

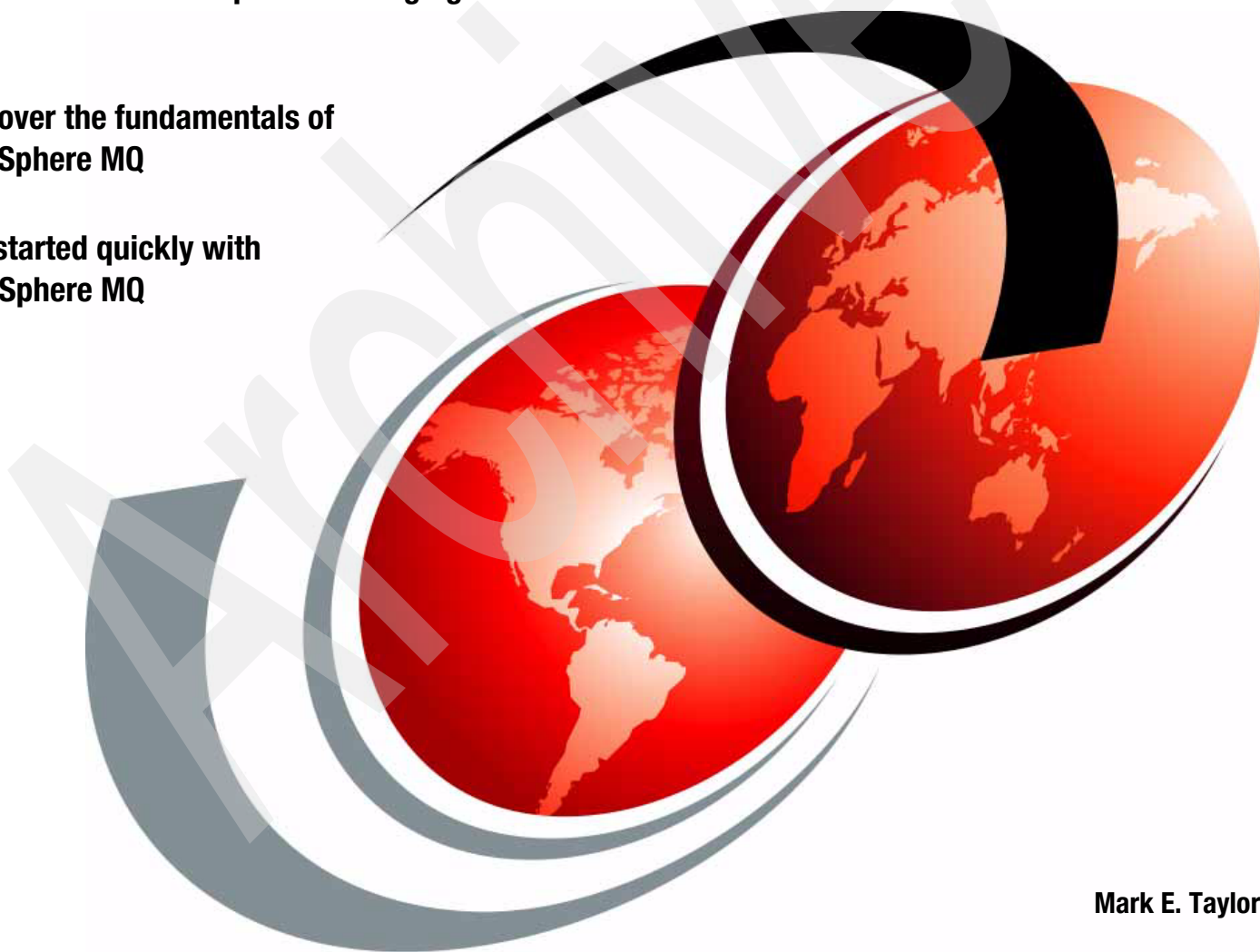
# WebSphere MQ Primer

## An Introduction to Messaging and WebSphere MQ

Learn the basic concepts of messaging

Discover the fundamentals of  
WebSphere MQ

Get started quickly with  
WebSphere MQ



Mark E. Taylor





International Technical Support Organization

**WebSphere MQ Primer: An Introduction to Messaging  
and WebSphere MQ**

December 2012

Archived

**Note:** Before using this information and the product it supports, read the information in “Notices” on page v.

## **Second Edition (December 2012)**

This edition applies to Version 7, Release 1, Modification 0 and Version 7, Release 5, Modification 0 of WebSphere MQ for Multiplatform (product number 5724-H72) and to Version 7, Release 1, Modification 0 of WebSphere MQ for z/OS (product number 5655-R36).

**© Copyright International Business Machines Corporation 2012. All rights reserved.**

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

# Contents

<b>Notices</b> .....	v
Trademarks .....	vi
<b>Preface</b> .....	vii
The author of this paper .....	vii
Now you can become a published author, too! .....	viii
Comments welcome .....	viii
Stay connected to IBM Redbooks .....	viii
<b>Chapter 1. Concepts of messaging</b> .....	1
1.1 The business case for message-oriented middleware .....	2
1.2 Application simplification .....	3
1.3 Example scenarios .....	4
1.3.1 Retail kiosks .....	4
1.3.2 Faster bank payments .....	4
1.3.3 Airport information .....	4
1.4 A messaging-based solution .....	5
1.4.1 Messaging in a service-oriented architecture .....	5
<b>Chapter 2. Introduction to WebSphere MQ</b> .....	7
2.1 Messaging with WebSphere MQ .....	8
2.1.1 A history of WebSphere MQ .....	9
2.2 Core concepts of WebSphere MQ .....	10
2.2.1 Asynchronous messaging .....	11
2.2.2 WebSphere MQ clients .....	12
2.2.3 WebSphere MQ Telemetry clients .....	12
2.2.4 Application programming interfaces (APIs) .....	13
2.2.5 Reliability and integrity .....	14
2.2.6 WebSphere MQ messaging styles .....	15
2.2.7 WebSphere MQ topologies .....	16
2.2.8 Availability .....	20
2.2.9 Security .....	21
2.2.10 Management and monitoring .....	21
2.3 Diverse platforms .....	22
2.4 Relationships with other products .....	23
2.4.1 WebSphere Message Broker .....	23
2.4.2 WebSphere Application Server .....	24
<b>Chapter 3. Getting started with WebSphere MQ</b> .....	25
3.1 Messages .....	26
3.1.1 Message descriptor (MQMD) .....	26
3.1.2 Message properties .....	27
3.2 WebSphere MQ objects .....	28
3.2.1 Queue manager .....	28
3.2.2 Queues .....	28
3.2.3 Topic objects .....	30
3.2.4 Channels .....	31
3.3 Configuring WebSphere MQ .....	32
3.3.1 Creating a queue manager .....	32

3.3.2 Managing WebSphere MQ objects .....	33
3.4 Writing applications .....	34
3.4.1 A code fragment .....	37
3.5 Triggering .....	39
3.6 Configuring a WebSphere MQ client .....	42
3.6.1 How to define a client/server connection .....	43
3.7 Security .....	45
3.8 Configuring communication between queue managers .....	46
3.8.1 How to define a connection between two systems .....	47
3.9 Summary .....	50
<b>Related publications</b> .....	51
IBM Redbooks .....	51
Online resources .....	51
Help from IBM .....	51

# 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.*

**The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:** INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

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

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

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

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.

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.


# Trademarks

IBM, the IBM logo, and [ibm.com](http://www.ibm.com) are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. These and other IBM trademarked terms are marked on their first occurrence in this information with the appropriate symbol (® or ™), indicating US registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at <http://www.ibm.com/legal/copytrade.shtml>

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

AIX®  
CICS®  
DB2®  
IBM®  
IMS™

MQSeries®  
PowerHA®  
RACF®  
Redbooks®  
Redpaper™

Redbooks (logo) ®  
Smarter Planet®  
WebSphere®  
z/OS®

The following terms are trademarks of other companies:

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

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

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

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

Other company, product, or service names may be trademarks or service marks of others.



# Preface

The power of IBM® WebSphere® MQ is its flexibility combined with reliability, scalability, and security. This flexibility provides a large number of design and implementation choices. Making informed decisions from this range of choices can simplify the development of applications and the administration of a WebSphere MQ messaging infrastructure.

Applications that access a WebSphere MQ infrastructure can be developed using a wide range of programming paradigms and languages. These applications can run within a substantial array of software and hardware environments. Customers can use WebSphere MQ to integrate and extend the capabilities of existing and varied infrastructures in the information technology (IT) system of a business.

This IBM Redpaper™ publication provides an introduction to message-oriented middleware to anyone who wants to understand messaging and WebSphere MQ. It covers the concepts of messaging and how WebSphere MQ implements those concepts. It helps you understand the business value of WebSphere MQ. It provides introductory information to help you get started with WebSphere MQ. No previous knowledge of the product and messaging technologies is assumed.

## The author of this paper

This paper was produced by a specialist working at the International Technical Support Organization, Raleigh Center.



**Mark E. Taylor** is a member of the WebSphere MQ Technical Strategy organization at the IBM Hursley laboratory in England. Mark is responsible for defining the functions included in new releases of WebSphere MQ. During his 25 year career with IBM, Mark worked in a variety of development and services roles. He wrote code for the early versions of IBM MQSeries®, porting it to numerous UNIX operating systems.

Thanks to the following people for their contributions to this project:

Marcela Adan  
IBM Redbooks® Project Leader, International Technical Support Organization, Raleigh Center

Deana Coble  
Technical Writer, International Technical Support Organization, Raleigh Center

Thanks to the author of the previous edition of this paper:

- ▶ Dieter Wackerow, author of the first edition, *MQSeries Primer*, published in October 1999

## Now you can become a published author, too!

Here's an opportunity to spotlight your skills, grow your career, and become a published author—all at the same time! Join an ITSO residency project and help write a book in your area of expertise, while honing your experience using leading-edge technologies. Your efforts will help to increase product acceptance and customer satisfaction, as you expand your network of technical contacts and relationships. Residencies run from two to six weeks in length, and you can participate either in person or as a remote resident working from your home base.

Find out more about the residency program, browse the residency index, and apply online at: [ibm.com/redbooks/residencies.html](http://ibm.com/redbooks/residencies.html)

## Comments welcome

Your comments are important to us!

We want our papers to be as helpful as possible. Send us your comments about this paper or other IBM Redbooks publications in one of the following ways:

- ▶ Use the online **Contact us** review Redbooks form found at:  
[ibm.com/redbooks](http://ibm.com/redbooks)
- ▶ Send your comments in an email to:  
[redbooks@us.ibm.com](mailto:redbooks@us.ibm.com)
- ▶ Mail your comments to:  
IBM Corporation, International Technical Support Organization  
Dept. HYTD Mail Station P099  
2455 South Road  
Poughkeepsie, NY 12601-5400

## Stay connected to IBM Redbooks

- ▶ Find us on Facebook:  
<http://www.facebook.com/IBMRedbooks>
- ▶ Follow us on Twitter:  
<http://twitter.com/ibmredbooks>
- ▶ Look for us on LinkedIn:  
<http://www.linkedin.com/groups?home=&gid=2130806>
- ▶ Explore new Redbooks publications, residencies, and workshops with the IBM Redbooks weekly newsletter:  
<https://www.redbooks.ibm.com/Redbooks.nsf/subscribe?OpenForm>
- ▶ Stay current on recent Redbooks publications with RSS Feeds:  
<http://www.redbooks.ibm.com/rss.html>



# Concepts of messaging

This chapter discusses the reasons why a business might require a messaging solution and different requirements for messaging. This chapter provides some examples of where messaging is used. This chapter contains the following sections:

- ▶ The business case for message-oriented middleware
- ▶ Application simplification
- ▶ Example scenarios
- ▶ A messaging-based solution

## 1.1 The business case for message-oriented middleware

Business environments are constantly changing. Applications that were written 20 years ago need to exchange data with applications written last week. Examples of this changing environment can be one company that merges with another company, a new partner to communicate with, an application that is used internally in a company is now exposed to customers, or different departments within a company need to share programs.

The growth of Internet banking required services, such as managing payments or querying account information, to be made available through a range of channels. The core data can be held in a database on a mainframe, but a user of a browser requires a front-end web application server to interact with that database. As new delivery channels are created, such as smartphone applications, easy ways for that new mechanism to interact are needed. The smartphone application must communicate with the same applications and database without changing the database.

The evolutionary speed of IT systems makes essential the ability to integrate across many environments with multiple applications reliably and quickly.

Messaging is an effective way to connect these systems. It hides many of the details of communication from the application developer and gives a simple interface. Simplifying allows the developer to concentrate on the business problem instead of worrying about matters such as recovery, reliability, and operating system differences.

A further feature of messaging solutions is the decoupling of one application from another. Many mechanisms for communicating between applications require that both applications are available at the same time. Messaging uses an *asynchronous* model, which means that an application that generates messages does not have to execute at the same time as an application that consumes those messages. Reducing the requirements for simultaneous availability reduces complexity and can improve overall availability. Messages can be sent to specific applications or distributed to many different applications at the same time.

Figure 1-1 shows basic patterns for connectivity.

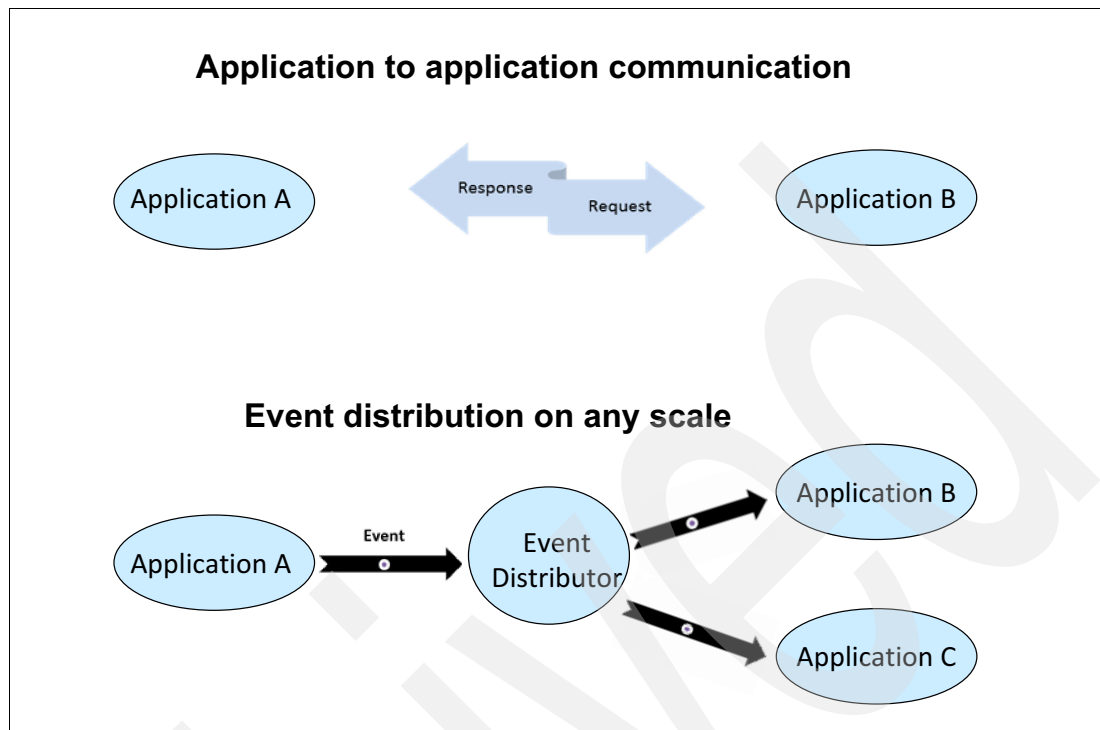


Figure 1-1 Using messaging to connect applications and distribute data

Asynchrony is used in various ways. It can queue large volumes of work, and that work is then submitted once a day by a batch process. Or, work can be submitted immediately and then processed as soon as the receiving application finishes dealing with a previous request.

Asynchronous processing does not imply long response times. Many applications were built and successfully execute by using messaging for real-time interactive operations.

## 1.2 Application simplification

One of the key features of messaging is to remove complexity from application code. Worrying about reliability or recovery is the job of the *middleware* product or component, such as the messaging provider. Developers can simply focus on writing the business logic.

The following list covers some of the factors that need to be dealt with by middleware, and hidden from application developers, regardless of the business application:

**Once-only processing** Business transactions normally happen exactly once. Middleware needs to ensure all the systems that are involved in that transaction do their job, exactly once. There must be no loss or duplication. Applications must not write complex code for recovery or to reverse partially completed operations (compensation) code if individual systems fail.

**Ubiquity** Run your applications on the platform where it makes the most sense. Be able to integrate applications seamlessly on a wide range of operating systems and hardware environments.

<b>Easy to change</b>	Using industry standards can help you become more responsive as skills can be maintained and reused across projects. New applications can be rapidly written and deployed.
<b>Easy to extend</b>	Scale your applications rapidly to any volume without downtime. Add more processing without application change. Roll out new applications and features quickly and safely without downtime.
<b>Compliance</b>	Meet enterprise, industry, and regulatory requirements easily, especially as those requirements evolve. A middleware implementation needs to assist with auditing and other compliance objectives, giving a consistent interface for all applications.
<b>Performance and availability</b>	Make the best use of the systems. Provide services for high and continuous availability.
<b>Security</b>	Provide a secure environment. Data is protected from loss, modification, and reading. Losing sensitive data, or failing to comply with regulations, costs time, money, and reputation.

## 1.3 Example scenarios

This section gives examples of where messaging solves a real-world problem.

### 1.3.1 Retail kiosks

A retailer wants to respond rapidly to online customer needs and have a consistent view of data that is shown at 25,000+ self-service kiosks. Data is changing rapidly, and it needs to be distributed rapidly to these kiosks.

The kiosks in the network are connected by a messaging service to the central databases and changes are pushed out rapidly. More than 14,000 transactions are processed each second.

### 1.3.2 Faster bank payments

Government regulation that limits payment clearing times requires faster payment processes through near-instant money transfers between banks.

A high-performance payment solution is developed that uses a messaging solution to route message traffic between systems. Many existing components of the older payment system were able to be adapted to work with the new messaging layer, preserving investment in these programs. The government requirement was to be able to clear payments in under two hours. However, this solution is technically capable of completing its processing in seconds. It handles millions of transactions every day.

### 1.3.3 Airport information

At an airport, rapid changes in government regulations are coupled with increasing passenger volumes. This airport needs a flexible IT infrastructure that is populated with seamlessly integrated systems. The airport currently supports daily business operations with an inflexible heterogeneous infrastructure.

## 1.4 A messaging-based solution

A messaging-based solution establishes a shared integration layer, enabling the seamless flow of multiple types of data between the customer's heterogeneous systems. For example, airport staff can now easily access important information, such as flight schedule changes from multiple systems. A messaging-based solution can also flexibly integrate new systems to accommodate evolving government regulations.

### 1.4.1 Messaging in a service-oriented architecture

Service-oriented architecture (SOA) is a business-centric IT architectural approach. This approach supports business integration as linked, repeatable business tasks or services. SOA helps users build composite applications. Composite applications draw upon functionality from multiple sources within and beyond the enterprise to support horizontal business processes. SOA is an architectural style that makes this possible. For more information about SOA, see the following link:

<http://www.ibm.com/soa>

The most important characteristic of SOA is the flexibility to treat elements of business processes and the underlying information technology infrastructure as secure, standardized components (services). These components can be reused and combined to address changing business priorities. SOA requires that applications can interact with each other.

This book is not intended to discuss SOA in detail, but a critical aspect of implementing the architecture is to have connectivity. The seven main attributes of the connectivity infrastructure are:

- ▶ Reliable
- ▶ Secure
- ▶ Time flexible and resilient
- ▶ Transactional
- ▶ Incremental
- ▶ Ubiquitous
- ▶ Basis for enterprise service bus (ESB)

Any messaging component fitting in the connectivity infrastructure must satisfy and support all these capabilities. A flexible and robust messaging backbone is a key component of SOA and provides the transport foundation for an ESB.

Archived





# Introduction to WebSphere MQ

This chapter introduces IBM WebSphere MQ and how it implements messaging. Core concepts are described, and major features are discussed.

This chapter contains the following sections:

- ▶ Messaging with WebSphere MQ
- ▶ Core concepts of WebSphere MQ
- ▶ Diverse platforms
- ▶ Relationships with other products

## 2.1 Messaging with WebSphere MQ

WebSphere MQ is the market-leading messaging integration middleware product. Originally introduced in 1993 (under the MQSeries name), it always focused on providing an available, reliable, scalable, secure, and high-performance transport mechanism to address the requirements that we discussed in the previous chapter.

WebSphere MQ always connected systems and applications, regardless of the platform or environment. It is essential to be able to communicate between a GUI desktop application that is running on Microsoft Windows and an IBM CICS® transaction that is running on IBM z/OS®. That value of universality is core to the product, and this value has not changed in all the time that it has been available. What *has* changed is the range of environments in which WebSphere MQ can or must live.

There are newer platforms, environments, requirements for qualities of service (QoS), and newer messaging patterns. Security has become more important as systems are made accessible to more users across an enterprise and beyond it. Performance and scalability requirements have increased. Regulators and auditors have imposed more controls on what can or must be done. Systems, which need access to enterprise data, have become both more powerful (faster, more processors, and so on) and much less powerful (sensors, tablets, and mobile phones). Therefore, WebSphere MQ has evolved.

There are now more ways for applications to reach a WebSphere MQ queue manager and to access any existing applications that were already WebSphere MQ enabled. New applications can be written that use alternative interfaces and still maximize the reliability and performance of WebSphere MQ. Those new applications do not need a complete replacement of existing infrastructure. The applications work with what you already have and what you know how to manage.

At the same time as adding new interfaces, protocols, and environments, a large amount of the WebSphere MQ workload continues to be executed in mainframe-based data centers. The efficient use of, and integration with, the capabilities of the IBM z hardware and operating systems is critical. As this book shows, WebSphere MQ V7.1 for z/OS significantly improves the performance, capacity, and availability that can be achieved when running in a sysplex.

## 2.1.1 A history of WebSphere MQ

Figure 2-1 shows a time line of WebSphere MQ versions and some highlights of the features and products that have been made available.

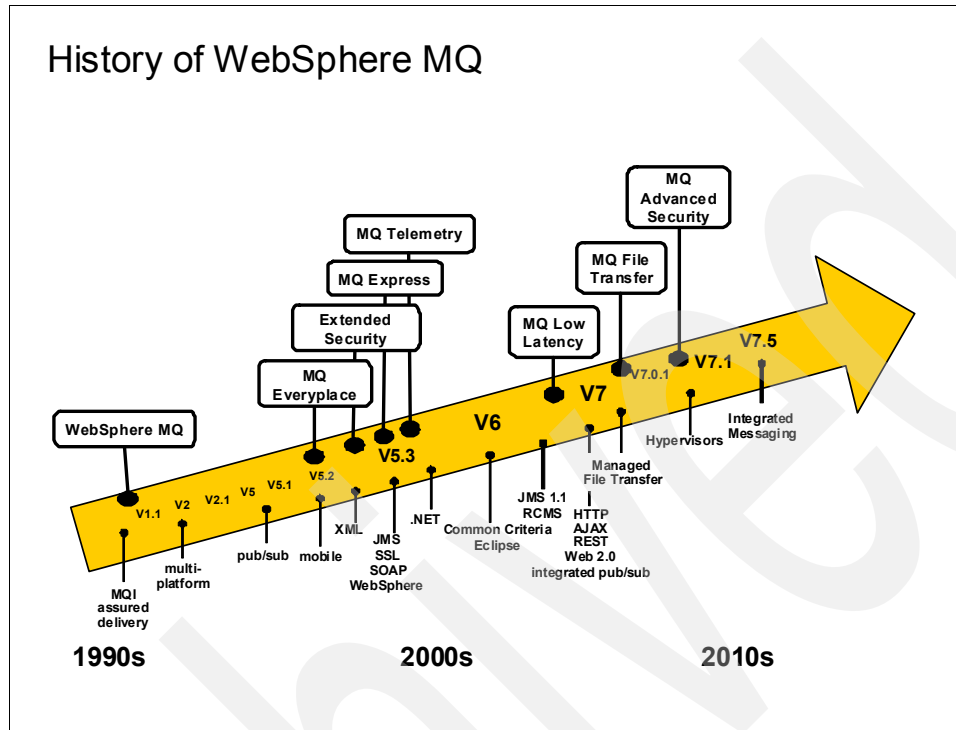


Figure 2-1 A history of WebSphere MQ

WebSphere MQ V7.0, in 2008, was a major release with significantly enhanced application programming interfaces (APIs) and a fully integrated publish/subscribe component. These enhancements made that messaging pattern a first-class way of working. WebSphere MQ V7.0.1, in 2009, was primarily an update to improve availability and performance.

Even though there was nearly a 2 ½ year gap before V7.1 in 2011, there were still incremental improvements. The improvements were to both the core WebSphere MQ product and new products.

WebSphere MQ File Transfer Edition is a solution for the integration of applications that are file-based, making it possible to reliably transfer files between systems. WebSphere MQ File Transfer Edition provides management, monitoring, and security of those transfers. WebSphere MQ File Transfer Edition uses existing skills and infrastructure as it uses WebSphere MQ as its backbone transfer medium.

Advanced Message Security (AMS) is a solution to the problem of securing message contents, even when those messages are *at rest* in queues or logs. Several regulatory frameworks require data to be protected from viewing, changing, or being stored on disks in the clear even during processing. One example of these regulations is the Payment Card Industry (PCI) for dealing with credit card data. There are other regulatory bodies for other types of data, for example, healthcare, that have similar requirements.

Both WebSphere MQ File Transfer Edition and WebSphere MQ Advanced Message Security have been integrated into the WebSphere MQ V7.5 package. WebSphere MQ File Transfer Edition has been renamed to Managed File Transfer. There are two variations of V7.5 that are

functionally identical but that have different licensing terms. The WebSphere MQ Advanced offering can be ordered under a single part number and price and it includes licenses to use the Managed File Transfer and AMS features. The original V7.5 offering has these features as separately orderable and priced components.

## 2.2 Core concepts of WebSphere MQ

Data is transferred between applications in messages. A *message* is a container that consists of three parts:

- ▶ **WebSphere MQ Message Descriptor (MQMD):** Identifies the message and contains additional control information. Examples of additional information are the type of message and the priority that is assigned to the message by the sending application.
- ▶ **Message properties:** An optional set of user-definable elements that describes the message without being part of the payload. Applications that receive messages can choose whether to inspect these properties.
- ▶ **Message data:** Contains the application data. The structure of the data is defined by the application programs that use it, and WebSphere MQ is largely unconcerned with its format or content.

The nodes, within a WebSphere MQ message queuing infrastructure, are called *queue managers*. The queue manager is responsible for accepting and delivering messages. Multiple queue managers can run on a single physical server or on a wide network of servers across a large variety of different hardware and operating system platforms.

Each queue manager provides facilities for reliable messaging.

The queue manager maintains *queues* of all messages that are waiting to be processed or routed. Queue managers are tolerant of failures and maintain the integrity of business-critical data that flows through the message queuing infrastructure. In the event of failures, such as a failure of the queue manager, machine failures, or a power loss, messages can be recovered. Recovery occurs when the queue manager is restarted and any transactions are brought to a consistent state.

The queue managers, within the infrastructure, are connected by logical *channels* over a communications network. Messages automatically flow across these channels from the initial producer of a message to the eventual consumer of that message based on the configuration of the queue managers in the infrastructure. Changes can be made to the configuration of queues and channels transparently for the applications. For example, a receiving application can be moved to a new machine, and a route to that machine can be defined without needing any changes to sending applications.

Figure 2-2 shows these essential elements.

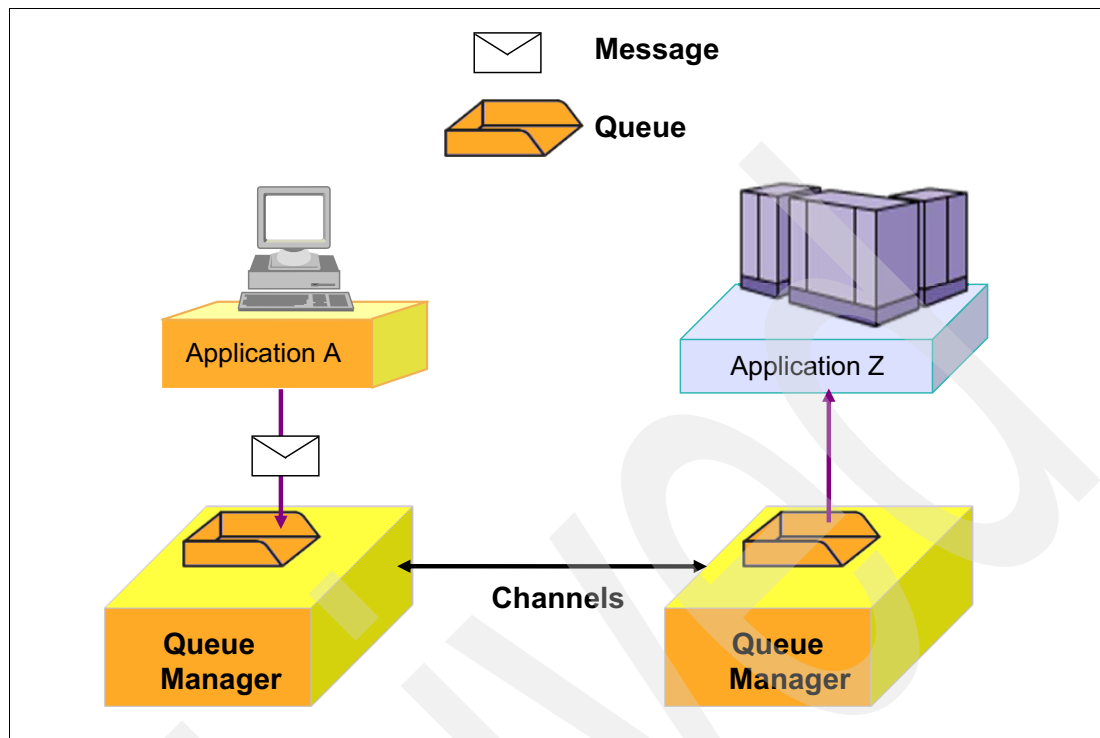


Figure 2-2 Basic WebSphere MQ components to connect applications

## 2.2.1 Asynchronous messaging

Two applications that must communicate, whether hosted on the same machine or separate machines, might have originally been designed to do so directly and synchronously. This was a common messaging technique that was used before the introduction of WebSphere MQ and is still used often. For example, using HTTP as a communications protocol is a synchronous operation.

In the synchronous model, the two applications exchange information by waiting for the partner application to become available and then sending the information. If the partner application is unavailable for any reason, including if it is busy communicating with other applications, the information cannot be sent.

All intercommunication failures that can occur between the two applications must be considered individually by the applications. This consideration happens whether the applications are on the same machine or on different machines that are connected by a network. The process requires a protocol for sending the information, confirming receipt of the information, and sending any subsequent reply.

Placing a WebSphere MQ infrastructure between the two applications allows this communication to become asynchronous. One application places information for the partner in a message on a queue, and the partner application processes this information when it is available to do so. If required, it can then send a reply message back to the originator. The applications do not need to be concerned with communication failures or recovery.

## 2.2.2 WebSphere MQ clients

A WebSphere MQ client is a lightweight component of WebSphere MQ that does not require the queue manager runtime code to reside on the client system. It enables an application, running on a machine where only the client runtime code is installed, to connect to a queue manager that is running on another machine and perform messaging operations with that queue manager. Such an application is called a *client* and the queue manager is referred to as a *server*.

Using a WebSphere MQ client is an effective way of implementing WebSphere MQ messaging and queuing. The following benefits are available by using a WebSphere MQ client:

- ▶ There is no need for a licensed WebSphere MQ server installation on the client machine.
- ▶ Hardware requirements on the client system are reduced.
- ▶ System administration requirements on the client system are reduced.
- ▶ An application that uses a WebSphere MQ client can connect to multiple queue managers on different machines.

There are several implementations of the WebSphere MQ client that allow work in different language environments, including Java and .Net classes. There is no client for z/OS. On the z/OS platform, applications can only connect to queue managers within the same logical partition (LPAR).

**Important:** Because there must be a synchronous communication protocol between the client and the queue manager, WebSphere MQ clients require a reliable and stable network.

## 2.2.3 WebSphere MQ Telemetry clients

WebSphere MQ Telemetry (or the MQTT protocol) was designed for small footprint, low processing, low bandwidth, unreliable networked systems where even a WebSphere MQ client was too heavyweight. It has an even simpler API to minimize the processing that is needed. The MQTT protocol is public and a number of implementations, which are not from IBM, exist. For more information, see the following website:

<http://www.mqtt.org>

The first implementations of MQTT were in places, such as sensors on pipelines, but it is also key to integration with newer environments, such as tablets and mobile phones. In the IBM Smarter Planet® vision of interconnected, integrated systems, much of that integration is expected to be with MQTT. The MQTT integration can be either as a raw API and protocol or working with solutions, such as IBM Worklight, that simplify how you build, deploy, and manage applications in the mobile world.

The WebSphere MQ Telemetry feature brings together several pieces of existing technology and simplifies the experience for a seamless integration with WebSphere MQ applications.

It is critical that the queue managers, to which MQTT devices connect, are able to handle the levels of scale that those solutions demand. Many thousands of devices can be generating data and require connectivity. Benchmarks have shown that a single queue manager can handle 100,000 MQTT connections. Messages from these devices are then passed to applications that are enabled with WebSphere MQ for further processing. Solutions, such as

smart meters for power utilities, health monitors, and traffic management systems in cities, have all been implemented by using MQTT services.

## 2.2.4 Application programming interfaces (APIs)

Applications can use WebSphere MQ with several programming interfaces.

In many cases, applications are written (or rewritten) to use these interfaces. However, there are circumstances, particularly on z/OS, in which existing applications can be unchanged as WebSphere MQ provides bridges to and from their environments. In particular, there are bridges for CICS and IBM IMS™ transactions that convert WebSphere MQ messages. The messages are converted into input for these transactions and the reverse is done for returned responses.

### Message Queue Interface

The native interface is the Message Queue Interface (MQI). The MQI consists of the following elements:

- ▶ Calls or verbs through which programs can access the queue manager and its facilities
- ▶ Structures that programs use to pass data to, and get data from, the queue manager
- ▶ Elementary data types for passing data to, and getting data from, the queue manager
- ▶ Classes in object-oriented languages for accessing data, the queue manager, and its facilities

Many programming languages and styles are supported, depending on the software and hardware platform, such as C, COBOL, Java, and C#.

### Standardized APIs

Using a standardized API can add additional flexibility when accessing services through a message queuing infrastructure. This book uses the term *standardized API* to represent APIs that are not proprietary to an individual product, such as WebSphere MQ.

The following list notes examples of standardized APIs that can be used to access services that are provided through a WebSphere MQ infrastructure:

- ▶ Java Message Service (JMS)
- ▶ IBM Message Service Client (XMS)

Wide adoption of these APIs can occur across multiple products. For example, the JMS API is an industry standardized API for messaging within the Java Enterprise Edition (Java EE) specification.

These standardized APIs generally have less function than the MQI when they use a less capable messaging service than WebSphere MQ. The standards were written based on a common subset of functions available at the time on all relevant messaging providers.

### HTTP

There is a defined mapping between HTTP operations (**GET**, **POST**, and **DELETE**) and a subset of MQI functions. A queue manager can be configured to have an HTTP server that is associated with it and then process HTTP requests directly. This makes it simple for applications to interact with WebSphere MQ systems without needing any WebSphere MQ code installed at all. For example, a smartphone has an HTTP interface that user-written applications can use, but there might not be a WebSphere MQ client available for that machine.

## 2.2.5 Reliability and integrity

WebSphere MQ provides a number of features to maintain the reliable delivery of messages.

### Persistent and non-persistent messages

Messages that contain critical business data, such as the receipt of payment for an order, must be reliably maintained and must not be lost in a failure.

Some messages might contain query data only, where the loss of the data is not crucial because the query can be repeated. In this case, performance is considered more important than data integrity.

To work with these opposing requirements, WebSphere MQ uses two types of messages. The following list shows the message types (persistent and non-persistent) with details:

- ▶ **Persistent messages:** WebSphere MQ does not lose a persistent message through network failures, delivery failures, or the restart of the queue manager.  
Each queue manager keeps a failure-tolerant recovery log of all actions that are performed upon persistent messages. This log is sometimes referred to as a *journal*.
- ▶ **Non-persistent messages:** WebSphere MQ optimizes the actions that are performed upon non-persistent messages for performance.  
Non-persistent message storage is based in system memory, so it is possible that the non-persistent messages can be lost in situations, such as network errors, operating system errors, hardware failure, queue manager restart, and internal software failure.

WebSphere MQ assures once-only delivery of persistent messages and it assures at-most-once delivery of non-persistent messages. Both persistence options allow developers to write applications with the knowledge that there are no duplicated messages.

### Units of work

Many operations that are performed by an application cannot be considered in isolation. An application might need to send and receive multiple messages as part of one overall action. Only if all of these messages are successfully sent or received are any messages sent or received. These actions are considered to be a unit of work (UOW). All WebSphere MQ implementations support transactional operations, with sets of messages being committed or backed out as a single action.

An application that processes messages might also need to perform coordinated work against other resources and the WebSphere MQ infrastructure. For example, it might perform updates to information in a database based upon the contents of each message. The actions of retrieving the message, sending any subsequent reply, and updating the information in the database must only complete if all actions are successful.

Units of work that are performed by applications that access a WebSphere MQ infrastructure can include sending and receiving messages and updates to databases. On Distributed platforms, WebSphere MQ can coordinate all resources to ensure that a unit of work is only completed if all actions within that unit of work complete successfully. On all platforms, WebSphere MQ can also participate in global units of work that are coordinated by other products. For example, actions against a WebSphere MQ infrastructure can be included in global units of work that are coordinated by WebSphere Application Server and IBM DB2®. On z/OS, global units of work are often coordinated by CICS, IMS, or Resource Recovery Services (RRS).



## Interconnected queue managers

The communication that is performed across channels between queue managers is tolerant of network failures. The communication protocol that is used between queue managers maintains the once-only or at-most-once delivery that is indicated by the message's persistence level.

### 2.2.6 WebSphere MQ messaging styles

There are two basic styles of messaging, which are known as point-to-point and publish/subscribe.

#### Point-to-point

The *point-to-point* style of messaging is built around the concept of message queues. Messages are stored on a queue by a source application, and a destination application retrieves the messages. This provides the capability of storing and forwarding messages to the destination application in an asynchronous or decoupled manner. Synchronous Request/Reply designs can also be implemented by using point-to-point asynchronous messaging.

The source application needs to know the destination of the message. The queue name is usually enough when alias queues, remote queues, or clustered queue objects are used, but some designs might require the source application to know the name of the remote queue manager.

Point-to-point messaging is normally used when there is known to be a single producer and a single consumer of the messages.

For example, an application running on Linux might inquire about a piece of information in a database that is only directly readable by using a CICS transaction. The application sends a message with the inquiry details, a CICS transaction reads that message, then reads the database, and sends the response back in another message. In this example, a single CICS transaction can be processing requests from many instances of the Linux application and sending unique responses to each of those instances.

#### Publish/subscribe

WebSphere MQ *publish/subscribe* allows the provider of information to be further decoupled from consumers of that information.

In point-to-point messaging, connected applications need to know the name of the queues through which they interact. The queue names might not actually be the same in the sender and receiver because the WebSphere MQ administrator can define aliases and other mechanisms to route to other systems.

In publish/subscribe, the applications agree on the name of a *topic*. The producer of a message is known as a publisher, and the message is labeled with a topic. The consumers of messages (subscribers) tell WebSphere MQ that they are interested in a topic, and each is sent a copy of the relevant messages.

A *topic* is simply a text string, normally with a hierarchical structure, such as */Price/Fruit/Apples*. A subscriber might use this topic exactly or subscribe to topics with a wild card so that there is no need to know all the topics that might be used by the publisher.

There can be a single subscriber, many, or none to any particular topic. The publisher does not need to know how many subscribers exist.

The classic example of publish/subscribe is to distribute stock prices. Many people are interested in the price of a particular company's stock, and there is a single provider publishing that price. Traders can join and leave the feed at any time, and the provider does not need to know that. Another example might be for a retailer to distribute catalog changes from a central site to all branches in a region.

## 2.2.7 WebSphere MQ topologies

A topology that consists of a single queue manager running on the same machine as its applications has limitations of scale and flexibility. WebSphere MQ clients can allow applications to run on remote machines. The connections between the applications and the queue manager require a reliable and stable network.

The limitations of this topology can be eliminated without alteration to the applications by using distributed messaging. Applications that access a service can use a queue manager that is hosted on the same machine as the application, providing a fast connection to the infrastructure. WebSphere MQ channels extend the connectivity by providing transparent asynchronous messaging with applications hosted by other queue managers on remote machines using a communications network.

There are two common types of distributed messaging:

- ▶ Hub and spoke
- ▶ WebSphere MQ clustering

### Hub and spoke

Applications that are accessing a service connect to their local queue manager. Usually, this occurs as a direct connection to a queue manager that runs on the same machine. A client connection can also be used to a queue manager on the same machine or to a queue manager on a different machine over a fast, reliable network. For instance, an application in a branch office needs to access a service at headquarters. Asynchronous communication occurs through a local queue manager that acts as a spoke to the service application that is hosted on a hub queue manager.

The machine that hosts a hub queue manager can host all the applications that provide the central services, for instance, headquarters applications and databases.

This type of architecture is developed by manually defining the routes from the spoke queue managers to the hub queue manager or to many hub queue managers. Multiple services that are provided by the infrastructure can be hosted on different hub queue managers or through multiple queues on the same hub queue manager.

Figure 2-3 shows a simple hub and spoke design. Messages can be sent from Branch A to Branch D, but only by passing through the Corporate Office queue manager.

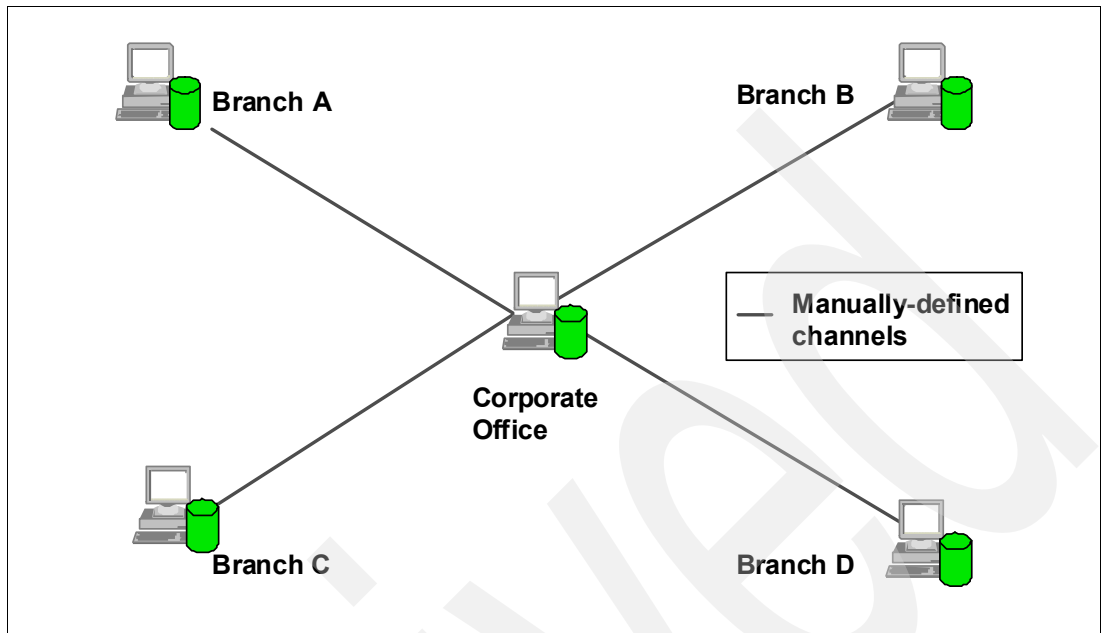


Figure 2-3 A simple hub and spoke configuration

Manually defined routes do not have to follow a hub and spoke design. It is possible to have all the queue managers in the enterprise directly connected one to another. But when the number of queue managers grows, maintaining these routes can be a time-consuming activity.

Figure 2-4 shows what happens when direct connections are created between all queue managers, and the number of queue managers grows.

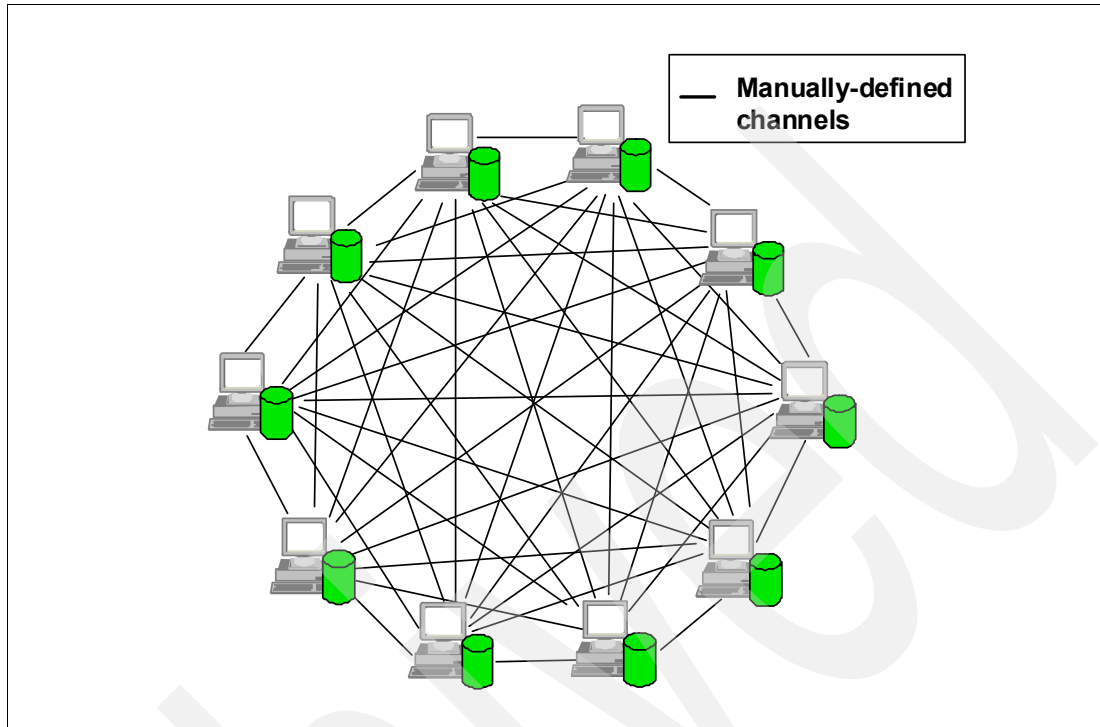


Figure 2-4 Any-to-any direct paths without clustering

### WebSphere MQ cluster

A more flexible approach is to join many queue managers together in a dynamic logical network that is called a *queue manager cluster*. A cluster allows multiple instances of the same service to be hosted through multiple queue managers.

Applications, requesting a particular service, can connect to any queue manager within the queue manager cluster. When applications make requests for the service, the queue manager to which they are connected uses a workload balancing algorithm to spread these requests across all available queue managers that host an instance of that service.

Figure 2-5 shows an example of workload balancing. The message that is sent from Application A can be processed by any instance of Application Z. The cluster chooses one of the routes based on various criteria, including knowledge of which route it previously chose and which routes are still available. By default, each instance of Application Z must be sent the same number of messages from Application A.

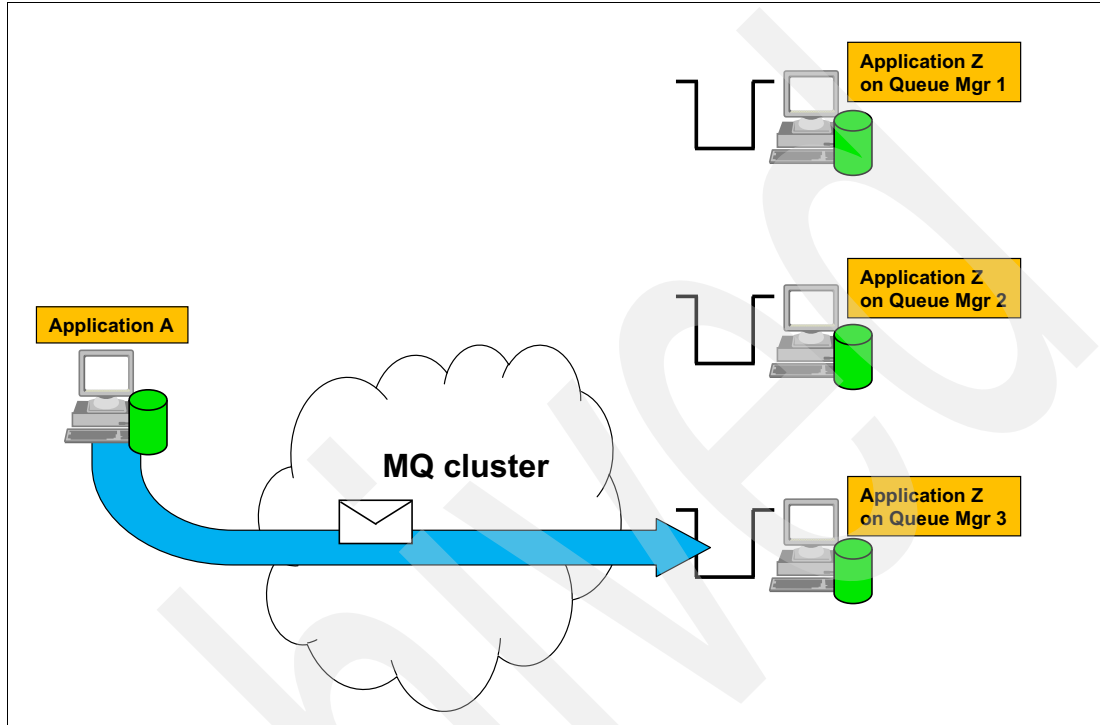


Figure 2-5 Workload balancing in a WebSphere MQ cluster

This configuration allows a pool of machines to exist within the WebSphere MQ cluster, each of which hosts a queue manager and the applications that are required to provide the service. This topology is especially useful in a distributed environment, where capacity is scaled to accommodate the current load through multiple servers rather than one high-capacity server. A server can fail or be shut down for maintenance and service is not lost.

**Cluster:** The word *cluster* can have many meanings. In particular, it is often used to refer to high availability (HA) solutions that are based on failover technology. WebSphere MQ clusters can be used effectively with HA clusters, but they are providing different services.

Using WebSphere MQ cluster technology can also simplify administration tasks because most of the message channels for application data transport are maintained by the cluster and do not have to be explicitly created.

Figure 2-6 shows the simplified administration that is possible with a WebSphere MQ cluster. Messages flow directly between any two queue managers, without needing to pass through a hub. Only a single initial definition is required to a designated queue manager that is known as a *full repository*. From that point, relationships are automatically discovered when required and the associated channels are defined without intervention.

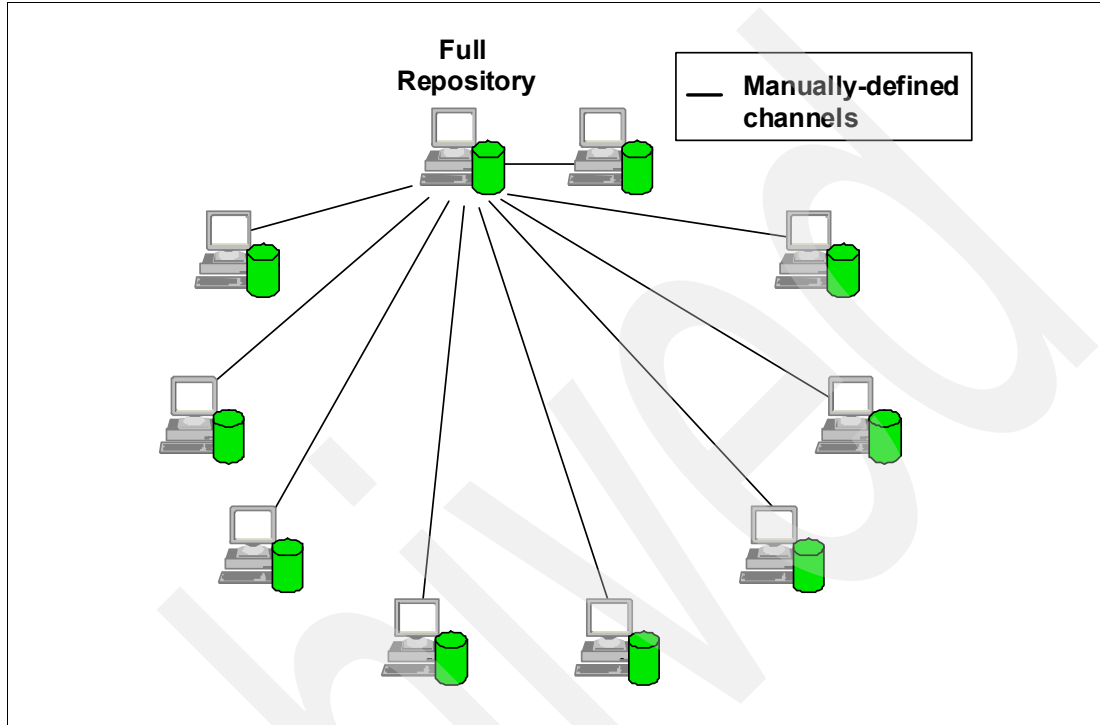


Figure 2-6 Simplified administration with a WebSphere MQ cluster and automatic definition

## 2.2.8 Availability

WebSphere MQ provides and integrates with a range of technologies for availability. The following list shows some of those technologies and the availability benefits:

- ▶ Platform-specific products, such as IBM PowerHA® (for IBM AIX®), Microsoft Cluster Services (for Windows), and automatic restart manager (ARM) for z/OS, can be used to control the restart and recovery of queue managers in failures.
- ▶ On distributed platforms, WebSphere MQ has a feature that is known as *multi-instance support*, which allows the queue manager to restart itself automatically on a backup machine.
- ▶ On z/OS, WebSphere MQ integrates with operating system and hardware-provided services for a sysplex. Shared queues can provide continuous availability, with messages being processed even when an entire LPAR fails and takes out a queue manager.
- ▶ WebSphere MQ clusters are not a complete answer to availability, but they can often form part of a fuller HA solution.

For a more detailed description of WebSphere MQ HA capabilities, see *High Availability in WebSphere Messaging Solutions*, SG24-7839.

## 2.2.9 Security

WebSphere MQ has a range of security features to protect against unauthorized access to resources and data. The following list is of the most commonly used capabilities:

- Access Control** WebSphere MQ can be configured to permit only certain users or groups to connect or to use resources, such as queues or topics.
- The permissions can be granular, for example, a user is permitted to **put** messages to a particular queue, but not to **get** messages from that same queue. New **CHLAUTH** records are used for controlling access through channels.
- SSL/TLS** The Secure Sockets Layer (SSL) and Transport Layer Security (TLS) are industry standardized technologies that provide assurance of identity and data privacy. SSL and TLS combine with WebSphere MQ client applications that connect to queue managers by using a communications network infrastructure. These security layers also work with queue manager to queue manager distributed queuing and clustering using a network.
- SSL and TLS provide similar capabilities and build upon similar principles for establishing identity. TLS is often considered the successor of SSL because it provides some enhanced security features. SSL or TLS can be used for all communication that is performed over a network within a WebSphere MQ infrastructure.
- AMS** The Advanced Message Security feature or component supplies an additional layer of security on individual messages. It ensures that messages cannot be read or changed by unauthorized users, even when the message is not being transmitted on a network but is stored on a queue or recorded in a recovery log file.

## 2.2.10 Management and monitoring

WebSphere MQ provides many interfaces and services for configuring, managing, and monitoring its operation.

For example, you can define and alter queue attributes, be sent alerts when a queue gets full or a channel stops unexpectedly, and obtain statistics about how much traffic is passing through a queue manager or individual applications.

There are two primary ways in which this management is done:

1. You can enter text commands, which are known as WebSphere MQ script commands (MQSC), into a console or scripting environment. For example, this command creates a queue with specified attributes:  

```
DEFINE QLOCAL(MY.QUEUE) DESCR('Test Queue') DEFPSIST(YES)
```
2. Programmatic control is done with WebSphere MQ messages that are sent to, or read from, well-known queues. The messages have documented formats, which are called the programmable command format (PCF), and many tools and products exist to create and use them.

Shipped as part of the WebSphere MQ product is an Eclipse-based tool that is called the WebSphere MQ Explorer. This graphical interface makes it easy to start using WebSphere MQ.

Figure 2-7 shows the WebSphere MQ Explorer interface.

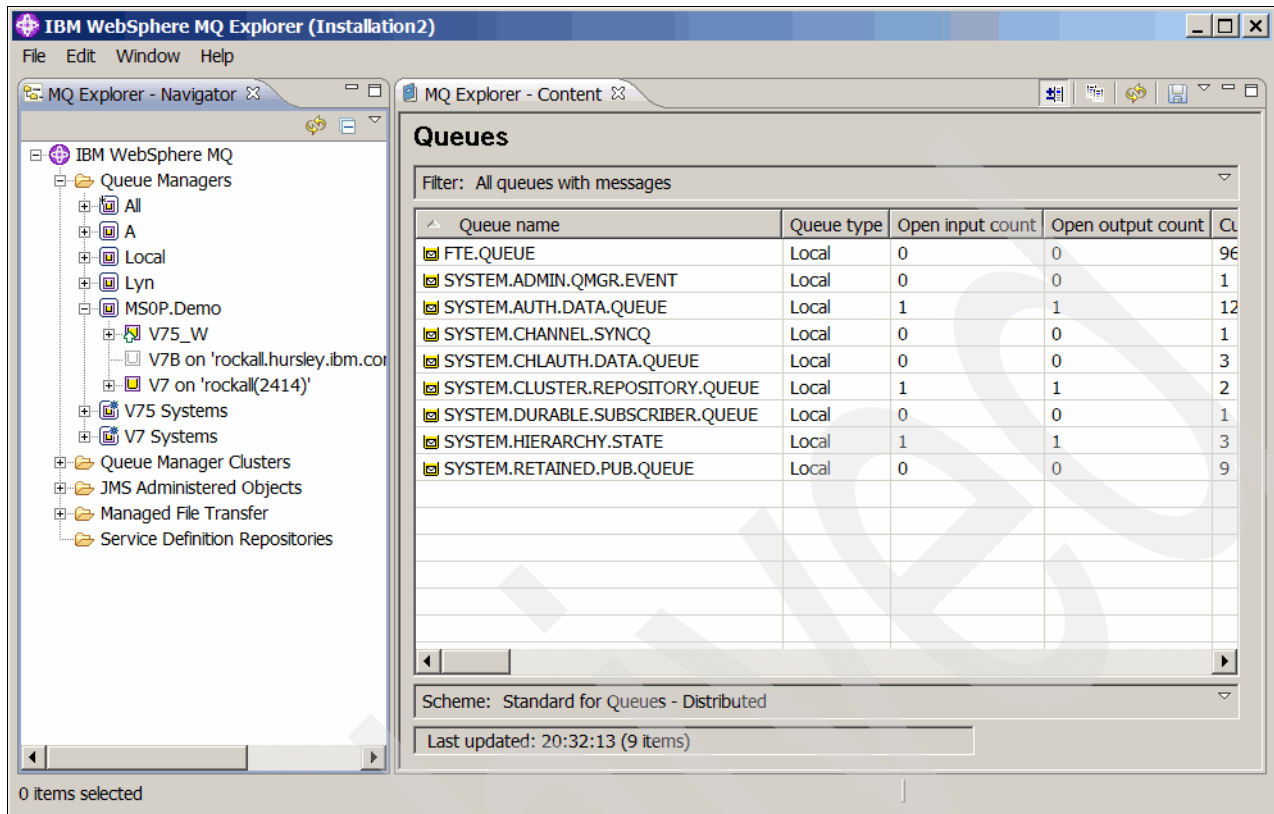


Figure 2-7 WebSphere MQ Explorer

## 2.3 Diverse platforms

WebSphere MQ provides simplified communication between applications running on many different hardware platforms and operating systems, which are implemented using different programming languages. This flexibility enables a business to choose the most appropriate infrastructure components for implementing or accessing services within their system. The messaging infrastructure understands differences between the underlying hardware and software on which individual nodes are running.

Some conversion of character data might be required for the data to be readable across different hardware and software platforms. For example, WebSphere MQ can convert between ASCII and EBCDIC code pages. It can also convert to and from Unicode characters. The messaging infrastructure can be configured to perform this conversion transparently so that each message is valid when it is retrieved at the destination.

To learn about supported platforms for WebSphere MQ, see the following IBM web pages:

- ▶ *WebSphere MQ Product Information*, available at:  
<http://www.ibm.com/support/docview.wss?uid=swg27007431>
- ▶ *WebSphere MQ System Requirements*, available at:  
<http://www.ibm.com/software/integration/wmq/requirements>



**Distributed Platforms:** When it is important to point out platform differences, the term *Distributed Platforms* is often used to cover all of the WebSphere MQ implementations, except for z/OS. This term refers to WebSphere MQ on Windows, UNIX, Linux, and i. Do not confuse Distributed Platforms with references to Distributed Queueing.

## 2.4 Relationships with other products

One of the strengths of WebSphere MQ is that it does not require the use of any other specific products. The WebSphere word at the front of the product name merely indicates a branding within IBM, as one of many products that share the WebSphere brand. It does not force the use of any other WebSphere brand products. For example, if you want to use another vendor's Java Platform, Enterprise Edition (JEE) application server, WebSphere MQ implements standard interfaces to support such a product.

However, there are two products that are frequently used alongside WebSphere MQ:

- ▶ WebSphere Message Broker
- ▶ WebSphere Application Server

### 2.4.1 WebSphere Message Broker

Enterprise systems consist of many logical endpoints, such as off-the-shelf applications, services, packaged applications, web applications, devices, appliances, custom built software, and many more. These endpoints expose a set of inputs and outputs, including:

- ▶ Connection protocols, such as WebSphere MQ, TCP/IP, database, HTTP, files, FTP, SMTP, and POP3
- ▶ Data formats, such as C or COBOL structures, XML, industry-specific (SWIFT, EDI, and HL7), and user-defined

An enterprise service bus (ESB) runs at the center of this wide range of connections and applications and protocols. It performs tasks, such as routing and data transformation, so that any endpoint can communicate with any other endpoint. Traffic passes through an ESB without requiring a complex and unmaintainable mesh of individual point-to-point connections.

WebSphere Message Broker is an ESB product that connects these endpoints in meaningful ways to simplify application and device integration. It requires that WebSphere MQ is also installed on systems where it runs. This is partly because WebSphere MQ enabled applications are an important class of connectivity that it must support, but also because it uses queue manager-provided services, such as transaction coordination, publish/subscribe, and the reliable storage of messages. One of the WebSphere Message Broker administration tools (the WebSphere Message Broker Explorer) runs inside WebSphere MQ Explorer to provide a single point of control for both products.

One example of how WebSphere Message Broker simplifies integration is to consider a typical retail enterprise. Many different endpoints exist both inside and outside the organization, with data in multiple formats and protocols, such as TLOG, files, and JSON/HTTP. As new capabilities need to blend in (such as support for mobile apps and analytics), the flexibility of WebSphere Message Broker makes it simple to add these functions without disrupting existing services.

For more information about WebSphere Message Broker, see the following resources:

- ▶ *Using WebSphere Message Broker V8 in Mid-Market Environments*, SG24-8020:  
<http://www.redbooks.ibm.com/abstracts/sg248020.html>
- ▶ WebSphere Message Broker Library:  
<http://www-01.ibm.com/software/integration/wbimessagebroker/library/>

## 2.4.2 WebSphere Application Server

WebSphere Application Server is a product that hosts and runs applications. At its core, it is a JEE environment, implementing and supporting a range of open standards. One such standard that JEE applications often use is the Java Message Service (JMS).

JEE application servers make it possible to use any vendor's JMS implementation, but there is no requirement about how simple it is to use a non-native JMS provider. WebSphere Application Server includes both the runtime Java client code and the administration panels that make it easy to connect to a WebSphere MQ queue manager. Therefore, web applications can reliably send messages to, and get responses from, any other WebSphere MQ enabled application that might be running in the enterprise.

For more information about WebSphere Application Server, see the following resources:

- ▶ *WebSphere Application Server V8.5: Technical Overview Guide*, REDP-4855:  
<http://www.redbooks.ibm.com/abstracts/redp4855.html>
- ▶ WebSphere Application Server Library:  
<http://www-01.ibm.com/software/webservers/appserv/was/library/>



# Getting started with WebSphere MQ

This chapter goes into a deeper level of detail than the previous chapter. It shows how to get started with configuring a WebSphere MQ environment. Adapted and brought up-to-date from the *MQSeries Primer*, REDP-0021, published in 1999, it shows how the elements described in Chapter 2, “Introduction to WebSphere MQ” on page 7 are implemented and used.

This chapter contains the following topics:

- ▶ Messages
- ▶ WebSphere MQ objects
- ▶ Configuring WebSphere MQ
- ▶ Writing applications
- ▶ Triggering
- ▶ Configuring a WebSphere MQ client
- ▶ Security
- ▶ Configuring communication between queue managers
- ▶ Summary

## 3.1 Messages

A *message* is a container that consists of three parts:

- ▶ WebSphere MQ Message Descriptor (MQMD): Identifies the message and contains additional control information, such as the type of message and the priority that is assigned to the message by the sending application.
- ▶ Message properties: An optional set of user-definable elements that describes the message without being part of the payload. Applications that receive messages can choose whether to inspect these properties.
- ▶ Message data: Contains the application data. The structure of the data is defined by the application programs that use it; WebSphere MQ is largely unconcerned with its format or content.

An individual message can contain up to 100 MB of data. There are no constraints on the format of this data. It can be readable text, binary, XML, or any combination.

### 3.1.1 Message descriptor (MQMD)

The MQMD is a fixed structure that describes the message. The fields within the structure have various functions:

- ▶ Describe the message format
- ▶ Routing
- ▶ Describe the sender of the message
- ▶ Instruct the queue manager how to process the message

This list explains the important fields in the MQMD:

- ▶ The *MsgType* describes the high-level purpose of the message. The most frequently used values in this field are datagram, request, reply, and report. A *datagram* is set by the sending application when it is not expecting any response from the receiving application.
- ▶ *MsgID* (message ID) and *CorrelID* (correlation ID) are used to identify a specific request or reply message. The programmer can set a value in one or both fields or have WebSphere MQ automatically create a unique ID. These fields are 24 bytes long and are not treated as characters that can be converted between code pages, such as ASCII to EBCDIC.

A normal pattern for application programs is that the message ID that is generated by WebSphere MQ when a message is put to a queue is used to identify responses sent back to the designated reply queue. The program that receives the request message copies the incoming message ID into the correlation ID of the reply message. This allows the originating program (the one that gets the reply) to instruct the queue manager to return a specific message in the reply queue instead of just getting the first message from the queue. It is particularly important when multiple applications are reading messages from the same queue.

- ▶ The *Persistence* flag selects whether the message must be treated as persistent, and therefore recoverable in a system failure, or whether the message is non-persistent and can be discarded when an error occurs. A programmer can explicitly set persistence to be on or off, or allow the value to be controlled by a system administrator.

Queues can hold a combination of persistent and non-persistent messages.

- ▶ Messages have a *Priority* that is set as a value between 1 and 10. An administrator can define that messages are retrieved in priority order, rather than true first in first out (FIFO) sequencing.

- ▶ The queue manager records the *PutTime* and *PutDate* when the message was initially put to a queue. These values are in Coordinated Universal Time (UTC), or Greenwich Mean Time (GMT), not the local time, to allow a consistent view of time when messages pass between systems in different timezones.
- ▶ Messages might have *Expiry* time. After this time has passed, the message cannot be returned to any application. This is used for situations where a message is not going to be useful after a while. For example, after sending a request message, an application often waits a finite period (for example, 10 seconds) for a reply to be returned and shows an error if nothing is received in that time. Knowing that, both the request and response messages can have a 10-second expiry time set. So, in the event of a system failure, the messages are removed from the queues when they are no longer useful and do not waste disk or memory space.
- ▶ When sending a request message, the responding application must be told where to send the reply. This is usually done by providing the *ReplyToQ* and *ReplyToQMgr* names in the initial request.
- ▶ The *Format* field describes the message data. It is often set to MQSTR (to denote text data) or NONE (to denote binary data), but there are other possible values. In some cases, the body of the message is itself made up of additional WebSphere MQ-defined structures and the format field tells WebSphere MQ what to do with that data. User-defined formats can also be used. The format setting causes data conversion to take place when it is needed. For example, it causes WebSphere MQ to convert text data between ASCII and EBCDIC code pages (the *CodedCharSetId* field) when the sending and receiving applications are on different systems using those character sets.
- ▶ Information about the sending application (program name and path) and the platform on which it is running are set in the *PutApplType* and *PutApplName* fields.
- ▶ The *Report* field is used to request information about the message as it is processed. For example, the queue manager can send a report message to the sending application when it puts the message in the target queue or when the receiving application gets it off the queue.
- ▶ Each time a message is backed out, the *BackoutCount* is increased. An application can check this counter and act on it, for example, send the message to a different queue where the reason for the backout is analyzed by an administrator.
- ▶ Messages can be sent as part of a related group. The receiving application can then wait until all messages from the group have arrived before starting its processing. Message grouping also guarantees that the order in which the messages are sent is preserved. The *GroupId* and *MsgSeqNumber* fields in the MQMD define this processing.

### 3.1.2 Message properties

An application can set arbitrary properties of a message that are not part of the fixed MQMD structure. For example, a message can have a *color* property assigned, and then be described as red or green. Properties can be strings, numbers, or boolean values.

These properties are not considered part of the message data, but can optionally be viewed and modified by applications. Properties can also be used to select the messages that are returned to an application. For example, an application can request to be sent all of the green messages but none of the red messages.

Message properties are most commonly used in publish/subscribe applications to provide an additional layer of selection beyond just the topic. It is similar to an SQL query using the WHERE clause to return specific rows from a database table.

## 3.2 WebSphere MQ objects

There are currently 12 types of *objects* that can be configured in WebSphere MQ, as well as various other resources (for example security access controls, or subscriptions) that are managed by WebSphere MQ. The most important objects from this set are queues, channels, topics, and the queue manager itself.

Objects have names that are up to 48 characters long (20 characters for channels) and are case-sensitive. A queue that is called `queue.abc` is different from a queue called `QUEUE.ABC`. Both of these queues can be defined on a queue manager, although it is confusing to do so. Objects have attributes or properties that define the behavior when they are used.

Some objects have subtypes, which in turn have different sets of properties. For example, the `CONNAME` is an attribute of `SENDER` channels but not of `RECEIVER` channels.

### 3.2.1 Queue manager

The *queue manager* is the fundamental resource in WebSphere MQ. Almost all other resources are owned by a queue manager. The only resource that is an exception is the queue-sharing group, which consists of a set of queue managers on z/OS.

It is the queue manager that manages the storage of data and the recovery after a failure. The queue manager coordinates all the applications updating messages on queues, and deals with the isolation and locking that are required to maintain consistency. The queue manager maintains statistics and status information for reporting, as needed.

A queue manager has many attributes to control its processing. For example, various types of events can be enabled or disabled to report an important activity in the system. Resource constraints, such as the maximum number of queues that an application can have open simultaneously, are defined here. The publish/subscribe capabilities are enabled at the queue manager level.

The first activity for a WebSphere MQ administrator is to create a queue manager. After a queue manager is created and started, the other objects can be defined on that queue manager by using WebSphere MQ management commands and application programming interfaces (APIs).

### 3.2.2 Queues

*Queues* are used either to store messages that are sent by programs or are pointers to other queues or topics. There are a number of different queue types and some special uses of queues.

#### Queue types

The following queue types can be defined within a queue manager:

- |                    |   |
|--------------------|---|
| <b>Local queue</b> | A local queue is the only place where messages are physically stored. All other queue types must be defined to point at a local queue.  |
| <b>Alias queue</b> | An alias queue is usually a pointer to a local queue. Using an alias queue definition is a convenient way to allow different applications to use different names for the same real queue. An alias queue can also be used as a pointer to a topic for publish/subscribe applications. |

**Remote queue** A remote queue is another type of pointer. A remote queue definition describes a queue on another queue manager. When an application opens a remote queue, the queue manager to which the application is connected ensures that messages are tagged and stored locally in preparation for sending to another queue manager. Applications cannot read messages from remote queues.

**Model queue** A model queue is a template that can be used by an application to dynamically create a real queue. These templates are often used to create a unique queue for reply messages and then the queue is automatically deleted when the application ends.

**Dynamic queue** These queues are created by the process of opening a model queue. They are real local queues, but they cannot be explicitly created through administration interfaces.

There are two subtypes of dynamic queue: A *temporary dynamic queue* (TDQ) is automatically deleted when the application that caused it to be created ends. A *permanent dynamic queue* (PDQ) survives the creating application and can be reused by another application.

One common misconception is to talk about *persistent queues* or *non-persistent queues*. With one exception, there are no such things. Queues can hold both persistent and non-persistent messages, although they have an attribute to define the default persistence value for messages put to them if the application does not explicitly set the value. The exception to this rule is a TDQ that can only be used for non-persistent messages.

## Special queues

Some queues are defined as having particular purposes. This list describes the main uses:

**Transmission queue** A transmission queue is a local queue with the `USAGE(XMITQ)` attribute configured. It is a staging point for messages that are destined for a remote queue manager. Typically, there is one transmission queue for each remote queue manager to which the local queue manager might connect directly. If the destination is unavailable, messages build up on the transmission queue until the connection can be successfully completed. Transmission queues are transparent to the application. When an application opens a remote queue, the queue manager internally creates a reference to the relevant transmission queue and messages are put there.

**Initiation queue** An initiation queue is a local queue to which the queue manager writes a *trigger message* when certain conditions are met on another local queue. For example, this is done when a message is put into an empty message queue or in a transmission queue. Such a trigger message is transparent to the programmer, but it is a powerful mechanism to cause applications or channels to start executing only when there is work to be processed. For more information about how this process works, see 3.5, “Triggering” on page 39.

**Dead-letter queue** A dead-letter queue (DLQ) is usually defined when a queue manager is created. It is used for situations when a queue manager cannot deliver a message across a channel. Here are some examples:

The destination queue is full.

The destination queue does not exist.

Message puts have been inhibited on the destination queue.

The sender is not authorized to use the destination queue.

The message is too large.

When these conditions are met, the messages are written to the DLQ. A queue manager has at most one DLQ.

Messages are only sent to the DLQ after they have successfully been put to a local transmission queue first.

If an application putting a message receives an error that indicates the output queue is full, the message is not put to a DLQ. The message is never accepted by the queue manager for processing.

#### **Event queue**

The queue manager generates event messages when certain things happen. For example, events can occur when a queue is nearly full or when an application is not authorized to open a queue. These event messages are written to one of the predefined event queues and can then be processed by management tools. All event queues have similar names that indicate the type of event that is held there. `SYSTEM.ADMIN.QMGR.EVENT` or `SYSTEM.ADMIN.CHANNEL.EVENT` are examples of queue manager event queues.

#### **Cluster queue**

A cluster queue is a local queue that is configured to advertise its existence within a WebSphere MQ cluster. Applications can refer to this queue without needing any additional definitions, and all the queue managers in the cluster know how to route messages to it. Usually, there are multiple cluster queues of the same name within the cluster. The sending queue manager selects which cluster queue to use based on a workload balancing algorithm.

### **3.2.3 Topic objects**

WebSphere MQ V7.0 introduced a fully integrated publish/subscribe capability, which includes the ability to define *topic* objects.

A topic is a character string that describes the nature of the data that is published in a publish/subscribe system. Instead of including a specific destination address in each message, a publisher assigns a topic to the message. The queue manager matches the topic with a list of subscribers who have subscribed to that topic, and delivers a copy of the published message to each of those subscribers.

Although any character string can be used as the topic, it is usual to choose topics that fit into a hierarchical tree structure. That structure helps with establishing security policies and makes it easy for applications to subscribe to multiple topics in a single step.

Topic objects represent a subset of the topics that are used by applications.

There can be many thousands of topics that are used, but no administrator wants to create definitions of all of them. The topic objects are used as control points in the tree of in-use topics where different configurations are needed.

For example, subbranches of the topic tree have different access control requirements. Topic objects are used to assign authorizations to any topic that is lower in the tree. Whenever a decision on access control is required, the queue manager finds the topic object that is the closest parent in the tree to the application-provided topic string and uses that object's definition.



## 3.2.4 Channels

A *channel* is a logical communication link. In WebSphere MQ, there are two kinds of channels, with various subtypes:

**Message channels** A message channel connects two queue managers via message channel agents (MCAs). These channels are unidirectional, in that messages are only sent in one direction along the channel.

An MCA is a program that transfers messages from a transmission queue to a communication link, and from a communication link into the target queue.

There are subtypes of message channel, with the different types distinguished by whether they are used for receiving or sending messages, whether they initiate the communication to the partner queue manager, and whether they are used within a cluster or not.

A channel definition on one queue manager has to be matched with a channel definition of the same name with an appropriate subtype on the partner queue manager.

The different subtypes of message channels are known as SENDER, RECEIVER, SERVER, and REQUESTER with CLUSTER-SENDER and CLUSTER-RECEIVER used for clustered configurations. While other combinations are possible, the most common configuration is to have a SENDER channel defined on one queue manager with an identically named RECEIVER channel on a second queue manager.

When the first versions were released, there was no single dominant network protocol. WebSphere MQ was designed to support several network protocols for the MCAs, and these other protocols can still be used, but it is now rare to find anything other than TCP/IP.

**MQI channels** A Message Queue Interface (MQI) channel connects a WebSphere MQ client to a queue manager across a network. An MQI channel is bidirectional in that one channel can be used for a client to both send and receive messages. The two subtypes of MQI channel are SVRCONN (a definition used by the queue manager) and CLNTCONN (a definition used by the client program).

For two-way messaging between queue managers, there are usually two symmetric pairs of message channels defined. One pair is for messages flowing in one direction; the other pair handles the reverse traffic.

Channels are usually configured to start automatically when messages are available on the associated transmission queue. This feature uses *trigger* attributes that are set on the transmission queue.

When a WebSphere MQ cluster is in use, it is not necessary to manually define the channels. The cluster has enough information to create channels dynamically when needed.

Figure 3-1 shows a simple configuration that connects two queue managers and their relationship to some of the object definitions that are described in this section.

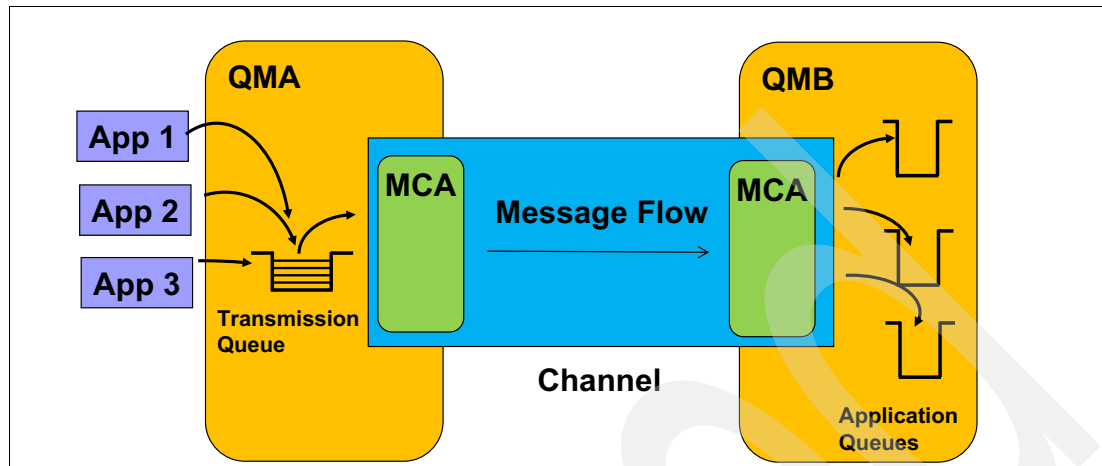


Figure 3-1 Using a channel to send messages from QMA to QMB

### Listeners

One further part of the inter-communications configuration is the concept of a *listener*. Listeners are programs that wait for inbound network connection requests, and then start the appropriate channel to handle the communication.

All queue managers need at least one listener running if they are going to receive messages or client connections. A standard configuration is to have a TCP/IP listener waiting for connections on port 1414. If multiple queue managers are running on the same system, they require individual listeners to be configured on different ports.

## 3.3 Configuring WebSphere MQ

This section shows some basic steps on getting started with WebSphere MQ. It uses Microsoft Windows as the operating system example. Other distributed platforms are similar.

On z/OS, the initial steps are different and might involve several different administrators. These steps are fully described in the WebSphere MQ Information Center:

<http://publib.boulder.ibm.com/infocenter/wmqv7/v7r1/index.jsp>

After the queue manager is running, managing the WebSphere MQ object definitions is the same regardless of platform. The graphical WebSphere MQ Explorer is an easy way to create and modify definitions but the examples that are shown here use the command-line mechanisms.

### 3.3.1 Creating a queue manager

First, a WebSphere MQ administrator creates a queue manager.

The `crtmqm` command creates a queue manager. Figure 3-2 shows an example of creating a queue manager called QMA.

```
C> crtmqm QMA
WebSphere MQ queue manager created.
Directory 'C:\mqm\qmgrs\QMA' created.
The queue manager is associated with installation 'Installation2'.
Creating or replacing default objects for queue manager 'QMA'.
Default objects statistics : 77 created. 0 replaced. 0 failed.
Completing setup.
Setup completed.
```

Figure 3-2 Creating a queue manager

**Important:** Queue manager names are case-sensitive.

There are no limits to the number of queue managers that can be created and run on a server. It is possible to designate one queue manager as the default for the server, and any application or administration task uses that default if no queue manager name is used. However, it is clearer to have no default queue manager and always explicitly use the name.

There are default definitions for the objects that every queue manager needs, such as model queues. These objects are created automatically.

To start this queue manager and make it available for applications, use the `strmqm` command as shown in Figure 3-3.

```
C> strmqm QMA
WebSphere MQ queue manager 'QMA' starting.
The queue manager is associated with installation 'Installation2'.
5 log records accessed on queue manager 'QMA' during the log replay phase.
Log replay for queue manager 'QMA' complete.
Transaction manager state recovered for queue manager 'QMA'.
WebSphere MQ queue manager 'QMA' started using V7.5.0.0.
```

Figure 3-3 Starting the queue manager

It is now possible for applications to start putting and getting messages; the administrator can perform additional configuration by using the WebSphere MQ administration tools.

### 3.3.2 Managing WebSphere MQ objects

On distributed platforms, the `runmqsc` program provides a console into which management commands can be typed. On z/OS, the same commands can be entered through panels, System Display and Search Facility (SDSF), or submitted as jobs.

The queue manager must be running before you use the `runmqsc` command. It works in two ways:

- ▶ Type the commands directly.
- ▶ Use input redirection to work with a text file that contains a list of commands. For example, entering `runmqsc QMA < commands.mqsc` runs the commands in the file.

The commands in Figure 3-4 are examples that are used to create the local queue QUEUE1. They then create a channel to send messages to another queue manager, and they create the transmission queue that is associated with the channel. To end the session, type `end`.

```
C> runmqsc QMA
5724-H72 (C) Copyright IBM Corp. 1994, 2011. ALL RIGHTS RESERVED.
Starting MQSC for queue manager QMA.

define qlocal('QUEUE1')
  1 : define qlocal('QUEUE1')
AMQ8006: WebSphere MQ queue created.
define channel('TO.QMB') chltype(sdr) xmitq('QMB') conname(hostB)
  2 : define channel('TO.QMB') chltype(sdr) xmitq('QMB') conname(hostB)
AMQ8014: WebSphere MQ channel created.
define qlocal('QMB') usage(xmitq)
  3 : define qlocal('QMB') usage(xmitq)
AMQ8006: WebSphere MQ queue created.
end
  4 : end
3 MQSC commands read.
No commands have a syntax error.
All valid MQSC commands were processed.

C>
```

Figure 3-4 Defining objects using `runmqsc`

There are commands to define, alter, and delete all the properties of all WebSphere MQ objects, including queues and channels, and the queue manager. The `runmqsc` command can also be used to modify queue managers on remote machines, but that is beyond the scope of this introduction.

**Case-sensitive:** Attributes in the `runmqsc` commands are automatically converted to uppercase unless they are enclosed in single quotation marks. WebSphere MQ is case-sensitive for object names, and forgetting this is a common mistake. It typically results in an `UNKNOWN_OBJECT_NAME` error when trying to open a queue.

## 3.4 Writing applications

This section is an outline of the WebSphere MQ programming interface, the MQI. It includes a brief program that is written in C to give you an understanding of how WebSphere MQ programs look. There are many sample programs that are included with WebSphere MQ to demonstrate more of the features. These samples are written in a range of languages, including C, COBOL, C#, and Java. The Java examples show the use of both the full-function MQI (in its object-oriented form) and Java Message Service (JMS).

## The MQI

The MQI currently has 26 verbs. There were about 13 until WebSphere MQ V7.0 was released, and then the number doubled to support publish/subscribe and message property operations. The most commonly used verbs are shown in Table 3-1. Each verb has only a few parameters.

Many of the parameters to these verbs are structures that contain additional fields that are used by the verb. New releases of WebSphere MQ typically introduce function that is controlled by additional fields in these structures. In order to maintain compatibility with existing applications, these structures contain a version number that is incremented whenever the structure is extended. Applications compiled against the older definitions have the original structure versions, and the queue manager does not reference the newer fields. The number of parameters for each verb does not change with new releases, which is done to maintain compatibility.

Success or failure of each call is indicated by two parameters that are included in every verb, the Reason and CompCode values.

Table 3-1 The main MQI verbs

MQI verb	Description
MQCONN/MQCONNX	Connect to a queue manager
MQDISC	Disconnect from a queue manager
MQOPEN	Open a specific queue or topic
MQCLOSE	Close a queue or topic
MQPUT	Put a message on a queue, or publish a message
MQGET	Get a message from a queue, or receive a publication
MQPUT1	MQOPEN + MQPUT + MQCLOSE
MQSUB	Create a subscription
MQINQ	Inquire about the properties of an object
MQSET	Set properties of an object
MQBEGIN	Begin a globally coordinated (XA) unit of work
MQCMIT	Commit a unit of work
MQBACK	Back out

The following list provides more information about these verbs:

### **MQCONN and MQCONNX**

These verbs create a connection with a queue manager. MQCONN was the original verb, but when additional parameters were required, a new verb (MQCONNX) was introduced to avoid requiring changes to old applications. MQCONNX gives much more control over the connection, including the ability to specify everything needed to create a client connection to a queue manager on a different machine.

An application creates the connection when it starts to run, and it does not disconnect until the application ends. Creating the connection is the most expensive (time-consuming) verb in the MQI, especially if a secure socket needs to be created to a remote queue manager. That means that it needs to be used infrequently.

**MQDISC** Disconnects the application from the queue manager. If there is an uncommitted transaction, calling MQDISC commits that transaction. If an application ends, or abends, without calling MQDISC, the transaction is rolled back.

**MQOPEN** Makes an object, such as a queue or topic, available to the application. A queue needs to be opened before an application can put or get messages. Parameters to this verb include the object name and an indication of how the application intends to use it. For example, flags indicate whether the application puts or gets messages (or both). This information is used to perform security checks. Applications that are not permitted to access the object in the requested way are rejected.

It is possible to open a queue for exclusive use, or to share it with other applications. Sharing a queue might improve performance because more work can be processed in parallel.

**MQCLOSE** It is the reverse of MQOPEN. It indicates that the application is no longer interested in using this object. Many people believe that MQCLOSE commits any transactional work that uses the queue, which is not true.

**MQPUT** Puts a message to a queue, or publishes a message on a topic.

**MQGET** Retrieves a message from a queue. This verb can return immediately, or wait a specified time until a message satisfying certain criteria arrives on the queues.

There are also options to retrieve a message non-destructively (*browse*), including the capability of browsing all messages in sequence on a queue.

**MQPUT1** This verb is a shorthand combination of MQOPEN, MQPUT, and MQCLOSE. It exists because a typical application pattern is to read incoming messages, perform some work, and then send a message back to a unique reply queue that is owned by another application. If the reply queue is only used one time, this verb is more efficient than the three verbs that it replaces.

There is *no* corresponding MQGET1 verb because the pattern of only getting a single message from a particular queue is not common. Even applications, which use a unique, dynamically created, reply queue, must have opened the model queue before trying to retrieve from the queue, so the shorthand does not work.

**MQSUB** Creates a subscription to a topic. The subscription can include wildcards so that multiple topics are referenced by the same subscription. Published messages that match the topic are sent to a queue that is ready for MQGET to process them. The subscriber's queue can either be named explicitly as a parameter to MQSUB, or automatically created.

**MQINQ** Requests information about the queue manager or one of its objects. Not all attributes of queues or queue managers can be inquired about by using this verb, but some important ones can be.

<b>MQSET</b>	Changes attributes of an object. It has a similar subset of capabilities to MQINQ.
<b>MQBEGIN</b>	<p>Begins a unit of work that is coordinated by the queue manager and that can involve external XA-compliant resource managers. It is used to coordinate transactions that use both queues (MQPUT and MQGET under syncpoint) and database updates (SQL commands). This verb is not required if only WebSphere MQ resources are updated by the application.</p> <p>This verb is only available on locally connected applications on distributed platforms. It is not available on z/OS, and on WebSphere MQ clients. On WebSphere MQ clients, an external coordinator, such as an application server, is required. On z/OS, applications use Resource Recovery Services (RRS) as the coordinator.</p>
<b>MQCMIT</b>	<p>Specifies that a syncpoint is reached. Messages put as part of a unit of work are made available to other applications. Messages retrieved as part of a unit of work are permanently deleted from the queue. If the queue manager is acting as a global transaction coordinator, the XA commit process in other resource managers is invoked.</p> <p>If the application uses an external transaction coordinator, such as WebSphere Application Server or CICS, it must not call MQCMIT. Transactional control is performed by using the coordinator's APIs.</p>
<b>MQBACK</b>	<p>Tells the queue manager to back out all message puts and gets that have occurred since the last syncpoint. Messages put as part of a unit of work are deleted. Messages retrieved as part of a unit of work are reinstated on the queue. If the queue manager is acting as a global transaction coordinator, the XA backout process in other resource managers is invoked.</p> <p>If the application uses an external transaction coordinator, such as WebSphere Application Server or CICS, it must not call MQBACK. Transactional control is performed by using the coordinator's APIs.</p>

### 3.4.1 A code fragment

The code fragment in Figure 3-5 on page 38 is a simple example to provide the concept of how to write WebSphere MQ applications.

The example shows the operations that are needed to put a message on one queue and get the reply from another queue. The sample program `amqsreq0.c`, which is shipped with WebSphere MQ, contains similar functions. There is no error handling in this fragment.

**CompCode and Reason fields:** The fields CompCode and Reason contain completion codes for the verbs. CompCode is a simple indicator of success, warning, or failure. The Reason variable is a more detailed code and it provides a more precise cause of any failure.

```
MQHCONN HCon; // Connection handle
MQHOBJ HObj1; // Object handle for queue 1
MQHOBJ HObj2; // Object handle for queue 2
MQLONG CompCode, Reason; // Return codes
MQLONG options;
MQOD od1 = {MQOD_DEFAULT}; // Object descriptor for queue 1
MQOD od2 = {MQOD_DEFAULT}; // Object descriptor for queue 2
MQMD md = {MQMD_DEFAULT}; // Message descriptor
MQPMO pmo = {MQPMO_DEFAULT}; // Put message options
MQGMO gmo = {MQGMO_DEFAULT}; // Get message options

// 1 Connect application to a queue manager.
strcpy (QMName,"QMA");
MQCONN (QMName, &HCon, &CompCode, &Reason);

// 2 Open a queue for output
strcpy (od1.ObjectName,"QUEUE1");
MQOPEN (HCon,&od1, MQOO_OUTPUT, &Hobj1, &CompCode, &Reason);

// 3 Open input queue
options = MQOO_INPUT_AS_Q_DEF;
strcpy (od2.ObjectName, "QUEUE2");
MQOPEN (HCon, &od2, options, &Hobj2, &CompCode, &Reason);

// 4 Put a message on the queue
strcpy(mqmd.ReplyToQ,"QUEUE2");
pmo.Options |= MQPMO_NO_SYNCPOINT;
MQPUT (HCon, Hobj1, &md, &pmo, 100, &buffer, &CompCode, &Reason);

// 5 Close the output queue
MQCLOSE (HCon, &Hobj1, MQCO_NONE, &CompCode, &Reason);

// 6 Get message
gmo.Options = MQGMO_WAIT | MQGMO_NO_SYNCPOINT;
gmo.WaitInterval = 10 * 1000;
buflen = sizeof(buffer - 1);
memcpy (md.MsgId, MQMI_NONE, sizeof(md.MsgId);
memset (md.CorrId, 0x00, sizeof(MQBYTE24));
MQGET (HCon, Hobj2, &md, &gmo, buflen, buffer, 100, &CompCode, &Reason);

// 7 Close the input queue
options = 0;
MQCLOSE (HCon, &Hobj2,options, &CompCode, &Reason);

// 8 Disconnect from queue manager
MQDISC (HCon, &CompCode, &Reason);
```

Figure 3-5 A code fragment



## Comments

The following numbers correspond to the numbers that are shown in Figure 3-5 on page 38:

**1** This statement connects the application to the queue manager with the name QMA. If the parameter QMName does not contain a name, the default queue manager is used. A handle that references the queue manager is stored in the *HCon* variable. This handle must be used in all subsequent verbs that are used on this connection.

**2** To open a queue, the queue name must be moved into the object descriptor to be used for that queue. This statement opens QUEUE1 for output only (open option MQ00\_OUTPUT). The handle to the queue and values in the object descriptor are returned. The handle *Hobj1* must be specified in the MQPUT.

**3** This statement opens QUEUE2 for input only using the queue-defined defaults.

For simplicity in this example, a predefined local queue is assumed. In practice, many applications use a model queue in the MQOPEN verb. In that case, returned values from MQOPEN contain the real name of a dynamically created queue, and that value is put into the ReplyToQ field in the MQMD.

**4** MQPUT places the message assembled in a buffer on a queue. MQPUT has these parameters:

- ▶ The handle of the queue manager (from MQCONN).
- ▶ The handle of the queue (from MQOPEN).
- ▶ The message descriptor.
- ▶ A structure that contains options for the put. The MQPMO\_NO\_SYNCPOINT flag explicitly ensures that the operation does not start a transaction.
- ▶ The message length.
- ▶ The buffer that contains the data.

**5** This statement closes the output queue. Because the queue is predefined, no close processing takes place (MQOC\_NONE).

**6** For the get, the MQGMO\_WAIT option is used to wait for up to 10 seconds for the reply message. The MQGET needs the length of the buffer as an input parameter. Because there are no additional criteria specified, the first message from the queue is read. If no message arrives within the timeout, the Reason code is 2033 (MQRC\_NO\_MSG\_AVAILABLE). The application must be able to handle such failures.

**7** This statement closes the input queue.

**8** The application disconnects from the queue manager.

## 3.5 Triggering

*Triggering* is a mechanism that is used by WebSphere MQ to automatically start applications only when there is work available for those applications to process. It can save system resources because an application does not need to be permanently running, perhaps sitting in a long-running MQGET call, waiting for messages to appear on its input queue.

Triggering relies on several of the attributes that are set on queues, and on a specialized application that is known as a *trigger monitor*. The trigger monitor's purpose is to wait until a controlling *trigger* message arrives, and then start the real application. While this might sound as though one long-running program has been exchanged for another, a single trigger monitor can deal with starting many different applications.

There is usually one trigger monitor running for each type of application that needs to be started. For example, **CKTI** is a trigger monitor that can only start CICS transactions on z/OS while **runmqdnm** is a trigger monitor that starts .Net programs on Windows. For all distributed platforms, **runmqtrm** is the standard trigger monitor that starts local programs or scripts.

## Configuring triggering

There are four requirements to trigger an application:

- ▶ The target queue used as input by the application to be started must have triggering conditions specified:

```
DEFINE QLOCAL(A_Q) REPLACE +
    TRIGGER +
    TRIGTYPE(first) +
    INITQ(SYSTEM.DEFAULT.INITIATION.QUEUE) +
    PROCESS(proc1) +
    DESCR('This is a triggered queue')
```

- ▶ The *process* object that is associated with the target queue defines the name and type of the program that needs to be started. The use of **WINDOWSNT** is a historic artifact but now refers to any currently supported Windows platform:

```
DEFINE PROCESS(proc1) REPLACE +
    DESCR('Process to start server program') +
    APPLTYPE(WINDOWSNT) +
    APPLICID('c:\test\myprog.exe')
```

- ▶ There must be an initiation queue that is defined. This can be any local queue, and it does not need any special attributes. A trigger monitor is the only application that gets messages from this queue. If multiple trigger monitors are running, each trigger monitor must have its own initiation queue defined.
- ▶ The triggered application must be written to expect a defined set of parameters on the command line. It must also be written to behave in a certain way. In summary, any triggered application must use a short wait-time when retrieving input messages, and to loop around its input queue until there are no more messages and the wait-time has expired. The application developer must also expect occasions when there are no messages available for the program to process.

Figure 3-6 on page 41 shows the logic that is needed in a triggered application.

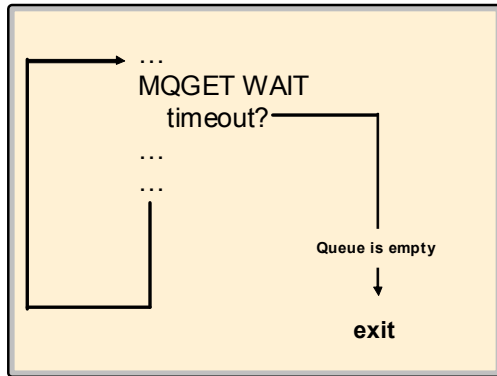


Figure 3-6 A triggered application

All triggered applications must loop until the input queue is empty and have a non-zero timeout on the MQGET.

### How triggering works

Figure 3-7 shows the logic of triggering. Here, Program A sends a message to A\_Q to be processed by Program B. Program B is not running when the message is sent, and it needs to be started.

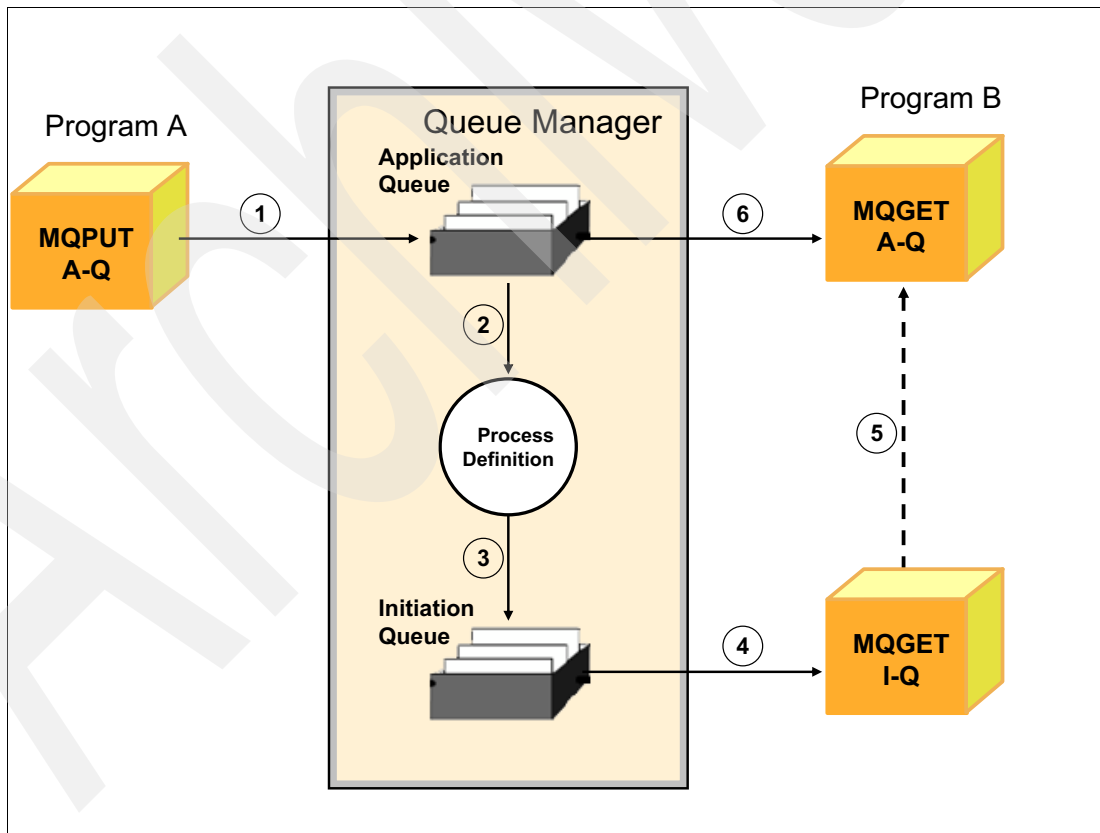


Figure 3-7 Triggering an application

The WebSphere MQ triggering mechanism works in the following manner:

1. Program A issues an MQPUT and puts a message into A\_Q for Program B.
2. The queue manager processes this verb and puts the message into the application queue.
3. The queue manager also finds out that the queue is triggered. It creates a trigger message and looks in the process definition that is named in the queue definition to find the name of the required application and then insert it into the trigger message. The trigger message is put into the initiation queue.
4. The trigger monitor gets the trigger message from the initiation queue.
5. It then starts the program specified.
6. The application program starts running and issues an MQGET to retrieve the message from the application queue.

The trigger type to use depends on how the application is written. There are three choices, but the normal recommendation is to use TRIGGER(FIRST) unless there are compelling reasons to pick one of the other values:

<b>FIRST</b>	A trigger message is put in the initiation queue only when the target queue has changed from empty to non-empty.
<b>EVERY</b>	Every time that a message is put in the target queue, a trigger message is also put in the initiation queue.
<b><i>n</i> messages</b>	A trigger message is put in the initiation queue when there are <i>n</i> messages in the target queue. For example, it might only be worth starting a batch program when the queue holds 1,000 messages.

Regardless of the chosen trigger type, any triggered application must follow the rule about looping until there is no more input. Even with TRIGGER(FIRST), no application can assume that there is always only a single input message.

## 3.6 Configuring a WebSphere MQ client

Many WebSphere MQ applications are run as clients, connecting to a queue manager on a different machine. This section discusses how to define and test the connection between a WebSphere MQ client and its server.

Figure 3-8 shows that the WebSphere MQ client is installed in the client machine. Clients and servers are connected with MQI channels. An MQI channel consists of pair of definitions, the Client Connection (CLNTCONN) and Server Connection (SVCONN) channels.

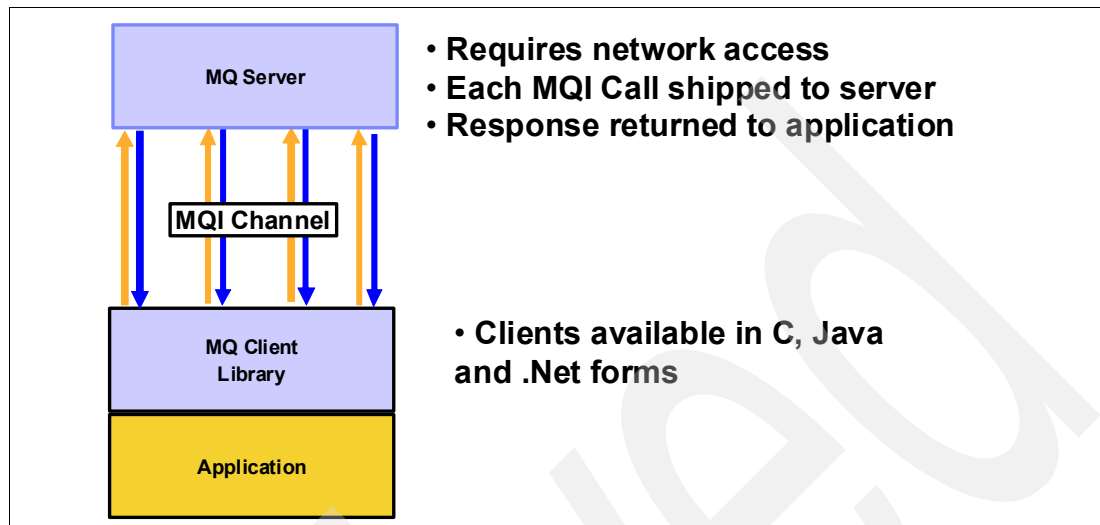


Figure 3-8 Client/server connection

### 3.6.1 How to define a client/server connection

There are three client implementations, in C, Java, and .Net, that are suitable for different application environments. The principles are the same regardless of the language. The main difference is how to define the connection to the queue manager.

#### C clients

The simplest way to define the client connection channel is to set the MQSERVER environment variable, for example:

```
set MQSERVER=CHAN1/TCP/9.24.104.206(1414)
```

where:

- ▶ MQSERVER is the name of the environment variable. This name is fixed.
- ▶ CHAN1 is the name of the channel to be used for communication between client and server. There must be a corresponding SVRCONN channel of this name that is defined on the server.
- ▶ TCP denotes that TCP/IP is used to connect to the machine with the address following the parameter.
- ▶ 1414 is the default port number for WebSphere MQ. It is not required if the listener on the server side is also using this default value, but specifying it makes it clearer if changes are needed in the future.

## Java clients

For the WebSphere MQ client for Java, equivalents to the environment variables are set in the application code. The following example shows the simplest statements to include in a Java program:

```
import com.ibm.mq.*;
MQEnvironment.hostname = "9.24.104.456";
MQEnvironment.channel = "CHAN1";
MQEnvironment.port = 1414;
```

## .Net clients

Like the Java clients, variables can be set in the application program to define the connection:

```
properties = new Hashtable();
properties.Add(MQC.TRANSPORT_PROPERTY, MQC.TRANSPORT_MQSERIES_MANAGED);
properties.Add(MQC.HOST_NAME_PROPERTY, hostName);
properties.Add(MQC.PORT_PROPERTY, port);
properties.Add(MQC.CHANNEL_PROPERTY, channelName);
```

## More complex configurations

There are many ways to define client configurations, and there are many attributes that are possible on these channels. For example, secure configurations might require Secure Sockets Layer (SSL)/Transport Layer Security (TLS) definitions. Designs for high availability might use groups of definitions and multiple addresses to which to connect.

These additional attributes cannot be set by using the environment variables and are usually set by creating the Client Channel Definition Table (CCDT). The CCDT is a file that is created by defining CLNTCONN channels in a queue manager. The generated file is then copied to the machine where the client program is running and referenced during the connection process. An administrator using the CCDT enters the following command:

```
DEFINE CHANNEL('CHAN1') CHLTYPE(CLNTCONN) +
    TRPTYPE(TCP) +
    CONNAME('9.24.104.206(1414)')
```

JMS and XMS programs also have the option of using the CCDT or similar definitions that are created in a Java Naming and Directory Interface (JNDI) repository.

The CCDT and JNDI mechanisms remove the need for connection information from the client application programs. This approach often makes it easier to change connection information administratively. But all client environments also have the option of programmatically defining the full connection details.

## Configuring the queue manager

On the queue manager, the minimum required definition for the corresponding SVRCONN channel is:

```
DEFINE CHANNEL('CHAN1') CHLTYPE(SVRCONN)
```

## 3.7 Security

This document is not the place for a discussion of security; however, there are a few basic points to make especially in relation to client connections:

- ▶ From WebSphere MQ V7.1 forward, the default configuration blocks most client connections. This can be disabled by using the **ALTER QMGR CHLAUTH(DISABLED)** command, but it is discouraged.
- ▶ To get started quickly with a WebSphere MQ client, without completely disabling the channel security rules, set an explicit user ID for the SVRCONN channel:

```
DEFINE CHANNEL('CHAN1') CHLTYPE(SVRCONN) MCAUSER('userA')
```

where userA is a user ID that is defined on the machine where the queue manager is running. If it is also the user ID under which the client application is running, this command permits client connections:

```
SET CHLAUTH('CHAN1') TYPE(USERMAP) CLNTUSER('userA') USERSRC(MAP)  
MCAUSER('userA') ADDRESS('*') ACTION(ADD)
```

- ▶ The userA identity then needs access to the queue manager and resources, such as queues. For the program fragment that is shown in Figure 3-5 on page 38, the following commands are needed on a distributed platform if it connects as a client:

```
setmqaut -t qmgr -m QMA -p userA +connect  
setmqaut -t q -n QUEUE1 -p userA +put  
setmqaut -t q -n QUEUE2 -p userA +get
```

If connecting to a z/OS queue manager, the external security manager, such as IBM RACF®, needs similar definitions. Also, on UNIX and Linux systems, security is group-based, so applying authorizations to a user actually results in changes to group permissions.

- ▶ Client channels can be protected by using SSL/TLS certificates and algorithms. Configuring those values is also beyond the scope of this chapter.

## 3.8 Configuring communication between queue managers

Figure 3-9 shows the key parts and architecture of WebSphere MQ when sending a message between two systems. The application programs use the MQI to communicate with the queue manager. The queuing system consists of the following parts:

- ▶ Queue manager
- ▶ Listener
- ▶ Trigger monitor (optional)
- ▶ Channel initiator
- ▶ Message channel agent (MCA) or mover

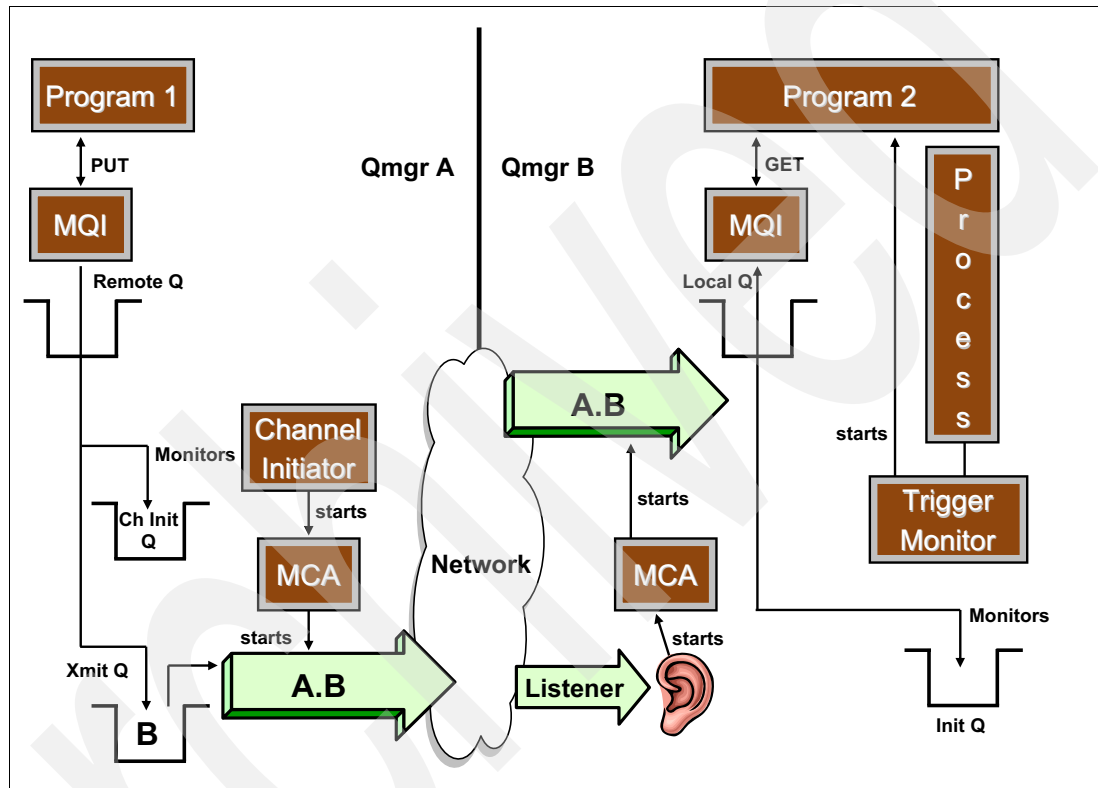


Figure 3-9 Sending messages to a remote application

When the application program wants to put a message on a queue, it issues an MQOPEN call followed by an MQPUT. The queue manager checks whether the queue that is referenced in the MQOPEN is local or remote.

If it is a remote queue, the subsequent MQPUT of a message is placed into the transmission (xmit) queue. The queue manager adds a header that contains information from the remote queue definition, such as the destination queue manager name and destination queue name.

A remote queue definition is not needed if the application explicitly fills in the name of the remote queue manager. The queue manager then works out which transmission queue to use, or returns an error indicating that there is no suitable transmission queue defined. This approach is often used by applications sending reply messages and removes the need for explicit definitions of all possible reply queues.



**Transmission queue:** Each remote queue must be associated with a transmission queue. Usually, all messages that are destined for one remote queue manager use the same transmission queue. The transmission queue is usually given the same name as the remote queue manager.

Transmission is done via channels. Channels can be started manually or automatically. To start a channel automatically, the transmission queue must be associated with a channel initiation queue. Figure 3-9 on page 46 shows that the queue manager puts a message into the transmission queue and another message into the channel initiation queue. This queue is monitored by the *channel initiator*.

The channel initiator is the part of WebSphere MQ that must be running to monitor initiation queues. When the channel initiator detects a message in the initiation queue, it starts the message channel agent (MCA) for the particular channel. This program moves the message over the network to the other machine, by using the sender part of the unidirectional message channel pair.

On the receiving end, a *listener* program must have been started. The listener, which is also supplied with WebSphere MQ, monitors a specified port. By default, this is the port assigned to WebSphere MQ, 1414. When a connection request arrives from the sending queue manager, the listener starts the receiving channel. This receiving MCA moves the message into the specified local queue from where the receiving application gets the message.

**Important:** Both channel definitions, sender and receiver, must have the same name. Create a symmetric set of definitions with a different channel name for the reverse path to send a reply back to the original application.

The program that processes the incoming message can be started manually or automatically. Use triggering to start the program automatically.

### 3.8.1 How to define a connection between two systems

Figure 3-10 on page 48 shows the required objects for connecting two queue managers. The definitions assume two applications communicating via fixed queue names.

Each system needs the following objects:

- ▶ A remote queue definition that mirrors the local queue in the receiver machine and links to a transmission queue (Q1 in system A and Q2 in system B)
- ▶ A transmission queue that holds all messages destined for the remote system until the channel transmits them (QMB in system A and QMA in system B)
- ▶ A sender channel that gets messages from the transmission queue and transmits them to the other system by using the existing network (QMA.QMB in system A and QMB.QMA in system B)
- ▶ A receiver channel that receives messages and puts them into a local queue (QMB.QMA in system A and QMA.QMB in system B)
- ▶ A local queue from which the program gets its messages (Q2 in system A and Q1 in system B)

In each system, you must define the appropriate queue manager objects. The objects are defined in the two columns that are shown in Table 2 on page 48. The queue managers must be running before the objects are defined.

**WebSphere MQ clustering:** Using WebSphere MQ clustering removes the need for some of these definitions, but it requires more initial setup. Adding new systems to an existing cluster is easy, but getting started with a cluster is beyond the scope of this chapter.

The channels can then be configured to start automatically or manually.

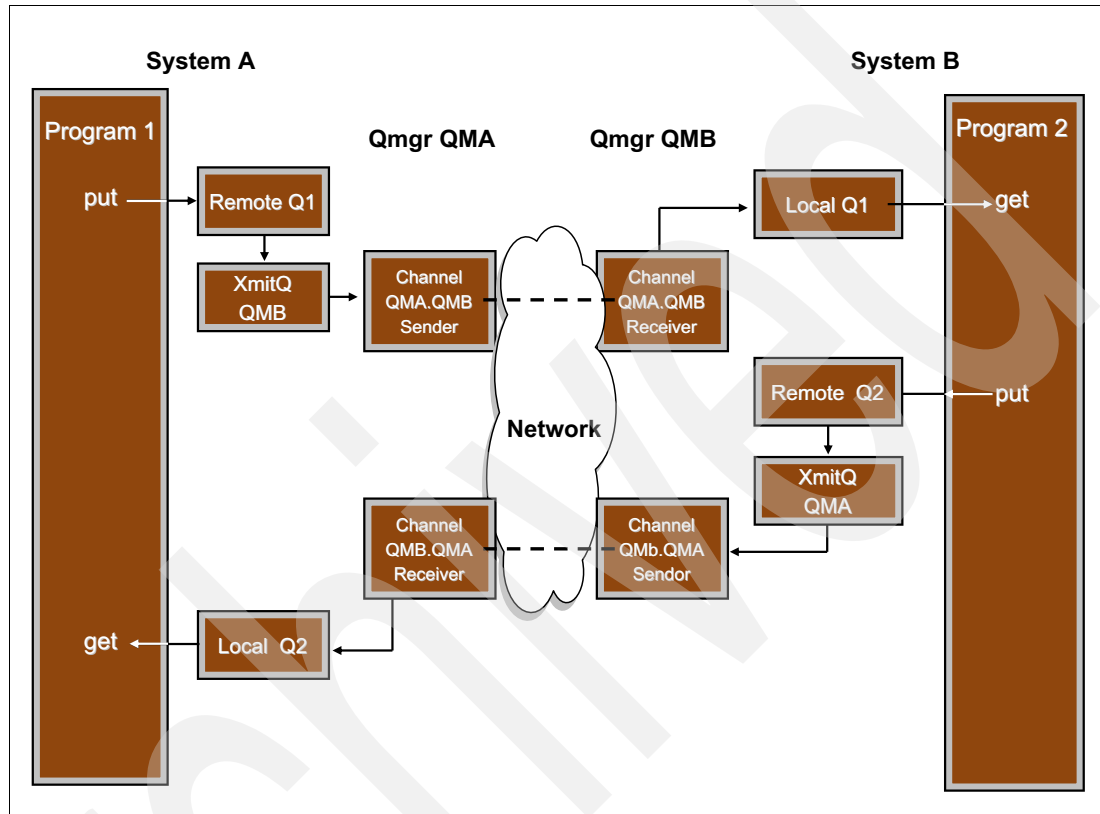


Figure 3-10 Communication between two queue managers

Table 2 Objects defining the connection between two queue managers

System A (QMA)	System B (QMB)
DEFINE QREMOTE(Q1) + RNAME(Q1) RQMNAME(QMB) + XMITQ(QMB)	DEFINE QLOCAL(Q1)
DEFINE QLOCAL(QMB) + USAGE(xmitq)	
DEFINE CHANNEL(QMA.QMB) + CHLTYPE(sdr) + XMITQ(QMB) + TRPTYPE(tcp) + CONNNAME(machine2)	DEFINE CHANNEL(QMA.QMB) + CHLTYPE(rcvr) + TRPTYPE(tcp)
DEFINE QLOCAL(Q2)	DEFINE QREMOTE(Q2) + RNAME(Q2) RQMNAME(QMA) + XMITQ(QMA)
	DEFINE QLOCAL(QMA) + USAGE(xmitq)

System A (QMA)	System B (QMB)
DEFINE CHANNEL(QMB.QMA) + CHLTYPE(rcvr) + TRPTYPE(tcp)	DEFINE CHANNEL(QMB.QMA) + CHLTYPE(sdr) + XMITQ(QMA) + TRPTYPE(tcp) CONNAME(machine1)

### Starting communication manually

After the objects are defined on the queue managers, the sender channels in each direction must be started. Figure 3-11 shows the commands to start the listener and channel for queue manager QMA.

```
C> strmqm QMA
WebSphere MQ queue manager started
...

C> start runmq1sr -t tcp -m QMA -p 1414
C> runmqsc QMA
5724-H72 (C) Copyright IBM Corp. 1994, 2011. ALL RIGHTS RESERVED.
Starting MQSC for queue manager QMA.

start channel(QMA.QMB)
  1: start channel(QMA.QMB)
AMQ8018 start WebSphere MQ channel accepted
end
  2 : end
1 MQSC commands read.
No commands have a syntax error.
All valid MQSC commands were processed.

C>
```

Figure 3-11 Manually starting a queue manager and channels

The first command starts queue manager QMA if it is not already running. The next command starts the listener. It listens on behalf of QMA on port 1414. On Windows, the **start** command runs the listener in a separate window. On UNIX platforms, use the ampersand (&) character to run the listener in the background. The third command starts **runmqsc** in interactive mode, and the channel QMA.QMB is started. For the other queue manager, issue the equivalent commands. Also, start the applications in both systems.

### How to start communication automatically

The *channel initiator* is a special version of a trigger monitor that is used to start channels. By default, the channel initiator is automatically started when the queue manager starts. The only additional requirement for channels to be automatically started is to set the trigger attributes on the transmission queue.

To have the transmission queue triggered, add three more parameters, which are shown in bold in the following example:

```
DEFINE QLOCAL(QMB) REPLACE +
  USAGE(xmitq) +
  TRIGGER
```

```
TRIGTYPE(first) +  
INITQ(SYSTEM.CHANNEL.INITQ)
```

The channel now starts automatically when there are any messages to be transferred.

The DISCONT attribute of a channel controls whether a channel must stop if there are no more messages to be transferred. This attribute is used to release resources, such as network sockets and memory, that might not be needed for a period. Triggering then restarts the channel with no manual intervention if a further message arrives on the transmission queue.

## 3.9 Summary

This chapter provided an outline of many of the core aspects of WebSphere MQ, including how to administer it and how to write application programs. There are many more capabilities in the product to explore, but understanding these basic components is a good start toward learning the next level of detail.

# Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this paper.

## IBM Redbooks

The following IBM Redbooks publications provide additional information about the topic in this document. Note that some publications referenced in this list might be available in softcopy only.

- ▶ *WebSphere MQ V7.1 and V7.5 Features and Enhancements*, SG24-8087
- ▶ *WebSphere MQ V7.0 Features and Enhancements*, SG24-7583
- ▶ *WebSphere MQ V6 Fundamentals*, SG24-7128
- ▶ *Getting Started with WebSphere MQ File Transfer Edition V7*, SG24-7760-00

You can search for, view, download or order these documents and other Redbooks, Redpapers, Web Docs, draft and additional materials, at the following website:

[ibm.com/redbooks](http://ibm.com/redbooks)

## Online resources

These websites are also relevant as further information sources:

- ▶ WebSphere MQ Telemetry, TIPS0876  
<http://www.redbooks.ibm.com/abstracts/tips0876.html>
- ▶ WebSphere MQ Library  
<http://www-01.ibm.com/software/integration/wmq/library/>

## Help from IBM

IBM Support and downloads

[ibm.com/support](http://ibm.com/support)

IBM Global Services

[ibm.com/services](http://ibm.com/services)

Archived





# WebSphere MQ Primer

## An Introduction to Messaging and WebSphere MQ



**Learn the basic concepts of messaging**

**Discover the fundamentals of WebSphere MQ**

**Get started quickly with WebSphere MQ**

The power of IBM WebSphere MQ is its flexibility combined with reliability, scalability, and security. This flexibility provides a large number of design and implementation choices. Making informed decisions from this range of choices can simplify the development of applications and the administration of a WebSphere MQ messaging infrastructure.

Applications that access a WebSphere MQ infrastructure can be developed using a wide range of programming paradigms and languages. These applications can run within a substantial array of software and hardware environments. Customers can use WebSphere MQ to integrate and extend the capabilities of existing and varied infrastructures in the information technology (IT) system of a business.

This IBM Redpaper provides an introduction to message-oriented middleware to anyone who wants to understand messaging and WebSphere MQ. It covers the concepts of messaging and how WebSphere MQ implements those concepts. It helps you understand the business value of WebSphere MQ. It provides introductory information to help you get started with WebSphere MQ. No previous knowledge of the product and messaging technologies is assumed.

**INTERNATIONAL  
TECHNICAL  
SUPPORT  
ORGANIZATION**

**BUILDING TECHNICAL  
INFORMATION BASED ON  
PRACTICAL EXPERIENCE**

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

**For more information:  
[ibm.com/redbooks](http://ibm.com/redbooks)**