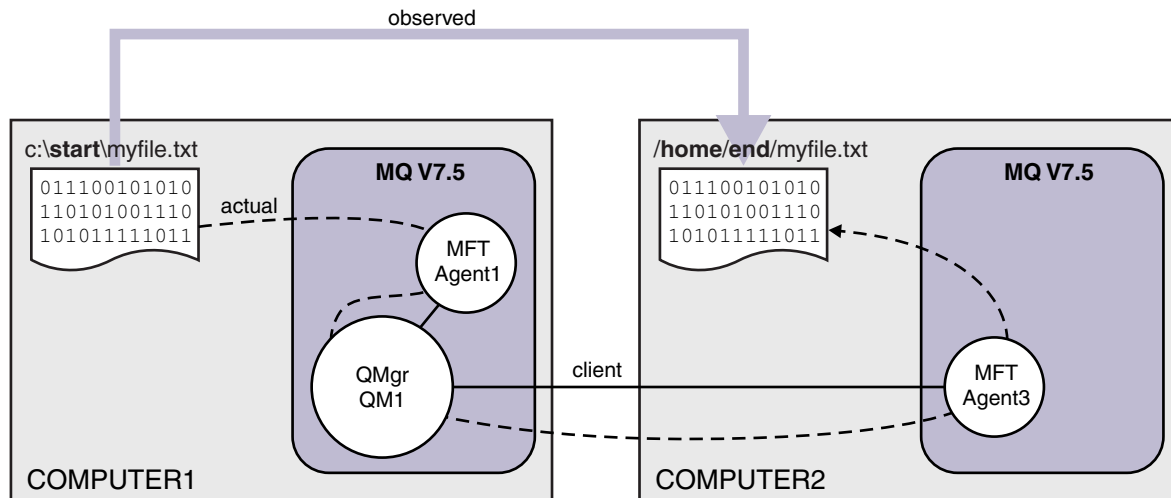


In a file transfer network, a single queue manager is selected to act as a single point in the network to broadcast audit and file transfer information. In the earlier “Basic file transfer in detail” on page 147 scenario, a single queue manager was implicitly used as the coordination queue manager. In this scenario, you continue to use the queue manager QM1 on computer 1 to act as the coordination queue manager, and part of the configuration of computer 2 sets the environment to use this queue manager.

In this topology, the ability to initiate file transfers from computer 2 is not required, therefore do not install the optional Managed File Transfer Command Line Tools component on this second computer.

It is important to note that although some basic security features are considered, the file transfer topology is not secured to a level you might find appropriate. For a discussion on securing IBM WebSphere MQ, and file transfers in particular, see .

When installed on the second computer, you create the actual agent Agent3, and then demonstrate the solution by moving a file from computer 1 to computer 2.



The underlying file transfer takes place over reliable and performing IBM WebSphere MQ connections. The tasks that follow show how to configure and appropriately secure these connections.

In this example, assume that computer 1 is a Windows computer and that computer 2 is a Linux computer. You might want to use alternative platforms and architectures, for a full list of supported platforms, see .

Planning the solution

Planning the two computer file transfer solution, including a description of the appropriate infrastructure, and the groups and users you need to create.

Before you begin

You need the following items:

- A working configuration from the basic file transfer scenario, for details, see “Basic file transfer in detail” on page 147.
- IBM WebSphere MQ Version 7.5. You can download a trial version from .
- A second test computer that satisfies the hardware and operating system prerequisites for IBM WebSphere MQ Version 7.5, for details, see .

In this scenario, the second computer interacts with the queue manager over a client connection channel. Using channel authentication records ensures:

- The incoming connection is authenticated as originating from the new computer that hosts a further managed file transfer agent.
- The incoming request is mapped to a user that has appropriate access to managed file transfer resources.

Sandboxing, a technique to restrict file transfers from manipulating particular areas of the file system, for example, IBM WebSphere MQ configuration files, is not considered.

Procedure

On **both** computers:

1. Create the groups FTEUSERS and FTEAGENTS. For more details, see your operating system instructions.
2. Create the user fteuser, and add it to the FTEAGENTS group.
3. Create the user fteagent, and add it to the FTEUSERS group.

The user fteuser initiates file transfers, and the user fteagent starts and stops agent processes. These users are not members of the mqm (or Windows Administrators) group, and so helps harden the topology against unwanted administration attacks. You can provide more granular security access by defining specific users on an agent basis, for example, fteagent1, fteagent2.

Modify computer 1 to support the extended topology

Prepare computer 1 to support the extended topology for the IBM WebSphere MQ two computer file transfer scenario.

About this task

In this task, you delete Agent2 because it is no longer needed. You then create and start a listener to accept a client connection from the agent created on computer 2 and thus construct a simple IBM WebSphere MQ network across the two computers. It is assumed computer 1 is running the Windows operating system. If you are using a different platform to run this scenario, substitute the appropriate platform-specific commands.

Procedure

1. Stop the agent Agent2.

```
fteStopAgent AGENT2
```

For more details about the **fteStopAgent** command, see fteStopAgent (stop a WebSphere MQ Managed File Transfer agent).

2. Delete the agent Agent2.

```
fteDeleteAgent AGENT2
```

```
runmqsc QM1 < <output>
```

For more details about the **fteDeleteAgent** command, see fteDeleteAgent (delete a WebSphere MQ Managed File Transfer agent).

Configure IBM WebSphere MQ security so that a new file transfer agent configured on computer 2 can interact with the coordination queue manager, QM1. This new agent connects into QM1 over the existing SYSTEM.DEF.SVRCONN channel. Your own security needs might differ, for further details on hardening this topology, see What to do next.

3. Start the MQSC interface for QM1.

```
runmqsc QM1
```

4. Create two channels to handle incoming requests from users and agents.

```
DEFINE CHANNEL(FTE.USER.SVRCONN) CHLTYPE(SVRCONN)
```

```
DEFINE CHANNEL(FTE.AGENT.SVRCONN) CHLTYPE(SVRCONN)
```

5. Create a channel authentication record to allow a connection from computer 2 into QM1, assigning the created user.

```
SET CHLAUTH('FTE.USER.SVRCONN') TYPE(ADDRESSMAP) ADDRESS('<IP address of computer2>') USERSRC (MAP) MCAUSER('fteuser' D
```

```
SET CHLAUTH('FTE.AGENT.SVRCONN') TYPE(ADDRESSMAP) ADDRESS('<IP address of computer2>') USERSRC(MAP) MCAUSER('fteagent')
```

For more details, see Channel authentication records.

The goal of this scenario is not to lock down and harden the topology, but to demonstrate a basic file transfer. The implemented security model supports this demonstration, but you must understand your own security threats and take appropriate actions where necessary. For discussions of options to consider, see *What to do next*.

6. Identify a free port that can be used for network communications with IBM WebSphere MQ. Define a listener LISTENER1 to use this free port, for example, 1414.
`DEFINE LISTENER(LISTENER1) TRPTYPE(TCP) CONTROL(QMGR) PORT(1414)`
7. Start the listener LISTENER1.
`START LISTENER(LISTENER1)`
8. Stop the MQSC interface for QM1.
`end`
9. Check that the FTEAGENTS and FTEUSERS groups have appropriate access to IBM WebSphere MQ objects to do file transfer actions for an agent AGENT3 to be created on computer 2. You might want to tailor this configuration to suit your own security requirements.
 - a. `setmqaut -m QM1 -t qmgr -g FTEAGENTS +connect +inq`
 - b. `setmqaut -m QM1 -t qmgr -g FTEUSERS +connect`
 - c. `setmqaut -m QM1 -n "SYSTEM.FTE" -t q -g FTEAGENTS +get +put`
 - d. `setmqaut -m QM1 -n "SYSTEM.FTE.COMMAND.AGENT1" -t q -g FTEUSERS +put`
 - e. `setmqaut -m QM1 -n "SYSTEM.FTE.COMMAND.AGENT1" -t q -g FTEAGENTS +setid +get +put`
 - f. `setmqaut -m QM1 -n "SYSTEM.FTE.COMMAND.AGENT3" -t q -g FTEUSERS +put`
 - g. `setmqaut -m QM1 -n "SYSTEM.FTE.COMMAND.AGENT3" -t q -g FTEAGENTS +setid +get +put`
 - h. `setmqaut -m QM1 -n "SYSTEM.FTE.DATA.AGENT1" -t q -g FTEAGENTS +get +put`
 - i. `setmqaut -m QM1 -n "SYSTEM.FTE.DATA.AGENT3" -t q -g FTEAGENTS +get +put`
 - j. `setmqaut -m QM1 -n "SYSTEM.FTE.EVENT.AGENT1" -t q -g FTEAGENTS +get +put`
 - k. `setmqaut -m QM1 -n "SYSTEM.FTE.EVENT.AGENT3" -t q -g FTEAGENTS +get +put`
 - l. `setmqaut -m QM1 -n "SYSTEM.FTE.REPLY.AGENT1" -t q -g FTEAGENTS +get +put`
 - m. `setmqaut -m QM1 -n "SYSTEM.FTE.REPLY.AGENT3" -t q -g FTEAGENTS +get +put`
 - n. `setmqaut -m QM1 -n "SYSTEM.FTE.STATE.AGENT1" -t q -g FTEAGENTS +get +put +inq`
 - o. `setmqaut -m QM1 -n "SYSTEM.FTE.STATE.AGENT3" -t q -g FTEAGENTS +get +put +inq`
 - p. `setmqaut -m QM1 -n "SYSTEM.FTE" -t topic -g FTEUSERS +sub`
 - q. `setmqaut -m QM1 -n "SYSTEM.FTE" -t topic -g FTEAGENTS +pub +sub`
 - r. `setmqaut -m QM1 -n "SYSTEM.DEFAULT.MODEL.QUEUE" -t q -g FTEUSERS +dsp +browse +get +put`
 - s. `setmqaut -m QM1 -n "SYSTEM.DEFAULT.MODEL.QUEUE" -t q -g FTEAGENTS +dsp +browse +get +put`

For more details about the **setmqaut** command, see **setmqaut**.

For more details about granting authority to groups, see *Group authorities for resources specific to WebSphere MQ Managed File Transfer*.

Prepare computer 2 for file transfers

Describes preparing computer 2 for file transfers for the IBM WebSphere MQ two computer file transfer scenario.

About this task

This task assumes computer 2 is running the Linux operating system. If you are using a different platform to run this scenario, you must substitute the appropriate platform-specific commands.

Procedure

1. Install IBM WebSphere MQ on each computer together with the Managed File Transfer Agent component, and any appropriate prerequisite components, for example, Managed File Transfer Command Line Tools. For details, see [Choosing what to install](#) .

This step assumes that you are using a clean computer with no previous installations of IBM WebSphere MQ or IBM WebSphere MQ File Transfer Edition installed. If not, you must determine whether coexistence is supported and adjust the installation mechanism or configuration of environments appropriately, for details, see [Multiple installations](#) .

Installing the Managed File Transfer Command Line Tools component gives you the ability to define and initiate transfers from computer 2 in addition to computer 1, as demonstrated in this scenario.

2. As a user in the mqm group, configure file transfers to QM1 on computer 1 as a coordination manager. Enter the following commands:

```
fteSetupCoordination -coordinationQMGr QM1 -coordinationQMGrHost <computer1_hostname>
-coordinationQMGrPort 1414 -coordinationQMGrChannel FTE.USER.SVRCONN
```

You do not have to run the generated MQSC script because you ran it when you configured computer 1.

```
fteSetupCommands -connectionQMGr QM1 -connectionQMGrHost <computer1_hostname>
-connectionQMGrPort 1414 -connectionQMGrChannel FTE.USER.SVRCONN
```

For more information, see:

- [WebSphere MQ Managed File Transfer topology overview](#)
- [fteSetupCoordination](#) (set up coordination details)
- [fteSetupCommands](#) (create the command.properties file)

3. List the agents registered with QM1 to ensure that your configuration for client connectivity is correct. Enter the following command:

```
fteListAgents
```

You will see the following output:

Agent Name:	Queue Manager Name:	Status:
AGENT1	QM1	Ready

4. Create the second file transfer agent AGENT3:

```
fteCreateAgent -agentName AGENT3 -agentQMGr QM1 -agentQMGrHost <computer1_hostname> -agentQMGrPort 1414 -agentQMGrChan
```

Switch to computer 1 and enter the following command:

```
runmqsc QM1 < <AGENT3_create.mqsc>
```

5. Switch to computer 2, and as user fteagent start your new file transfer agent AGENT3.

```
fteStartAgent AGENT3
```

6. Optional: Switch to computer 1, and restart AGENT1 as the user fteagent rather than the privileged IBM WebSphere MQ administrator user used in the first scenario.

7. Check that your configuration for client connectivity is correct by listing the agents registered with QM1.

```
fteListAgents
```

You will see the following output:

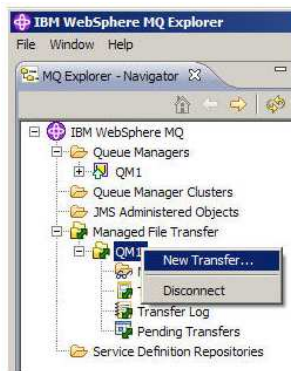
Agent Name:	Queue Manager Name:	Status:
AGENT1	QM1	READY
AGENT3	QM1	READY

Implementing the solution

Implementing the solution in this scenario involves using IBM WebSphere MQ Explorer to define and initiate a file transfer. You can monitor transfer progress, and confirm its success by inspecting the file system.

Procedure

1. Identify source file and target directory:
 - a. Create a sample file on computer 1 to transfer, for example, `C:\start\myfile.txt`.
 - b. Identify an existing directory on computer 2 to transfer this file to, for example, `/home/end/`. Ensure the user that started the agent has write access to that directory.
2. Start IBM WebSphere MQ Explorer on computer 1. Start the program from the Start menu (or equivalent), or run the command **MQExplorer**. For more details, see [Launching WebSphere MQ Explorer](#).
3. Click **Managed file transfer** in the IBM WebSphere MQ Explorer navigator, right-click on **QM**, and select **New Transfer** to start the New Transfer wizard.



4. Select **AGENT1** as the source agent in the **From** section:

From:

Agent:

Type:

File:

Include subdirectories

5. Enter the path to the file you created earlier, for example, `C:\start\myfile.txt`.

From:

Agent:

Type:

File:

Include subdirectories

6. Select **AGENT2** as the destination agent in the **To** section.
7. Enter the destination directory you identified earlier, for example, `/home/end/`.

To:

Agent: AGENT2

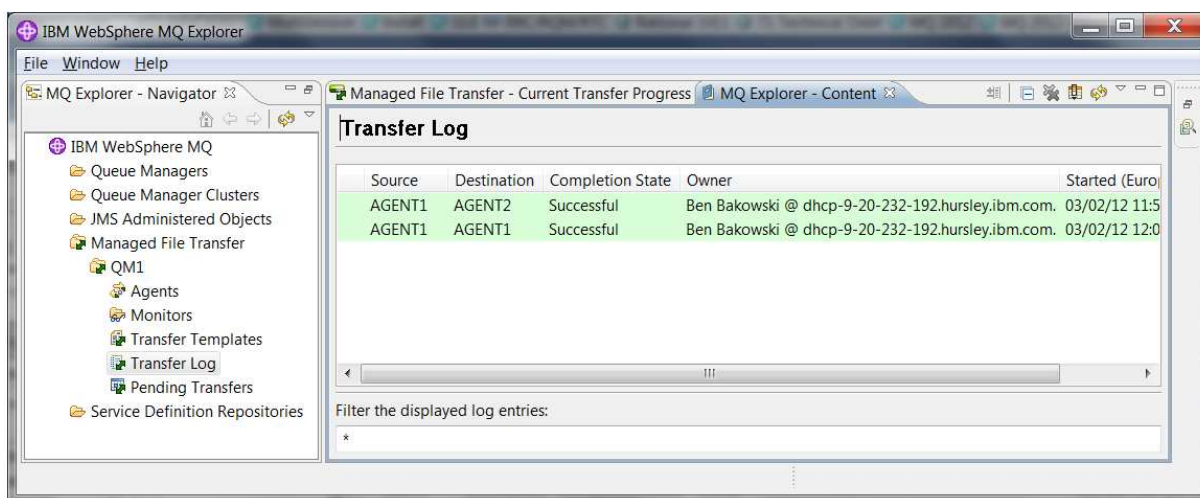
Type: File

Directory: /home/ben/FTETEST/END/

File name: myDemoTransfer.txt

Overwrite files on the destination file system that have the same name

8. Click **Finish now** and the transfer starts.
9. You can monitor transfer progress in the **Current transfer progress** tab in IBM WebSphere MQ Explorer:



10. You can also inspect the file system of computer 2 manually to confirm that the new file exists, for example, /home/end/myfile.txt.

Using the command line to transfer a file:

In this section, the flexibility of file transfers starts to be demonstrated by showing you how to use the command line to initiate a transfer. Although out of scope for this scenario, you can build on these principles and the Ant scripting technology to define and implement much more powerful file transfer scenarios.

Procedure

1. Delete the transferred file from the earlier demonstration, for example, /home/end/myfile.txt.
2. As user fteuser, enter the **fteCreateTransfer** command to initiate the transfer of your file from C:\start\myfile.txt on computer 1 to /home/end/myfile.txt on computer 2:

```
fteCreateTransfer -sa AGENT1 -sm QM1 -da AGENT3 -dm QM1 -w -dd "/home/end/" "C:\start\myfile.txt"
```
3. Confirm the file transfer is successful by inspecting the file transfer log in IBM WebSphere MQ Explorer, or by manually inspecting the file system.

What to do next

For more details about IBM WebSphere MQ security, see [Security](#) .

You might want to explore more features of file transfer capability through external media.

- For details about adding audit capability to provide the managed aspect of managed file transfer, see [“Adding audit capability to managed file transfer.”](#)
 - For more details about:
 - Triggering: Moving new files when they appear.
 - Triggering: Configuring a single file's appearance to initiate transfer of multiple files.
 - Scripting transfers using Apache Ant.
- See [. See .](#)
- Your own requirements might mandate a different access model to the one used in this scenario. For more information about best practices in securing your environment further, see [. For more information about best practices in securing your environment further, see .](#)

Adding audit capability to managed file transfer

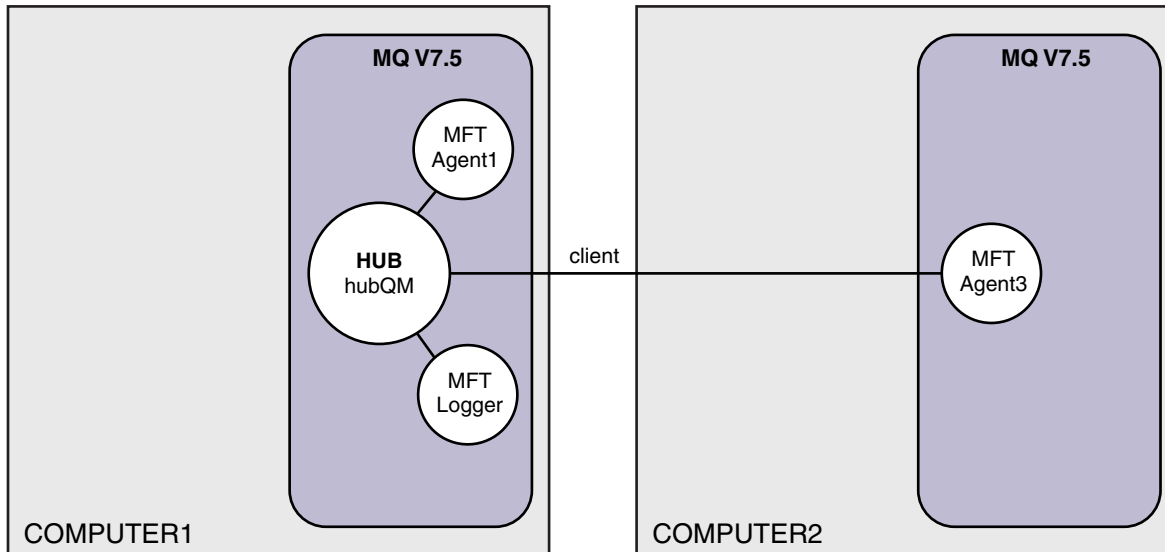
Use this scenario to configure a logger, and how to use this capability to provide an audit trail.

You built a demonstration of the Managed File Transfer capability across computers through the two previous scenarios, [“Basic file transfer in detail”](#) on page 147 and [“Two computer file transfer in detail”](#) on page 158, and you are familiar with configuring a file transfer topology. You see how IBM WebSphere MQ Version 7.5 provides capabilities to log and audit file transfers thus providing the managed aspect of Managed File Transfer.

It is assumed that you completed the second managed file transfer scenario according to the instructions in the [“Two computer file transfer in detail”](#) on page 158 scenario.

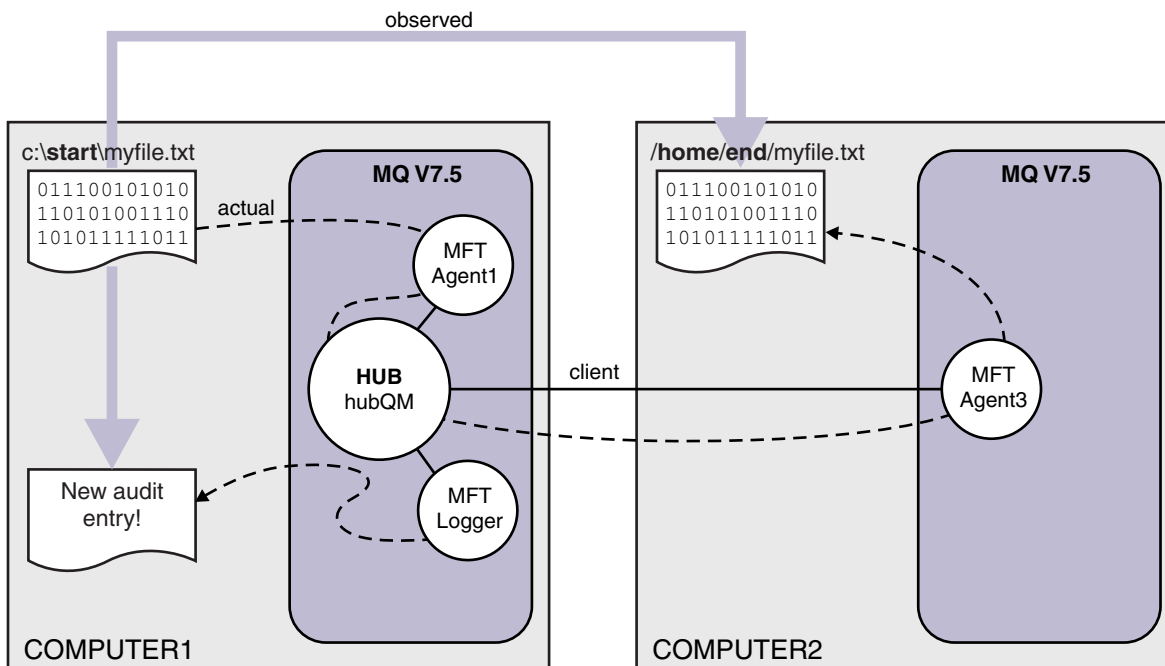
Overview

In this scenario, you continue with the existing Windows and Linux topology you configured in [“Two computer file transfer in detail”](#) on page 158, and you enable file logging capability. File logging capability does not require installation of any other components or products, and therefore this scenario does not require you to consider licensing or entitlement changes from the two computer file transfer scenario.



You can also implement a database logger, which might be more appropriate in a production environment, for example, for scalability and failover. However, to keep this scenario simple and without the need for installation of other products, use the fully supported file logger. No further security aspects are considered. Because this capability might be providing audit information, you might want to consider hardening security around this.

In this scenario, a file transfer is initiated and details are captured in a log file.



Planning the solution

Describes prerequisites and license requirements for the IBM WebSphere MQ adding audit capability to managed file transfer scenario.

In this scenario, you create a logger to audit file transfers. The logger finds most parallels with an agent process, and so you can use the existing `ftagent` user to start and stop the logger. You might want to create your own user or group to manage the logger.

Prerequisites and licenses

You need a working configuration from the “Two computer file transfer in detail” on page 158 scenario.

Implementing the solution

Modifying computer 1 to configure a file logger for the IBM WebSphere MQ adding audit capability to the managed file transfer scenario.

Procedure

1. Check that the `ftagent` group has appropriate access to IBM WebSphere MQ objects when running the logger process. Enter the following commands:

```
setmqaut -m hubQM -n "SYSTEM.FTE.LOG.RJCT.MYFILELOGGER" -t q -g FTAGENTS +put
```

```
setmqaut -m hubQM -n "SYSTEM.FTE.LOG.CMD.MYFILELOGGER" -t q -g FTAGENTS +get
```

For more details, see *Authorities for the database logger*.

2. As a IBM WebSphere MQ administrator, create a file logger, use `hubQM` as the logger queue manager.

```
fteCreateLogger -loggerType FILE -loggerQMgr hubQM -fileLoggerMode LINEAR -fileSize 5MB myFileLogger
```

```
runmqsc hubQM < <MYFILELOGGER_create.mqsc>
```

For more details, see `fteCreateLogger` (create a WebSphere MQ Managed File Transfer logger). You might want to consider the use of a database logger in production.

3. As the user `ftagent`, start the logger.

```
fteStartLogger MYFILELOGGER
```

4. Confirm the logger started.

To confirm the logger started, inspect the file system. After you configured the file logger with the commands above, the logs can be found in `<MQ INSTALL>/mqft/logs/hubQM/loggers/MYFILELOGGER/logs`. Confirm `output0.log` contains the message "BFGDB0023I: The logger has completed startup activities and is now running."

5. Delete the transferred file from the earlier demonstration, for example, `/home/end/myfile.txt`.

6. As user `ftuser`, use the `fteCreateTransfer` (create new file transfer) command from computer 1 to start the transfer of your file from `C:\start\myfile.txt` on computer 1 to `/home/end/myfile.txt` on computer 2. Enter the following command:

```
fteCreateTransfer -sa AGENT1 -sm hubQM -da AGENT3 -dm hubQM -w -dd "/home/end/" "C:\start\myfile.txt"
```

7. Confirm that the logger captures this transfer, and understand the contents of the log entry.

- a. Open the file `<MQ INSTALL>/mqft/logs/hubQM/loggers/MYFILELOGGER/MYFILELOGGER-XXXXXXXXX.log`

- b. The log entry shows the transfer that you initiated, including the file source and destination locations, and date, time, and requestor ID. For example:

```
2012-03-23T16:42:21;414d5120514d312020202020202020207a556b4f2000aa03;[TSTR]; ;  
AGENT1;hubQM;STANDARD;AGENT3;hubQM;User;;;com.ibm.wmqfte.SourceAgent=AGENT1,  
com.ibm.wmqfte.DestinationAgent=AGENT3, com.ibm.wmqfte.MqmdUser=User,  
com.ibm.wmqfte.OriginatingUser=User, com.ibm.wmqfte.OriginatingHost=  
dhcp-9-10-123-123.hursley.ibm.com., com.ibm.wmqfte.TransferId=  
414d5120514d312020202020202020207a556b4f2000aa03, com.ibm.wmqfte.Priority=0;
```

```
2012-03-23T16:42:21;414d5120514d312020202020202020207a556b4f2000aa03;[TPRO];0 ;  
C:\start\myfile.txt;51447;file;leave ;;;;/home/end/myfile.txt;51447;file;
```

;;;;;;;;;

```
2012-03-23T16:42:21;414d5120514d31202020202020202020207a556b4f2000aa03;[TCOM];0 ;
AGENT1;hubQM;STANDARD;AGENT3;hubQM;STANDARD;User;;BFGRP00321: The file transfer
request has successfully completed.;com.ibm.wmqfte.SourceAgent=AGENT1,
com.ibm.wmqfte.DestinationAgent=AGENT3, com.ibm.wmqfte.MqmdUser=User,
com.ibm.wmqfte.OriginatingUser=User, com.ibm.wmqfte.OriginatingHost=
dhcp-9-20-123-123.hursley.ibm.com.,
com.ibm.wmqfte.TransferId=414d5120514d31202020202020202020207a556b4f2000aa03,
com.ibm.wmqfte.Priority=0;
```

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.

To view glossaries for other IBM products, go to www.ibm.com/software/globalization/terminology.

“A” on page 171 “B” on page 171 “C” on page 172 “D” on page 177 “E” on page 179 “F” on page 180 “G” on page 182 “H” on page 182 “I” on page 183 “J” on page 185 “K” on page 185 “L” on page 186 “M” on page 187 “N” on page 191 “O” on page 192 “P” on page 194 “Q” on page 197 “R” on page 198 “S” on page 200 “T” on page 206 “U” on page 208 “V” on page 209 “W” on page 209 “X” on page 210

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 and topics to be advertised among them for load balancing and redundancy.

cluster queue

A local queue that is hosted by a cluster queue manager, and defined as a target for messages being put from an application connected to any queue manager within the cluster. All applications retrieving messages must be locally connected.

cluster queue manager

A queue manager that is a member of a cluster. A queue manager can be a member of more than one cluster.

cluster-receiver channel (CLUSRCVR)

A channel on which a cluster queue manager can receive messages from other queue managers in the cluster, and cluster information from the repository queue managers.

cluster-sender channel (CLUSDR)

A channel on which a cluster queue manager can send messages to other queue managers in the cluster, and cluster information to the repository queue managers.

cluster topic

An administrative topic that is defined on a cluster queue manager and made available to other queue managers in the cluster.

cluster transmission queue

A transmission queue that holds all messages from a queue manager destined for another queue manager that is in the same cluster. The queue is called SYSTEM.CLUSTER.TRANSMIT.QUEUE.

CMS key database

A CMS key database is the format of the Database supported by Windows systems, UNIX systems, Linux, and the clients of those platforms. Files ending with .kdb are CMS format. The .kdb files contain the certificates and the keys.

coded character set identifier (CCSID)

A 16-bit number that includes a specific set of encoding scheme identifiers, character set identifiers, code page identifiers, and other information that uniquely identifies the coded graphic-character representation.

coexistence

The ability of two or more different versions of 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**identity 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 (MQXR) 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.

OCSF 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.

proxy subscription

A proxy subscription is a subscription made by one queue manager for topics published on another queue manager. A proxy subscription flows between queue managers for each individual topic string that is subscribed to by a subscription. You do not create proxy subscriptions explicitly, the queue manager does so on your behalf.

PTF See program temporary fix.

public key

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 (MQXR) service

An MQ service that handles the server half of the MQTT protocol (see MWTT Server). The telemetry (MQXR) 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 (MQXR) 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 (MQXR) 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.

Accessibility features for IBM WebSphere MQ

Accessibility features assist users who have a disability, such as restricted mobility or limited vision, to use information technology content successfully.

Accessibility features

IBM WebSphere MQ includes the following major accessibility features:

- Keyboard-only operation

- Operations that use a screen reader

IBM WebSphere MQ uses the latest W3C Standard, WAI-ARIA 1.0 (<http://www.w3.org/TR/wai-aria/>), to ensure compliance to US Section 508 (<http://www.access-board.gov/guidelines-and-standards/communications-and-it/about-the-section-508-standards/section-508-standards>), and Web Content Accessibility Guidelines (WCAG) 2.0 (<http://www.w3.org/TR/WCAG20/>). To take advantage of accessibility features, use the latest release of your screen reader in combination with the latest web browser that is supported by this product.

The IBM WebSphere MQ online product documentation in IBM Knowledge Center is enabled for accessibility. The accessibility features of IBM Knowledge Center are described at <http://www.ibm.com/support/knowledgecenter/about/releasenotes.html>.

Keyboard navigation

This product uses standard navigation keys.

Interface information

The fully accessible way of using IBM WebSphere MQ is to use the command line interface. For more information about using commands, see [How to use IBM WebSphere MQ control commands and Administration using MQSC commands](#).

For Windows, the accessible way to install IBM WebSphere MQ is by using a non interactive installation. For further information, see [Advanced installation using msixexec](#).

The IBM WebSphere MQ user interfaces do not have content that flashes 2 - 55 times per second.

The IBM WebSphere MQ web user interface does not rely on cascading style sheets to render content properly and to provide a usable experience. However, the product documentation does rely on cascading style sheets. IBM WebSphere MQ provides an equivalent way for low-vision users to use a user's system display settings, including high-contrast mode. You can control font size by using the device or browser settings.

Related accessibility information

In addition to standard IBM help desk and support websites, IBM has established a TTY telephone service for use by deaf or hard of hearing customers to access sales and support services:

TTY service
800-IBM-3383 (800-426-3383)
(within North America)

IBM and accessibility

For more information about the commitment that IBM has to accessibility, see [IBM Accessibility \(www.ibm.com/able\)](#).

Accessibility on Windows

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**.

Index

A

- accessibility 212
 - Explorer object status 212
 - high contrast mode 212
 - screen reader mode 212
- accessibility features for this product 210
- addressing information 49
- administration
 - object name transformation 115
 - queue manager name transformation 115
 - understanding WebSphere MQ file names 115
- advantages of using WebSphere MQ
 - clients 122
- alias queue
 - examples of when to use 98
- alternate channels 47
- application programs
 - receiving messages 38
 - retrieving messages from queues 38
 - sending messages 38
 - time-independent applications 31
- applications on WebSphere MQ MQI
 - clients 122
- attributes
 - queue manager 105
 - queues 102
- authentication information objects
 - description of 106
- authorization service 130
- auto-definition of channels 58

B

- backlevel clients 127
- base queue 98
- benefits of using WebSphere MQ
 - clients 122

C

- channel
 - client-connection 44, 108
 - cluster-receiver 44
 - cluster-sender 44
 - message 107
 - MQI 44, 107
 - overview 107
 - receiver 44
 - requester 44
 - requester-sender 44
 - requester-server 44
 - segregating messages 47
 - sender-receiver 44
 - server-connection 44, 108
 - server-receiver 44
 - sharing 47
 - using alternate channels 47

- channels
 - description of 107
 - stopping 109
- channels, alternate to 47
- client connection channels
 - description of 111
- client to server connection 126
- client-connection channel 44
 - defining 108
- clients
 - compatibility 127
 - version 127
- clients and servers
 - definitions 120
- cluster
 - designing 54
- cluster components 39
- cluster queue 99
 - overview 99
- cluster-receiver 44
- cluster-receiver channel 44
- cluster-sender 44
- cluster-sender channel 44
- clusters
 - cluster transmission queues 103
 - components 39
 - description of 117
 - using 47
- command queues
 - description of 103
- communication
 - connectionless 33
 - time-independent 34
- communication protocol
 - overview 110
- communications
 - overview 110
- components
 - cluster, overview 55
- components of distributed-queuing environment
 - message channel 44
- components, cluster 39
- concepts of intercommunication 39, 49
- connection
 - client to server 126
 - overview 110
- connectionless communication 33
- creating
 - a dynamic (temporary) queue 39
 - a model queue 39
 - a predefined (permanent) queue 39

D

- DCE Generic Security Service (GSS)
 - name service, installable service 130
- dead-letter queue 97
 - overview 47
- dead-letter queues
 - description of 103

- defaults
 - objects 117
- designing clusters 54
- distributed queuing
 - components 44
- dynamic queue
 - overview 100
- dynamic queues
 - description of 39

E

- event queue 97
- event queues
 - description of 104
- event-driven processing 34
- example
 - cluster of queue managers 42
 - message channels
 - cluster-receiver 44
 - cluster-sender 44
 - requester-sender 44
 - requester-server 44
 - sender-receiver 44
 - multi-hopping 47
 - passing through intermediate queue managers 47
 - reply-to-queue alias 53
 - sending messages 41
 - sending messages in both directions 39
 - sharing a transmission queue 47
 - using multiple channels 49
- Explorer object status 212
- extended transactional client
 - defined 124
 - hardware and software requirements 125
 - introduction 124
 - supported platforms 125
 - supported transaction managers 125
- extending queue manager facilities 129

F

- file names 115
- files
 - names 115
 - naming conventions 115
 - understanding names 115

G

- global units of work
 - definition of 128
- glossary 168

H

- hardware and software requirements 125
- high contrast mode 212

I

- initiation queue 97
- initiation queues
 - description of 103
- installable services
 - authorization service 130
 - installable services, list of 130
 - name service 130
 - service component 130
- intercommunication
 - concepts 39, 49

L

- listener objects
 - description of 111
- local queue manager 39
- local queues
 - specific queues used by WebSphere MQ 103
- local unit of work
 - definition of 128

M

- message
 - definition 35
 - descriptor 35
- message channel 107
 - cluster-receiver 44
 - cluster-sender 44
 - receiver 44
 - requester 44
 - requester-sender 44
 - requester-server 44
 - sender 44
 - sender-receiver 44
 - server 44
 - server-receiver 44
- message queuing 31
 - features 33
- message-driven processing 31
- messages
 - application data 38
 - definition of 38
 - message descriptor 38
 - message lengths 38
 - message-driven processing 31
 - queuing 31
 - retrieval algorithms 39
 - retrieving messages from queues 38
 - sending and receiving 38
- model queues
 - creating a model queue 39
- moving service component 40
- MQI (message-queuing interface)
 - definition of 33
 - receiving messages 38
 - sending messages 38

- MQI channel 121
- MQI channels 44, 107
- multi-hopping 47

N

- name resolution 115
 - introduction 50
- name service 130
- name transformations 115
- namelists
 - description of 106
- naming conventions
 - files 115
 - object names 112
 - queue manager name transformation 115
 - rules 113
- NetBIOS 40

O

- OAM (object authority manager)
 - authorization service, installable service 130
 - overview 119
- object
 - creating 116
 - queue 94
 - queue manager 104
 - topic 112
- object name transformation 115
- objects
 - attributes of 117
 - description of 107, 117
 - name transformation 115
 - naming conventions 112
 - rules 113
 - object name transformation 115
 - process definitions 105
 - system default objects 117
 - types of 94

P

- PCF (programmable command format)
 - object attribute names 117
- permanent (predefined) queues 39
- platforms
 - for WebSphere MQ MQI clients 123
 - for WebSphere MQ servers 123
- predefined (permanent) queues 39
- process definitions
 - description of 105

Q

- queue
 - alias 98
 - base 98
 - cluster 99
 - creating 94
 - dead letter 97
 - dead-letter 97
 - event 97

- queue (*continued*)
 - initiation 97
 - introduction to 94
 - name resolution 115
 - remote
 - using 97
 - shared 99
 - system command 97
 - system default 97
 - transmission 96
 - undelivered message 97
- queue manager
 - alias 50
 - attributes 104
 - definition 35
 - number per system 36
 - types 39
 - workload management 105
- queue manager alias
 - introduction 50
- queue managers
 - extending queue manager facilities 129
 - name transformation 115
- queues
 - attributes 102
 - definition of 38
 - dynamic (temporary) queues 39
 - extending queue manager facilities 129
 - model queues 39
 - predefined (permanent) queues 39
 - retrieving messages from 38
 - specific local queues used by WebSphere MQ 103
- queuing
 - definition 31
 - features 33

R

- receiver
 - channel 44
- receiver channel definition
 - overview 41
- recovery 35
- remote queue
 - using 97
- remote queue definition
 - introduction 47
 - what it is 47
- remote queue manager 39
- reply-to queue
 - alias definition 52
 - aliases 50
 - preparing for 53
 - specifying 52
- reply-to queues
 - description of 104
- requester channel 44
- resolution of queue names 115
- resource manager 127
- resources
 - updating under sync point control 127
- retrieval algorithms for messages 39

S

- screen reader mode 212
- secure sockets layer (SSL)
 - overview 119
- security 35
 - channel security 119
 - name service security, overview 119
 - OAM 119
 - object authority manager (OAM) 119
 - SSL 119
- segregating messages 47
- sender channel 44
- sender channel definition
 - overview 41
- server
 - channel 44
 - platform support 123
- server queue manager 121
- server-connection channel 44
 - defining 108
- servers 120
- service component 130
- service objects
 - description of 111
- setting up WebSphere MQ clients 124
- shared queue 99
 - overview 99
- sharing channels 47
- SNA 40
- source queue manager 39
- SPX 40
- store-and-forward 34
- supported transaction managers 125
- sync point 127
 - overview 35
- sync point control 127
- sync point coordinator 127
- system command queue 97
- system default objects 117
- system default queue 97

T

- target queue manager 39
- task list 124
- TCP/IP 40
- temporary (dynamic) queues 39
- time-independent applications 31
- time-independent communication 34
- topic object
 - attributes 112
- transaction 127
- transaction manager
 - definition 127
 - external 127
- transactional support
 - updating under sync point
 - control 127
- transmission header
 - alias
 - definition 51
- transmission queue 96
 - sharing 47
- transmission queues
 - cluster transmission queues 103
 - description of 103

- transport type
 - supported 40
- two phase commit protocol 127

U

- unit of work 127
- UNIX operating system
 - object authority manager (OAM) 119
- user ID
 - maximum length 113

W

- WebSphere MQ commands
 - object attribute names 117
- WebSphere MQ extended transactional client 124
- WebSphere MQ MQI clients and servers 33
- WebSphere MQ object
 - creating 116
 - queue 94
 - queue manager 104
 - topic 112
- WebSphere MQ server
 - platform support 123
- Windows operating system
 - object authority manager (OAM) 119
- Windows systems
 - accessibility 212
- workload management
 - queue manager 105

X

- XA Interface 127

Notices

This information was developed for products and services offered in the U.S.A.

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

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

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

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

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

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

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Software Interoperability Coordinator, Department 49XA
3605 Highway 52 N
Rochester, MN 55901
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

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

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

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

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Programming interface information

Programming interface information, if provided, is intended to help you create application software for use with this program.

This book contains information on intended programming interfaces that allow the customer to write programs to obtain the services of WebSphere MQ.

However, this information may also contain diagnosis, modification, and tuning information. Diagnosis, modification and tuning information is provided to help you debug your application software.

Important: Do not use this diagnosis, modification, and tuning information as a programming interface because it is subject to change.

Trademarks

IBM, the IBM logo, [ibm.com](http://www.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.

