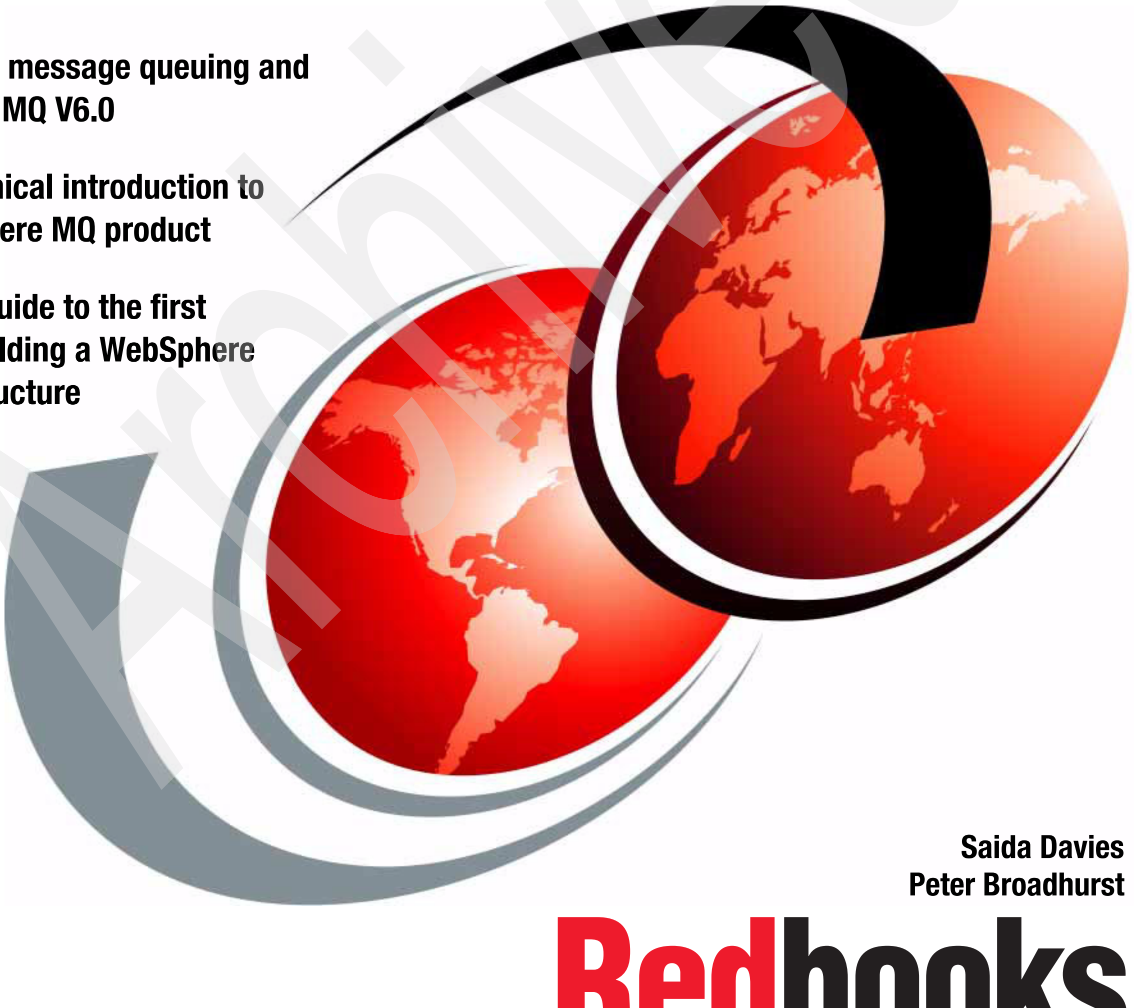
WebSphere® software

IBM

WebSphere MQ V6 Fundamentals

- Overview of message queuing and WebSphere MQ V6.0
- Broad technical introduction to the Websphere MQ product
- Hands-on guide to the first steps of building a WebSphere MQ infrastructure



Saida Davies
Peter Broadhurst

Redbooks

ibm.com/redbooks

International Technical Support Organization

WebSphere MQ V6 Fundamentals

November 2005

SG24-7128-00

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

First Edition (November 2005)
This IBM Redbook edition is an update to the Redpaper *MQSeries Primer*, REDP-0021.
This edition applies to Version 6, Release 0, Modification 0, of IBM WebSphere MQ (product number 5724-H72).

Contents

Notices

Trademarks

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

AIX 5L™
AIX®
CICS®
DB2®
developerWorks®
@server®
HACMP™
IBM®
iSeries™
MQSeries®
OS/400®
Parallel Sysplex®
RACF®
Redbooks (logo) ™
Redbooks™
SupportPac™
TXSeries®
WebSphere®
z/OS®
zSeries®

The following terms are trademarks of other companies:

Java, Java Naming and Directory Interface, JDK, JVM, J2EE, Solaris, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

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

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

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

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

Preface

This IBM® Redbook aspires to improve the breadth of understanding customers have of the IBM WebSphere® MQ product and increase the time value for customers investing in WebSphere MQ. A rounded understanding of the WebSphere MQ product can assist customers build reliable and scalable solutions on the exactly *once delivery assurance* provided by WebSphere MQ.

We describe the fundamental concepts and benefits of message queuing technology. This book is an update of a very popular Redpaper (REDP-0021) based on IBM WebSphere MQ Versions 5.0 to 5.2.

This publication provides a design-level overview and technical introduction for the established and reliable WebSphere MQ product.

The team that wrote this redbook

This redbook was produced by a team of specialists from around the world working at the International Technical Support Organization, San Jose Center.

Saida Davies is a Project Leader for the International Technical Support Organization (ITSO) and has 15 years of experience in IT. She has published several IBM Redbooks™ about various business integration scenarios on distributed platforms and the IBM z/OS® platform. Saida has experience in the architecture and design of WebSphere MQ solutions, extensive knowledge of the z/OS operating system, and a detailed working knowledge of both IBM and independent software vendor operating system software. In a customer-facing role as a senior IT specialist with IBM Global Services, her role included the development of services for z/OS and WebSphere MQ within the z/OS and Microsoft® Windows® platform. This covered the architecture, scope, design, project management, and implementation of the software on stand-alone systems and on systems in a Parallel Sysplex® environment. Saida has received Bravo Awards for her project contributions. She has a degree in Computer Studies and her background includes z/OS systems programming.

Peter Broadhurst is a Software Engineer with IBM Hursley in the United Kingdom. His current job role is as a Service Engineer, within the Level 3 service team for the WebSphere MQ product. In this role, Peter has gained a sound understanding of WebSphere MQ and the functionality it provides, as well as key insights into the solutions developed by customers using technology. Through resolving customer situations, he has developed an empathy for the areas of challenge faced by customers in understanding and building on the concepts of the WebSphere MQ product and the business requirement it fulfils. His career experience also includes working within the test organization for the WebSphere MQ version 6.0 product, where he has gained a key understanding of the new functionality provided by this new version of WebSphere MQ. Peter joined IBM after obtaining a Bachelor of Engineering degree in Computer Engineering from the University of Southampton.

The redbook team would like to thank the following people for their assistance during the initial planning of the redbook:

Craig Both, WebSphere MQ FV Test Team Lead
Software Engineer, IBM Software Group
Application and Integration Middleware Software, IBM Hursley, U.K.

Gary O’Connor, IT Specialist, Application Development
IBM Global Services, Application Management Services, IBM U.K.

Sushil Sharma, WebSphere MQ Distributed L3 Service
IBM Software Group, Application and Integration Middleware Software
IBM Hursley, U.K.

Paul Slater, WebSphere MQ Scenarios Testing, Software Engineer
IBM Software Group, Application and Integration Middleware Software
IBM Hursley, U.K.

Ope Soyannwo, Software Engineer and .NET Consultant
iMeta Technologies, U.K.

Jerry L Stevens, WebSphere MQ technical strategy and planning
IBM Software Group, Application and Integration Middleware Software
IBM Hursley, U.K.

The redbook team would like to thank the following people for reviewing this redbook:

Simon Bluck, WebSphere MQ Level 3 Service
IBM Software Group, Application and Integration Middleware Software
IBM Hursley, U.K.

Andy Emmett, WebSphere MQ Level 3 Service
IBM Software Group, Application and Integration Middleware Software
IBM Hursley, U.K.

Emir Garza
Consulting IT Specialist
IBM Software Group, Application and Integration Middleware Software
IBM Hursley, U.K.

Become a published author

Join us for a two- to six-week residency program! Help write an IBM Redbook dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You'll team with IBM technical professionals, Business Partners and/or customers.

Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you'll develop a network of contacts in IBM development labs, and increase your productivity and marketability.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

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

- Use the online **Contact us** review redbook form found at:

  ibm.com/redbooks

- Send your comments in an e-mail to:

  redbook@us.ibm.com

- Mail your comments to:

  IBM Corporation, International Technical Support Organization
  Dept. 8IB Building 80-E2
  650 Harry Road
  San Jose, California 95120-6099

Summary of changes

This section describes the technical changes made in this book. This edition might also include minor corrections and editorial changes that are not identified.

Summary of Changes
for SG24-7128-00
for WebSphere MQ V6 Fundamentals
as created or updated on December 15, 2005.

November 2005

This revision reflects the addition, deletion, or modification of the following new information described.

New information

This IBM Redbook, *WebSphere MQ Fundamentals*, SG24-7128, adds the following information to the Redpaper *MQSeries Primer*, REDP-0021:

- WebSphere MQ Version 6 changes
- WebSphere MQ Version 6 enhancements
- WebSphere MQ Version 6 basics

1

Overview

This book provides the level of technical detail about the IBM WebSphere MQ product required for individuals to make informed application and infrastructure design decisions prior to implementing a WebSphere MQ infrastructure or beginning development of a WebSphere MQ application.

This publication is intended to be of use to a wide-range audience.

Chapters 1 to 4 provide an introduction to the concepts of the WebSphere MQ product and the challenges for which it provides solutions.

Chapters 5 to 8 provide a broad summary of the technical concepts and implementation details of building and accessing WebSphere MQ infrastructures.

Chapters 9 and 10 provide a hands-on introduction to the concepts described in Chapters 1 to 8.

Chapters 11 and 12 provide further technical information to help secure a WebSphere MQ infrastructure and troubleshoot any problems encountered.

The level of technical detail provided increases progressively through this book. We cover a broad range of topics from across the WebSphere MQ product and documentation.

Throughout this publication, we introduce details of new areas of function for WebSphere MQ Version 6.0, such as the WebSphere MQ Explorer, 64-bit queue managers on UNIX®, and changes to the cluster workload balancing algorithm.

The hands-on sections of this book are located after the technical overview chapters. Depending on your preferences, you can work through the hands-on sections in combination with reading the technical sections. This can provide a working environment in which to gain an understanding of the concepts.

WebSphere MQ enables new applications to communicate with existing applications. In addition, a WebSphere MQ infrastructure abstracts this communication above individual communication protocols or the identities of individual machines within a network.

Applications that access a WebSphere MQ infrastructure can be developed using a wide range of programming paradigms and languages. These applications can execute within a wide range of software and hardware environments. You 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 publication does not focus on interacting with WebSphere MQ using a particular programming language. Instead, it covers the core WebSphere MQ concepts that should be understood when developing any application that accesses a WebSphere MQ infrastructure.

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

Therefore, this book does not distinguish between WebSphere MQ administrators and WebSphere MQ application programmers. A broader understanding is key to making informed design and implementation choices for both the infrastructure and the applications that access that infrastructure.

In Chapter 2, “Concepts of message queuing” on page 5, we introduce the fundamental concepts of message queuing.

Next, in Chapter 3, “Facilities for message queuing provided by WebSphere MQ” on page 25, we provide an overview of the facilities provided by WebSphere MQ within each of the areas introduced in Chapter 2.

Chapter 4, “Designing applications that access a WebSphere MQ infrastructure” on page 49 describes how applications can build on the facilities provided by WebSphere MQ to provide an access to the services of a system.

Chapter 5, “Understanding and configuring queue managers” on page 83 provides fundamental details of the queue managers which form a WebSphere MQ infrastructure.

In Chapter 6, “Technical introduction to message queuing” on page 123, we describe the specific actions performed against a queue manager by an application and how a queue manager is configured to provide facilities to the applications which access it.

Chapter 7, “Queue manager intercommunication and client connections in WebSphere MQ” on page 155 describes how a WebSphere MQ infrastructure is extended beyond a single machine over a network.

Then in Chapter 8, “Queue manager clusters” on page 181, we describe how simplification of administration and flexibility in scaling capacity can be provided by joining queue managers together in queue manager clusters.

In Chapter 9, “Hands-on introduction to messaging with WebSphere MQ” on page 217, we provide a hands-on introduction to performing point-to-point and publish/subscribe messaging using JMS with WebSphere MQ.

Chapter 10, “Hands-on guide to building WebSphere MQ infrastructure” on page 265 provides a hands-on introduction to building WebSphere MQ infrastructures using hub and spoke architectures or flexible queue manager clusters.

Chapter 11, “Securing a WebSphere MQ infrastructure” on page 301 provides and introduction to the security model of WebSphere MQ and how it can be used to secure a WebSphere MQ infrastructure.

In the final chapter, Chapter 12, “Troubleshooting” on page 321, we provide important information for all users of WebSphere MQ for understanding the information provided by WebSphere MQ and how to resolve issues you might experience.

2

Concepts of message queuing

In this chapter, we describe areas where a business can face challenges when developing information technology (IT) solutions. We introduce message queuing technology and outline the role it can play in overcoming these challenges.

We cover the following topics:

- Core concepts
- Simplification
- Scalability and performance
- Reliability and data integrity
- Security
- High availability
- Monitoring and accounting

2.1 Core concepts

A large number of different technologies can exist within the information technology infrastructure of a business. Often these include different hardware platforms, programming languages, operating systems, and communication links.

Services are built on this infrastructure, where each service provides a facility to perform an action or gain information. These services can be provided for use internally by the business or for use by suppliers and customers.

As a whole, these services and the infrastructure of the business that supports them, form a *system*. The general term *node* is used in this book to represent any point within such a system that can host a service, request a service, or connect other nodes together. These nodes can be individual hardware servers, or multiple nodes can exist within the software components of a single server.

New services can introduce new technologies into a system. However, these services often need to interact with existing services, using existing technologies.

A business might also want to interface their system with the technologies used in the systems of other businesses, or provide external interfaces into their system for existing and potential customers or business partners.

2.1.1 Middleware

Software *applications* developed by a business to provide and request services need to communicate with the existing services provided by the system. These services reside on the nodes within the system, where different nodes might be built on different infrastructure components and technologies.

Middleware technologies provide a layer of abstraction between the infrastructure components and the applications that access those components in order to provide services. This middleware layer then becomes a part of the infrastructure of the system: a common layer bridging the nodes of the system.

This middleware layer can simplify the development and maintenance of applications hosting or requesting services, enabling development to focus on the business logic required by a service, rather than the complexities of accessing existing services provided by the system.

Message queuing is a middleware technology that greatly simplifies communication between the nodes of a system and between the nodes that connect systems together. It allows services to communicate in a flexible way that does not require detailed knowledge of a target service or the current availability of that service. Reliable communication can be achieved regardless of the complexity of the infrastructure connecting the nodes in the system.

This technology builds on two basic concepts: messages and queues.

2.1.2 Messages

A node in a system often needs to communicate information to, or request a service from, another node in that system or an interconnected system. This piece of information or request can be considered a message.

The message can contain simple character data, numeric data, complex binary data, a request for information, a command, or a mixture of all of these. It is important that any target nodes that receive a message understand the format of that message. However, the messaging infrastructure that connects those nodes does not require this same understanding. The infrastructure must only ensure that the integrity of the information contained in a message is maintained and that the message is correctly delivered to its destination.

The messaging infrastructure understands differences between the underlying hardware and software on which individual nodes are running. Some conversion of character and numeric data might be required in order for the data to be readable on these different hardware and software platforms. The messaging infrastructure can be configured to perform this conversion transparently so that each message remains valid when it reaches each destination.

In order to transport the message to its destination, extra information might be added to this message while it travels through the messaging infrastructure. However, this information is separate from the information the message contains, in the same way that an envelope is separate from the letter that it contains.

2.1.3 Queues

A queue is a container of messages. New messages are placed at the end of the queue, and messages are usually retrieved from the front of the queue. Figure 2-1 on page 8 illustrates messages passing through a queue.

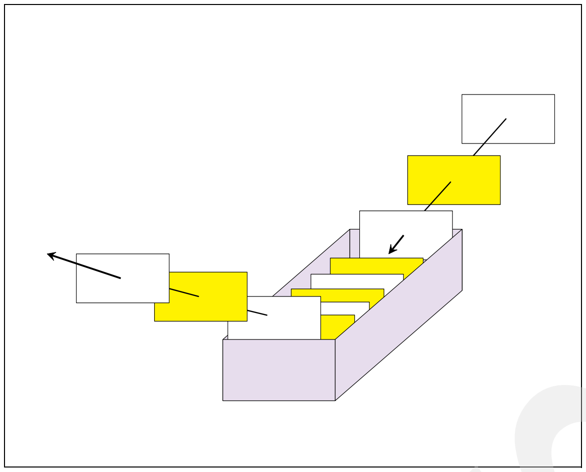


Figure 2-1 Messages passing through a queue

It is also possible for messages to be retrieved using identifiers associated with those messages. Messages can be given priorities so that high priority messages are always retrieved before lower priority messages, or be logically grouped so that one message is dependent on another message.

However, queues are not designed to replace databases of information. They are optimized to contain small numbers of messages that flow through the queue before reaching their destination.

This simple and powerful concept provides the basis of a scalable and reliable messaging infrastructure—a *message queuing* infrastructure. Ways the queue concept is used within a message queuing infrastructure include:

- Providing a buffer between the producer and consumer:
  A service that receives, consumes, and processes messages might not be able to process each message as it arrives, because it might be busy performing processing on another message or it might be unavailable for any other reason. Messages might not arrive at a constant rate and during busy periods; demand for the service might temporarily exceed its ability to supply. By placing a queue between the producer of a message and the consumer, the producer can send a message without knowing whether the consumer of that message is currently available.

  This is called *asynchronous* communication, because the producer can continue processing as soon as the message is placed in the queue. Contrast this with *synchronous* communication, where the producer must wait for the consumer to become available and complete processing on the message before being able to continue.

- Batch processing:
  A service might not want to process and consume each message as it arrives. Instead, it might want to wait until a certain threshold of messages has arrived on a particular queue and process them in one batch, or perform processing of all messages that have arrived during a given time interval. This allows the service to exploit efficiencies of scale in processing the data and optionally to perform processing at times when other related systems and services are quiet.
- Providing buffers between intermediate nodes in a system:
  In order for messages to flow from producers to a consumers, they might need to flow through intermediate nodes in the system. The number of nodes a message travels through, as well as the location of the final destination for that message, might not be known by the producer of a message and might be subject to change. The network links between those intermediate nodes, or the intermediate nodes themselves, might not always be available. By placing a queue as a buffer between each node in the system, messages can wait at any point in the system until a node or network link becomes available and then automatically flow through to the next node.

2.1.4 Point-to-point messaging

Many messages are intended to be consumed exactly once. The point within the system that consumes the message might or might not be known to the producer of the message. However, the producer provides enough information for the messaging infrastructure to deliver the message to a single consumer.

In point-to-point messaging a message arrives once, and once only, at a single and correct destination.

Point-to-point messaging can be split into the following two general categories:

- Send and forget messaging:
  A message is sent to a service that performs an action based on that message. In processing this message, the service might perform further messaging, but the originator of the request does not have direct knowledge of any further messaging that occurs.
  For example, the message might contain information to update a database. Or, the message might contain details of an event that has occurred and requires action, such as sending messages to system administrators.

- Request/reply messaging:
  A message is sent to a service that performs an action based on that message and then returns a reply to the originator of the message.
  For example, the request message might contain a query for information in a database that is sent back in the reply. Or, the request message might contain details of an action to perform that can have multiple outcomes, the result of which is sent back in the reply.

2.1.5 Publish/subscribe messaging

When using point-to-point messaging, each node that requires information must have knowledge of the individual services that can provide that information. Equally, services that provide information must have knowledge of the individual nodes to send information to and know when to send information to those nodes.

The send and forget, and request/reply, facilities provided by a message queuing infrastructure within the point-to-point messaging model provide solutions in many circumstances. However, for the following forms of information, the point-to-point model of messaging has limitations:

- State information:

  Some information can change over time, and a node within the system accessing that information might need to track changes to that information. These nodes need to gain the information initially when they begin processing and then be informed of any changes that occur to that information.

  This form of information is called *state information*. An example is the current share price of a company; this has a value at any point in time, but this value can also change at any point, and a node might need to track a stock price over time.

  To be kept informed of changes to information using request/reply messaging, a node must send regular requests to a service for the current state of that information. To do this, the node must have knowledge of nodes that host the services to provide this information. This is called *polling*.

  Polling services is not an efficient way to become notified of changes to state information. A large number of the requests might return the same information, and the requesting node is not notified of changes until the next request is issued. Shorter polling intervals can increase the accuracy of the information known to a node, but also increase resource requirements on the services and infrastructure of the system.

- Event information:

  Some actions need to be performed by a service whenever a particular event occurs.

Information that describes an event that has occurred is called *event information*. An example is each time an item held in stock is sold; a stock control service might need to update stock information when this event occurs, or reorder that item from a supplier.

Point-to-point messaging facilities allow individual event messages to be sent to services as they occur. However, each node that detects or produces events must have knowledge of all nodes hosting services that require those events and send a message to each service in turn.

The publish/subscribe messaging model addresses these limitations.

In publish/subscribe messaging, any number of consumers of information can receive messages that are provided by one or more producers of that information. In this case, a producer of information is called a *publisher* and a consumer of that information is called a *subscriber*.

Publish/subscribe messaging provides the concept of a *topic* on which any number of interested consumers of information can *subscribe* in order to register their interest. This is similar to the way that a person might subscribe only to magazines about topics in which they are interested. Each topic provides particular event or state information.

A publisher can send messages containing information about a particular topic to all subscribers to that topic, without any knowledge of how many subscribers there are, or the details of the nodes that host those subscribers. Because of this, publish/subscribe messaging completely decouples the provider of the information from the consumer of that information.

For *state information*, a subscriber can request the current state of the information when it first begins processing, and by subscribing to the topic for that state information, it is automatically informed of changes.

For *event information,* a subscriber is automatically informed of events by subscribing to the topic for those events.

In order to facilitate this publish/subscribe capability, a *broker* is required to hold information about which subscribers have subscribed to which topics and how to deliver messages to them. A publisher can then publish messages using the broker to all subscribers on that topic without knowing the details of those subscribers.

There can be multiple publishers for a particular topic, and a subscriber to information about one topic can also be a publisher of information about other topics.

The publish/subscribe messaging model is a powerful concept that allows information updates to logically flow from the producers of information to all consumers of that information. It provides the flexibility, because the number of subscribers can grow or shrink dynamically without the knowledge of the publishers.

2.2 Simplification

The process of transporting a message from its source to its destination can be complex and involve transport through multiple intermediate nodes over communications links of varied types, speeds, and quality. Sometimes, communications links are down, and a message is not able to reach its final destination until that link is restored. In addition, the node to which the message is being sent, or an intermediate node, might be unavailable or busy at the time a message is generated.

One important feature of a message queuing infrastructure is that the application development resources required to deal with these considerations, and others discussed later in this book, are significantly reduced.

Using a message queuing infrastructure, the following steps summarize the steps required to reliably send a message to a destination:

1. Supply enough information to the message queuing infrastructure to identify a valid destination, or destinations, for the message.
2. Put the message into the message queuing infrastructure.
3. From the success of step 2, the application can determine that the message reaches its destination successfully.

Note: In 2.4, “Reliability and data integrity” on page 16, we discuss assurances of delivery provided by a message queuing infrastructure.

2.2.1 Development focuses on business logic

Developing business applications to provide and access services can be a costly process requiring significant resources. A message queuing infrastructure provides application programmers more time to concentrate on the business logic involved in providing a service, rather than communications logic and complicated failure processing.

Without a secure, flexible, and reliable message queuing infrastructure, the application programmer might need to apply significant extra development to the following issues:

- A remote service might be temporarily unavailable at the time the local application needs to connect.
- A communications link between the local application and a remote service might be unavailable or might fail part way through transmission.
- A remote service might be running on a different hardware platform to that of the local application, with fundamental differences in character and numeric data formats.
- The route to a remote service might have changed since the application was first written.
- There might be multiple instances of a service in the network, and the application might need to choose to which instance of this service to connect.
- The action performed by a particular service might be sensitive, and as such, the service must only be invoked by authorized applications.
- The data being sent over a communications link to a remote service might be sensitive and require protection. The application might need to perform encryption of this data.
- A failure in communication with a particular service might affect the validity of other actions that have already been performed by the application.
- Current capacity requirements might change over time, typically increasing.

2.2.2 Application maintenance and portability

Another important consideration is how easy it is to maintain and interface with applications after they are originally developed. The individual developer, or team of developers, that implemented an application might not be available when changes need to be made to an application. The requirements placed on an application might be extended over time, or other nodes within or outside the system might need to access existing services. These new nodes might contain different hardware and software components, or be connected by different communication links.

By simplifying the considerations that need to be made during application development, and focusing development efforts on business logic, the structure of the code can be made more easily maintainable by future developers who need to make changes.

Software that resides between applications and the components of the underlying infrastructure with which they interact is called *middleware*.

Middleware can provide functionality that is not specific to an individual programming language, hardware platform, or operating system. Message queuing technology is a form of middleware, providing asynchronous communication between applications.

By using a middleware layer, application code can be moved or extended onto nodes with different underlying infrastructure components, with greatly reduced development efforts.

The process of changing an application to execute on a node with different underlying infrastructure components is called *porting*, and how easy it is to port an application to different infrastructure components is called the application's *portability*.

Access to existing services that do not use a middleware layer can often be extended in a flexible way by developing a *proxy*. A proxy provides an interface between the middleware layer and the existing service. New applications can then access the existing service through the middleware layer, using the proxy, rather than accessing the existing service directly.

2.3 Scalability and performance

When planning, designing, developing, and provisioning a service within the IT system of a business, it is important to consider whether a service can meet the requirements placed on it and can grow to meet requirements in the future.

The following list provides general descriptions of how some terms used in this book apply to a service within the IT system of a business:

- Performance: The time taken between submitting a request for a service and completion of that service. How the start and end points of a service are determined are specific to the function being performed by the service.
- Load: The number of attempts being made to request a service within a given time interval.
- Capacity: The number of requests for the service that can be processed by the system as a whole within a given time interval.
- Scalability: How easily the capacity of the system can be increased to cope with increased load and how this affects performance.

It is important to consider all of the above factors, rather than any one in isolation:

- A service that provides high performance for each successful request, but only allows small numbers of requests to be processed concurrently, might not be able to cope with loads placed on it.

- A service that provides high capacity through the operation of many instances of the service concurrently, but provides low performance for each individual instance, might be unacceptably slow.

Initial provisioning of resources for a service must ensure that the performance and capacity of the service are acceptable under anticipated loads. Consider reliability and security in these initial requirements, because technologies that increase reliability and provide security can affect performance.

It is important to consider the scalability of the service at the same time as considering the initial performance and capacity. A service that scales well allows additional resources to be added to the system to quickly increase capacity and cope with increased load, while not adversely affecting performance. If scalability is not initially considered, increasing the capacity of a service might require a significant amount of redevelopment or adversely affect performance.

Another important consideration is how the system as a whole copes during periods where the load exceeds capacity for a particular service. This can often be a temporary condition that occurs during peak time periods. It can also occur gradually or suddenly over time as demand for services increases, signifying that capacity needs to be increased. If this is not considered, negative outcomes can include a combination of the following possible results:

- The system is unable to process the additional requests for that service.
- The performance of each request for the service degrades to an unacceptable level.
- The performance of other services provided by the system are degraded.
- A fundamental failure occurs, such as reaching a hardware resource limitation. This results in some services being unavailable for a period of time. This is often called an *outage*.

The scalability and performance features provided by a message queuing infrastructure, combined with careful design and planning, can provide a system that scales efficiently. Resources can be allocated and deallocated when required to meet the capacity and performance requirements of the system under the changing loads experienced within a business environment.

2.4 Reliability and data integrity

Data that passes through a message queuing infrastructure can be broadly separated into two categories:

- Query data: This is transient data being sent through a system, derived from data that is stored safely within the system. If the query is lost, it might be inconvenient to the requester of that information, but the original data remains in the system and can be requested again.
  An example is querying the current status of an order; if the request for this information is lost, or the response is lost, it does not affect the actual status of the order and the request can be submitted again.
- Business-critical data: This is data that is not stored elsewhere in the system. If this data is lost, important information, or a change in state within the system, is lost.
  An example is a message confirming a payment for an order; if the message confirming the payment is lost, the status of that order can become inconsistent. The customer might have paid money, or been provided a receipt, but the state of the order in the system does not correctly reflect that.

A large proportion of the messages that flow through a message queuing infrastructure contain business-critical data. Therefore, it is important that a message queuing infrastructure provides assurances that these messages are not lost.

There are some performance implications in providing this assurance, so messages containing query data can be marked as such to increase the performance of passing that message through the infrastructure.

2.4.1 Exactly once delivery

In addition to ensuring that messages containing business-critical data are not lost as they pass through a message queuing infrastructure, it is important that messages reach their intended destination, that they reach it once, and that they reach it once only.

If a message has been sent to a valid destination within the system, the message queuing infrastructure must assure that the message is delivered to that destination. However, the target destination, or intermediate nodes, might be unavailable for periods of time. In this case, messages wait within the system until they are able to continue to their destination.

Requirements can be different for query data, because the requester of information might not want to wait indefinitely for a response. Therefore, messages passing through the infrastructure can be set to *expire* if they remain at any node within the system for longer than a specified period of time.

If a message reaches its destination twice, the results might be unpredictable. For example, if a message confirming payment is received twice, this might cause it to appear that the customer has overpaid and for their account to be in credit incorrectly. A message queuing infrastructure must provide assurances that a single message cannot arrive at its intended destination twice or at multiple destinations unexpectedly.

If a message cannot be delivered to its target location, the behavior must be well-defined so that message can be found and manually routed to the correct destination. This can occur due to a change in the infrastructure that has removed a destination, or due to incorrect specification of a destination by the producer of a message. A message must not be discarded after it has entered the message queuing infrastructure, unless it has been specifically marked to expire or as containing query data only.

The concept of exactly once delivery encompasses all of these considerations. WebSphere MQ assures exactly once delivery.

2.4.2 Units of work

Actions performed by an application cannot always be considered in isolation. For example, a service might read a message, update some customer information, and then send a reply. These are three separate actions, but if any one of these fails to complete, none of them should complete.

These actions are said to be within a *unit of work*. Units of work include sending messages, receiving messages, and updating databases of information. The message queuing infrastructure must allow units of work to complete or fail as a unit so that no individual action within that unit can complete without all actions completing. It must also be possible for the application to explicitly fail this unit of work at any point.

2.4.3 Failure handling

Failures within a business IT system can occur without warning and must be coped with in a way that does not leave the system in an inconsistent state. Examples of such failures include:

- Intermittent communication failures
- Planned and unplanned outages of other services within the system
- Failures due to resource limitations and hardware errors
- Failures due to software errors

- Failures due to configuration and infrastructure changes

A business IT system needs to be designed to consider these failures and provide consistent and tolerant handling for them. Facilities within the message queuing infrastructure simplify this process by providing a fault tolerant infrastructure for services to reside within and by significantly reducing the amount of application logic required to detect and process these failures.

These features can also help eliminate single points of failure within the system, thus enabling services to remain available when failures occur.

Note: We discuss fault tolerance and single points of failure in more detail in 2.6, “High availability” on page 20.

2.4.4 Quality assurance (QA) environments

A particularly important part of development and deployment of a new service into a business IT system is *testing*. Early detection of problems can significantly reduce the impact of encountering that problem and reduce the resources required to fix that problem.

A recommended technique for application development, testing, and deployment is to have two separate environments:

- Production environment: This is the environment on which the actual system runs and provides real services that are used and made available within the business, outside of the business, or both.
- Quality assurance (QA) environment: This is the environment on which development and testing is performed. This environment must match the production environment as closely as possible, including hardware, operating systems, infrastructure software, configuration, and all services. Any changes that are planned for the production environment need to be developed and tested in the QA environment before applying them in the production environment. Loads on this environment need to be simulated.

For a simple system, these environments can share infrastructure hardware and software. However, full separation of these environments can provide extra benefits and reduce the possibility of changes in the QA environment affecting the availability of services in the production environment.

The patterns and levels of usage of services simulated in the QA environment need to match the usage that is experienced in the production environment as closely as possible. The more closely these match, the more likely it is that potential problems are recognized in the QA environment before being encountered in the production environment.

If a problem is observed and can be re-created in the QA environment, the solution can also be developed and tested in the QA environment. Developing this solution can be an invasive process and require steps that are not be advisable in the production environment without prior testing. In the QA environment, these changes can be made, logged, and tested without affecting the production environment.

We recommend regular application of maintenance to all software within a system. By applying this maintenance to the QA environment prior to the production environment, the possibility of an unexpected consequence of applying maintenance affecting service availability is greatly reduced.

2.5 Security

It is important that security is considered carefully within a business IT system. Access to services and information provided within that system must be controlled, and the integrity of sensitive information transferred across public networks needs to be considered.

2.5.1 Security of access

It is often important to ensure that services within a business IT system are only available to authorized users or applications.

Malicious or unauthorized use of a service might cause corruption of the information contained in a system, or expose sensitive information to a wider audience than has been granted access to that information.

Generally, securing access to services involves the following two problems:

- Identifying the entity attempting to access services
- Preventing an identified or unidentified entity from accessing services to which it should not have access

A message queuing infrastructure must provide facilities to reliably identify an entity that is attempting to access a particular service, as well as methods for deciding whether that entity is authorized to access that service.

This can be especially important when services are provided to entities external to the company, such as suppliers or customers. A message queuing infrastructure must provide a solid and secure framework to allow these external entities to access services.

2.5.2 Security of communications

Nodes within a system might be connected by varied public or exposed communication links. Protection of those communications links might be required in order to prevent external intervention of data.

Technologies to protect communication links provide security against a third party reading or modifying sensitive data. They also protect against attempts to impersonate an entity that is authorized to access services within the system.

External intervention in a system through exposed communication links can cause a failure in data integrity or cause services to become unavailable.

2.6 High availability

Services that are provided by a business IT system might be relied on for the day-to-day operations of that business. Outages, where a particular service or group of services are unavailable, can affect the operation of the business and costs can be incurred as a result.

These outages broadly split into two categories:

- Planned outages: Periods when a service or services are unavailable in order to perform planned work on those services. These include:
  - Applying maintenance to software
  - Changing the configuration of the infrastructure
  - Backing up or reorganizing data
- Unplanned outages: Periods when a service or services become unavailable unexpectedly. These can be caused by:
  - Hardware or software failures
  - Resource shortages
  - Failures in data integrity

The objective of providing high availability of services is to minimize planned and unplanned outages. There are a number of ways in which this can be performed, including:

- Increase the reliability of the system:
  This is dependent on the reliability of all components within that system, including the hardware and software infrastructure, communication links, and the custom applications that perform services. Using a stable and reliable message queuing infrastructure, which provides simple interfaces to minimize the complexity of services, can contribute significantly to the reliability of the system as a whole.

- Reduce outages caused by updates to components:
  If component updates and maintenance can be applied without making services unavailable, planned outages can be reduced. Applying updates first in a QA environment can reduce the chance of changes in behavior causing an unplanned outage.
- Ensure that the system recovers efficiently following a failure:
  If a system is able to recover efficiently, or automatically, after a failure has occurred, without a loss in the integrity of the information within that system, the impact and length of any unplanned outage is reduced.
- Minimize the impact on services from an individual failure:
  If a failure does occur within a component of the system, it is likely to have some effect on the system and the services it provides. Within the context of message queuing, these effects can be split into the following categories:
  - Service availability
  - Message availability

2.6.1 Service availability

If a failure occurs within a component of the system, it affects all other components that depend on it. These components, in turn, can affect other components, and eventually one of more services within the system can be affected.

If there is only one instance of a service within the system, or if all instances of a particular service share a dependency on a particular component, this represents a single point of failure within the system.

By providing multiple instances of a service within the system, which are able to operate independently, a service can remain available for new requests after some instances of that service have failed.

Within the context of message queuing, this requires that new messages that would have reached, or passed through, queues hosted on nodes that have failed should be routed to other nodes in the system.

2.6.2 Message availability

Messages might pass through multiple nodes within the system before eventually reaching their destination, or might remain at a destination for a period of time before they are processed by a service. The service itself might take some time between consuming a message and successfully completing all actions on that message.

If a component within the system fails, nodes that host queues and provide services might fail as a result. This failure can be immediate and provide no time for the infrastructure or applications to react, such as in the case of a hardware failure. While a node is unavailable, all messages that reside on queues hosted by that node are also unavailable. These are called *stranded messages*.

If stranded messages contain business-critical data, reliable recovery of those messages is important. These messages must be recovered while retaining the integrity of data that they contain.

Units of work that were in progress at the time of a failure must also be resolved correctly. A service might have been in the middle of performing message processing and might not have completed this processing.

Recovery of messages containing business-critical data can also be time critical. If a fundamental failure occurs, it can take time and manual intervention to recover the affected resources. Therefore, manual or automated actions might be required to make those messages available using resources that have not been affected.

2.6.3 Disaster recovery

Some disastrous failures can affect all resources for a particular node, or even multiple interconnected nodes hosted at the same physical location. Making services available again after these disasters occur, or recovering data that was hosted on affected nodes, require careful planning and forethought.

A business can plan for these circumstances by performing regular *backups* of data hosted at nodes within their system that host business-critical data. Using a backup strategy, the data that can be recovered from a failed node, or nodes, might not be fully up to date. However, backup strategies can significantly reduce the quantity of data that is lost following a disastrous outage.

2.7 Monitoring and accounting

Information about the operation and usage of a system has a number of important uses: It can provide indications about the health of the system, indications of the performance and available capacity of the system under different loads, and can allow tracking of usage for individual entities.

2.7.1 Performance monitoring

Performance monitoring, in the context of a message queuing infrastructure, involves measuring the number and duration of actions that occur when services access queues or when messages pass through queues on intermediate nodes. You can use this information to infer performance and capacity information about those services and the infrastructure that supports them.

This can be helpful when provisioning resources for a service, or when deciding whether additional resources are needed to scale the capabilities of a service. Monitoring performance can also be useful to identify any sudden or gradual changes in usage patterns to anticipate problems before they occur.

2.7.2 Accounting

Accounting, in the context of message queuing, involves gathering information about the usage of particular queues by particular entities. These entities can be an application providing a service from that queue, an internal user, or an external user.

These accounting statistics allow patterns of usage to be identified and analyzed. One application is to ensure that a promised quality of service is being provided to a particular entity. Another application is to provide facilities for charging based on the usage of a particular service by a particular entity.

3

Facilities for message queuing provided by WebSphere MQ

This chapter introduces the specific features of the message queuing product IBM WebSphere MQ. For each area of challenge introduced in Chapter 2, “Concepts of message queuing” on page 5, we provide an overview of the related features provided by WebSphere MQ.

We discuss the following topics in this chapter:

- Core concepts
- Simplification
- Scalability and performance
- Reliability and data integrity
- Security
- High availability
- Monitoring and accounting

3.1 Core concepts

IBM WebSphere MQ is an established and reliable message queuing middleware platform. During more than 10 years of development, WebSphere MQ has grown to provide flexible and reliable solutions that address the breadth of considerations introduced in the previous chapter.

A message queuing infrastructure built on WebSphere MQ technology helps provide an available, reliable, scalable, secure, and maintainable transport for messages—with exactly once delivery assurance.

3.1.1 WebSphere MQ message queuing infrastructure

The nodes within a WebSphere MQ message queuing infrastructure are called *queue managers*. Multiple queue managers can run on a single physical server, and queue managers can run on a large variety of different hardware and operating system combinations.

Each queue manager provides facilities for reliable message queuing. Queue managers on all platforms provide facilities for message queuing using a point-to-point model, including request/reply and send and forget messaging, as described in the previous chapter.

WebSphere MQ Version 6.0 queue managers, with the exception of WebSphere MQ for z/OS, also provide a publish/subscribe broker for message queuing using a publish/subscribe model.

The queue managers maintain the queues of the message queuing infrastructure and all of the messages that reside on those queues waiting to be processed or routed. Queue managers are tolerant to failures, maintaining the integrity of the business-critical data flowing through the message queuing infrastructure.

The queue managers within the infrastructure are connected with *channels*. 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.

Many changes can be made to this configuration transparently to the applications, which provide and access the services in the system.

3.1.2 Facilities for building a WebSphere MQ infrastructure

WebSphere MQ V6.0 provides the *WebSphere MQ Explorer*, which is a graphical user interface (GUI) used to configure and monitor the queue managers within a

WebSphere MQ infrastructure from a desktop workstation. This includes the administration of queue managers hosted on the z/OS platform.

WebSphere MQ provides a scripting interface called MQSC to perform administration on all platforms. Panel driven administration interfaces are also provided on iSeries™ and z/OS.

WebSphere MQ *objects* are defined on each queue manager. These configure the queues hosted on that queue manager and the interaction between that queue manager, applications, and other queue managers within the WebSphere MQ infrastructure.

These objects can be used to configure specific routes between the individual queue managers within the infrastructure. They can also be used to join a queue manager to a *queue manager cluster*, in which the channels between queue managers are automatically created when required.

Queue manager clusters reduce the amount of administration required when building or modifying a WebSphere MQ infrastructure. Queue managers can join and leave a queue manager cluster without additional configuration of the existing queue managers within the infrastructure.

Queue manager clusters also provide significant extra functionality over manually defined intercommunication between queue managers. This functionality can be used to increase the scalability and availability of the services provided by a system.

3.1.3 SupportPacs

Some additional functionality and documentation, which is not supplied in the base WebSphere MQ product, is provided within a WebSphere MQ SupportPac™.

The areas of functionality provided by SupportPacs are varied and include the following features:

- Reports about the performance of WebSphere MQ
- Documentation and guides regarding particular areas of function
- Sample applications that interact with WebSphere MQ
- Scripts that simplify the administration of WebSphere MQ
- Interfaces into WebSphere MQ from additional programming languages

Each SupportPac has a unique reference and an associated category. The category specifies the origin of that SupportPac and the level of support that is provided by IBM for that SupportPac.

For a list of all available SupportPacs and for more details about the SupportPac system and the categories of SupportPacs, see the following Web page:

http://www.ibm.com/software/integration/support/supportpacs

3.2 Simplification

WebSphere MQ provides simplified communication between applications running on different hardware platforms and operating systems, implemented using different programming languages, or running within different hardware and software environments.

This enables a business to choose the most appropriate infrastructure components for implementing or accessing services within their system.

New applications can interact with existing services, without the knowledge of the existing infrastructure components that implement those services. Existing services that do not provide an interface through the message queuing infrastructure can be adapted by developing *proxies* to adapt the existing interfaces to ones accessed through the WebSphere MQ message queuing infrastructure.

WebSphere MQ provides a range of application programming interfaces (APIs) to interact with the message queuing infrastructure. An API can be chosen from this range in line with the design methodologies of the programming language and environment on which an application is developed.

The asynchronous nature of messaging in WebSphere MQ can simplify application logic and promote structured handling of failures if they occur.

3.2.1 Applications accessing a WebSphere MQ infrastructure

By connecting to a single queue manager within a WebSphere MQ infrastructure, an application is able to communicate with applications that connect to other queue managers within that infrastructure. The queue manager to which each application connects can be hosted on a separate machine from the application.

This can occur regardless of the hardware and operating system combinations of the machines hosting the applications and the queue managers to which the individual applications connect.

3.2.2 Asynchronous intercommunication using WebSphere MQ

Two applications that need to intercommunicate, whether hosted on the same machine or separate machines, might be originally designed to do so directly and synchronously.

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

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

Placing a WebSphere MQ queue between the two applications enables this intercommunication to become asynchronous. One application places information for the partner on a queue within a WebSphere MQ message, and the partner application processes this information when it is available to do so. It can then send a reply to that message back to the originator if required.

If the two applications are on different machines, the exactly once delivery assurance provided by WebSphere MQ ensures that each message arrives, and that it arrives once only. This often removes the requirement for a reply to be sent back from the service. If a failure occurs in processing the message, the service can perform a consistent action, which might involve replying to the originator of the message, or informing an administrator, or other application, by sending a report.

Applications that perform processing on messages are considered as providing a service. That service can be any form of processing, such as updating information in a database or sending an e-mail to an administrator.

3.2.3 Generalizing destinations using WebSphere MQ

Destinations within the infrastructure are identified by the names of queues from which applications process messages. These names are unique within each queue manager, but queues of an identical name can exist on other queue managers within the infrastructure.

This name is used to identify a particular service, or a generalized definition, provided by the infrastructure.

3.2.4 Specific destinations using WebSphere MQ

An individual application can optionally have its own unique destination within the infrastructure, consisting of the name of a queue allocated to that application and the name of the queue manager to which the application connects.

This destination can be permanent, thus allowing messages to be sent to the application even while it is inactive. It can also be temporary, lasting only for the life of the application.

These destinations can be dynamically created, allowing an unspecified number of applications connected to the same queue manager to have their own unique destinations within the infrastructure.

These unique destinations allow replies to be routed to the producer of a request in a point-to-point model and for publications to be routed to subscribing applications in a publish/subscribe model.

3.2.5 Providing services within a WebSphere MQ infrastructure

An application providing a service simply waits for messages to arrive on a queue, which is identified within the infrastructure by its name.

The application then processes each message that arrives based on the message contents and the custom business logic required for the service. This might include updating or requesting information to or from a database, making decisions about the required next manual or automated actions, and performing those actions, including sending messages to other services hosted within the system.

Multiple applications providing a service can process messages that arrive on the same queue, or a running application can request exclusive access to all messages on a particular queue.

3.2.6 WebSphere MQ queues as an interface for accessing services

An application requires the following information to access a service, provided through a WebSphere MQ message queuing infrastructure:

- Addressing information for the service: In WebSphere MQ, this is the name of a queue in a point-to-point model or the name of a topic in a publish/subscribe model.

- How to connect to the infrastructure: In WebSphere MQ, this is the name and connection details for one queue manager within the infrastructure. This queue manager can be hosted on the same machine as the application or on a remote machine connected by a network.
- Details of the interface into the service: This includes whether the interface is send and forget, request/reply, or publish/subscribe. It also includes the structure of the data contained in any messages that flow between the applications accessing and providing the service.
- A mechanism to send and receive messages: This is an application programming interface (API) that provides facilities compatible with the infrastructure being used. WebSphere MQ provides a number of different APIs that can be used from different programming languages and platforms.

Other considerations, such as how to route messages to their destination, are dealt with by the WebSphere MQ infrastructure, transparent to the operation of the application.

This includes dealing with intercommunication failures at any intermediate nodes on route to the destination. It can also include routing around failed nodes in the network and workload balancing requests to the available resources.

Accessing services in this way is flexible and maintainable. Separate departments within a business, or businesses themselves, can maintain their own infrastructure and services, as well as providing external access into these services.

To do this, they provide the previously mentioned information to other departments within the business or to other businesses such as business partners, customers, and suppliers. Alternatively, they can develop custom applications to access the service, through the message queuing infrastructure, and make the interface to these applications available externally, for example, by providing an external interface into a service through a Web browser.

Changes can be made to the infrastructure, such as moving applications between machines, increasing capacity, performing maintenance, and restructuring or combining infrastructures, without affecting the operation or availability of the services provided through the interface.

3.2.7 Standardized application programming interfaces (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.

Examples of standardized APIs, which can be used to access services provided through a WebSphere MQ infrastructure, include:

- The Java™ Message Service (JMS)
- The IBM Message Service Client (XMS)

We discuss these in more detail in 4.2.3, “Standardized APIs available for WebSphere MQ” on page 52.

Using these standardized APIs to access services allows the logic within the application to be independent of the technology providing message queuing. This enables the technologies used within the infrastructure to change.

For example, the broker providing publish/subscribe capabilities to a WebSphere MQ infrastructure can be upgraded from the WebSphere MQ publish/subscribe broker to WebSphere Business Integration Message Broker or WebSphere Business Integration Event Broker.

The WebSphere Business Integration Message Broker and WebSphere Business Integration Event Broker are separate broker products that build on the core messaging capabilities of WebSphere MQ to provide additional functionality, exploiting the full potential of the publish/subscribe messaging model.

Wide adoption of these APIs can occur across multiple products. For example, the JMS API is an industry standard API for messaging within the Java 2 Platform, Enterprise Edition (J2EE™) specification.

The customization of a standardized API to a specific message queuing implementation is not usually performed by the application itself. Instead, this information is usually held in a *directory* of information, accessed locally or over a network. An application gains the information required to perform message queuing from this directory.

For WebSphere MQ, the information in this directory includes the connection details for a queue manager and the names of queues.

The information in this directory can change to reflect changes to the interface into a service, such connecting to a different queue manager. The application does not need to be altered after these infrastructure change are made.

3.2.8 WebSphere MQ and WebSphere Application Server

J2EE-compliant application servers, such as IBM WebSphere Application Server, provide a framework within which applications can be developed and hosted.

An application hosted within a J2EE application server can access a range of functionality through the APIs defined within the J2EE specification.

JMS is just one of the standardized APIs within the J2EE specification, defining an API for performing point-to-point and publish/subscribe messaging.

Other APIs are defined for a broad spectrum of functionality that applications might need to access. For example, an application might be accessed through interfaces available over the Internet through a Web browser or an application might query and update information in a database.

The technologies that provide J2EE functionality can be *embedded* within the J2EE application server product, or can be separate products that are configured within the J2EE application server as a *provider* of that functionality.

WebSphere Application Server Version 5 is supplied with an embedded provider of JMS functionality based on the WebSphere MQ product.

WebSphere Application Server Version 6 is supplied with an embedded provider for JMS functionality called *WebSphere Platform Messaging*. WebSphere Platform Messaging is a separate technology to WebSphere MQ and provides point-to-point and publish/subscribe message queuing functionality.

WebSphere MQ can be configured as a JMS provider for WebSphere Application Server V6.0. WebSphere Platform Messaging infrastructures can also be interconnected with WebSphere MQ infrastructures.

3.2.9 Web services as an interface for accessing services

Standardized APIs in themselves do not solve the problem of defining the interface into a service. The application must still know the format of the message to send to a service and the information to provide to the service in order for it to perform its function.

Web services can provide a solution to this. The principles of Web services define common standards used to describe and exchange information between applications requesting services and the services themselves.

Web services are described using a common language, called the *Web Services Description Language (WSDL)*. The WSDL for each Web service can describe how it is accessed, the information it requires, and what information it returns and can provide general information about the service, such as a description of the action it performs.

The WSDL describing a Web service can be generated from the code implementing the business logic of the service it provides. This is called a *bottom-up* approach.

Alternatively, previously defined WSDL can be used as the basis for writing the code implementing the service. This is called a *top-down* approach.

In either case, the WSDL can be shared between the application implementing the Web service and all the applications accessing the Web service. This provides a common specification for interactions between these applications.

The WSDL describing each Web service provided by a system can be held in a common *registry*. This enables applications, or developers, to query this registry for the WSDL describing the service they want to access, including how to interact with it.

These interactions are performed through a layer of functionality called *Simple Object Access Protocol (SOAP)*.This defines the format in which data is shared between the Web service and the application accessing it. The data can be validated against the WSDL description to ensure that only correctly specified information is exchanged.

SOAP defines how the data exchanged is specified by applications. However, it does not, in itself, provide a mechanism for communication between the applications over a network. For this, a *transport* layer is required to provide the communication between applications.

Using a WebSphere MQ message queuing infrastructure to provide this transport can combine the benefits of Web services for defining interactions, with the asynchronous nature, reliability, and exactly once delivery assurance of messaging with WebSphere MQ.

For more information about Web services, including a guide to using WebSphere MQ as a transport for Web services, see the following publications:

- *WebSphere MQ Solutions in a Microsoft .NET Environment*, SG24-7012
- *WebSphere MQ Transport for SOAP*, SC34-6651

3.2.10 Simplification of failure handling with WebSphere MQ

When using any of the interfaces provided into WebSphere MQ, an application can benefit from simplified failure handling in comparison to synchronous interaction with a service over a communications link.

After a message has been placed within the WebSphere MQ infrastructure, WebSphere MQ assures delivery of that message to the destination occurs, that it occurs once, and that it occurs once only.

The exactly once delivery assurance provided by WebSphere MQ removes the responsibility for handling failures in communications links from an application.

Regardless of the final destination for a message, an application first places the message in a queue hosted by the queue manager to which it is connected. This can be the final destination of the message, or an intermediate queue for transfer to another queue manager within the infrastructure.

The application itself only needs to know the name of a queue. The configuration of the WebSphere MQ infrastructure determines how the message reaches its destination and that any subsequent replies can be returned to the queue manager to which the application is connected.

Queue manager clusters can simplify this configuration by providing queue managers with automatically maintained knowledge of the other queue managers within the system and their availability. Using queue manager clusters, there can be multiple possible destinations for one message, representing multiple resources in the system that can process the message. This, too, is transparent to the application, which only needs to know the name of a queue.

3.3 Scalability and performance

WebSphere MQ provides an efficient and scalable message queuing implementation, building on the facilities provided by each operating system to scale performance with the power provided by modern, multiple processor, server hardware.

3.3.1 Scalability features of WebSphere MQ queue managers

Each queue manager performs a large number of tasks simultaneously to benefit from the multiple physical or virtual processing units provided by the hardware on which it is running. WebSphere MQ coordinates these multiple tasks to assure that data integrity is not compromised.

A large number, many thousands in some cases, of connections to a queue manager can be active at any one time. Multiple connections can process messages from the same queues hosted by that queue manager concurrently.

WebSphere MQ allows flexibility in the design of the applications that access a queue manager to allow the performance and capacity of the applications themselves to scale.

Applications that access a WebSphere MQ infrastructure can be developed and deployed within WebSphere Application Server application servers to benefit from the scalability features of the WebSphere Application Server product.

Applications can be designed to build directly on the threading and multitasking features provided by operating systems, performing multiple tasks simultaneously within separate threads of a single application instance, or running many instances of the same application connected to a single queue manager.

3.3.2 An architecture based on a single queue manager

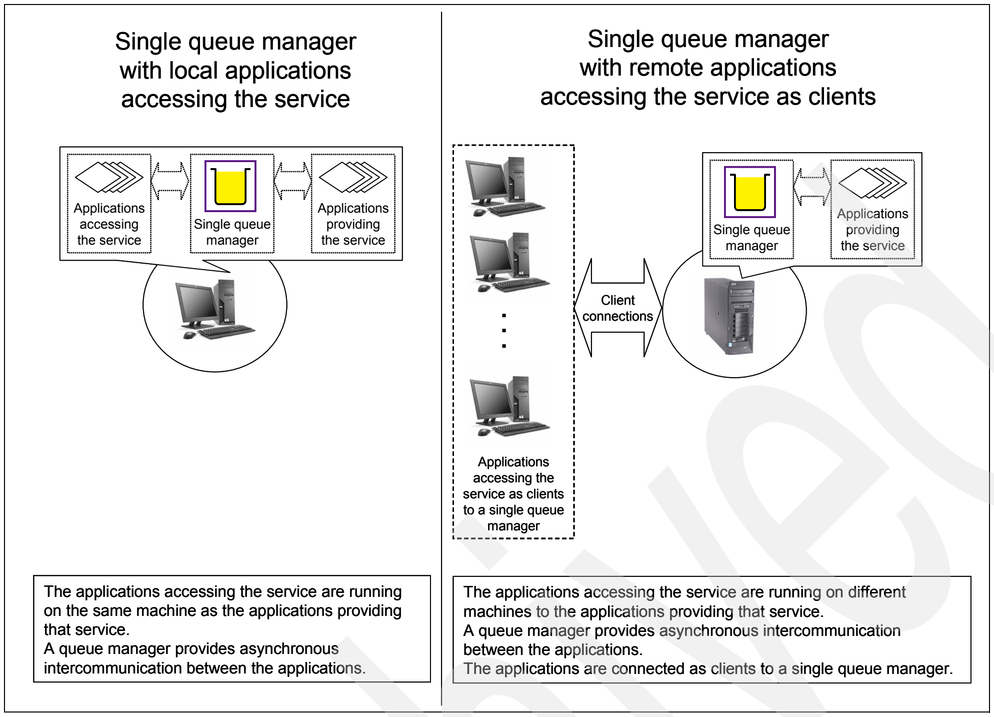
The asynchronous nature of messaging with WebSphere MQ helps enable applications to flexibly scale capacity, even when only a single machine is involved in providing that service.

Because queues provide a buffer between the application requesting a service and the application providing that service, the service can cope with varying loads in a flexible manner.

The application requesting the service can be hosted on the same machine as the application providing the service. A queue manager hosted on that same machine provides queues to allow efficient asynchronous intercommunication between those applications. This is shown in the left half of Figure 3-1 on page 37.

WebSphere MQ allows the applications connecting to a queue manager to be on remote machines, connecting as *clients* to that queue manager. Therefore, access to a service can be provided to multiple machines through a single queue manager. The number of clients is not limited by WebSphere MQ. However, this number is eventually limited by the capacity of that single machine.

This represents the simplest form of WebSphere MQ infrastructure: a single queue manager hosting a service required by the applications connecting to that queue manager. This is shown in the right half of Figure 3-1 on page 37.



Single queue manager with local applications accessing the service
Applications accessing the service
Single queue manager
Applications providing the service
The applications accessing the service are running on the same machine as the applications providing that service.
A queue manager provides asynchronous intercommunication between the applications.
Single queue manager with remote applications accessing the service as clients
Applications accessing the service as clients to a single queue manager
Client connections
Single queue manager
Applications providing the service
The applications accessing the service are running on different machines to the applications providing that service.
A queue manager provides asynchronous intercommunication between the applications.
The applications are connected as clients to a single queue manager.

Figure 3-1 Examples of architecture using a single WebSphere MQ queue manager

3.3.3 Hub and spoke WebSphere MQ architectures

Connecting to a WebSphere MQ queue manager as a client has limitations. A network connection is placed between the application and the queue manager, which has performance implications, especially over longer distances. This also requires that a network connection is available in order for the application to operate. The interaction between the application and the queue manager over that network connection is synchronous, although WebSphere MQ manages that connection for the application.

This single queue manager approach can be scaled, without alteration to the application, to include multiple queue managers.

Applications accessing a service can have a queue manager hosted on the same machine, providing a fast connection to the infrastructure, and gain asynchronous communication with the service hosted on another queue manager.

Alternatively applications accessing a service can connect as clients over a fast network to a queue manager, for example, all applications accessing a service in a branch office. Asynchronous communication occurs through this queue manager to the services hosted on another hub queue manager.

These applications themselves can be implemented to provide an external interface into the service, for example, by providing an interface over the Internet through a Web browser.

The queue managers through which the services provided by the hub are accessed are called the *spokes* of a hub and spoke WebSphere MQ architecture.

A machine, such as a mainframe or server hardware, can host the applications *providing* the service. This machine hosts a *hub* queue manager, holding the queue from which these applications process messages. This same machine can host other resources required by those applications, such as a database. Many instances of the applications providing the service can process requests from the same queue, depending on the implementation of the applications providing the service.

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

Figure 3-2 on page 39 shows an example hub and spoke architecture.

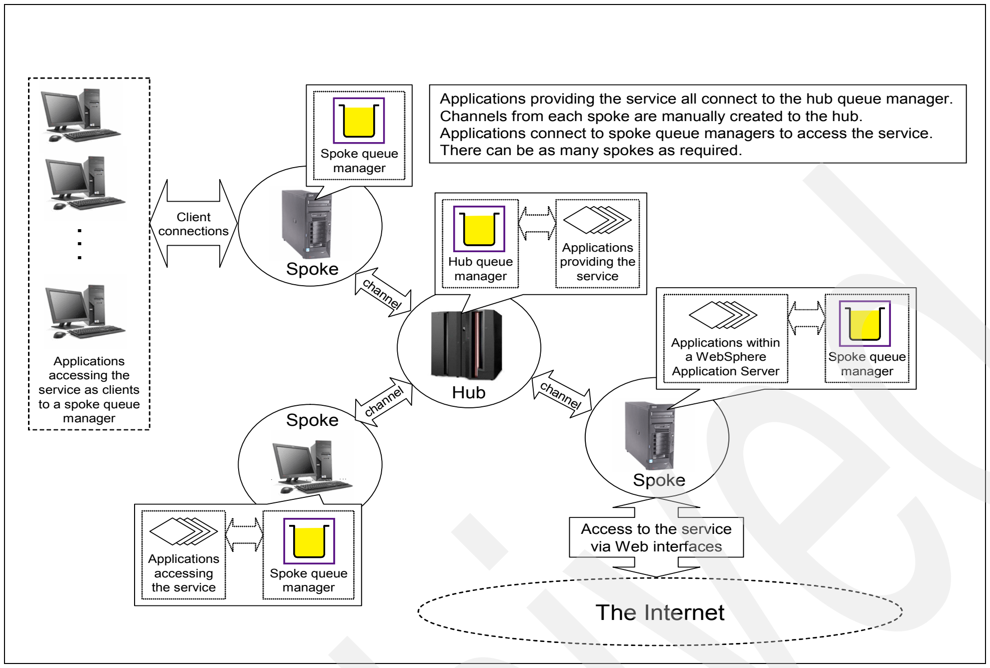

Applications providing the service all connect to the hub queue manager.
Channels from each spoke are manually created to the hub.
Applications connect to spoke queue managers to access the service.
There can be as many spokes as required.
Spoke queue manager
Client connections
Spoke
channel
Hub queue manager
Applications providing the service
Hub
Applications within a WebSphere Application Server
Spoke queue manager
Applications accessing the service as clients to a spoke queue manager
channel
channel
Spoke
Spoke
Applications accessing the service
Spoke queue manager
Access to the service via Web interfaces
The Internet

Figure 3-2 Example WebSphere MQ hub and spoke architecture

3.3.4 Flexibly scaling capacity using queue manager clusters

A more flexible approach is to join queue managers together in a *queue manager cluster*. Queue manager clusters allow 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 automatically *workload balances* these requests across all available queue managers that host an instance of that service.

This allows a *pool* of machines to exist within the queue manager cluster, each hosting a queue manager and the applications required to provide the service. This is especially useful in a *distributed* environment, where capacity is scaled to accommodate the current load through multiple servers, rather than one mainframe or high-capacity server.

Queue managers can dynamically join or leave the queue manager cluster to cope with varying loads placed on a particular service provided by a system. The configuration only needs to be performed on the queue manager joining or leaving the cluster, not the queue managers already within the cluster.

The queue managers to which applications connect when requesting services can also dynamically join and leave the cluster. For example, these might be gateways into the infrastructure from a Web-based interface that also has to cope with varying loads. Or, a queue manager might be hosted by each remote branch office of a business to access common services provided across the whole business.

Figure 3-3 shows an example of WebSphere MQ architecture base on a queue manager cluster.

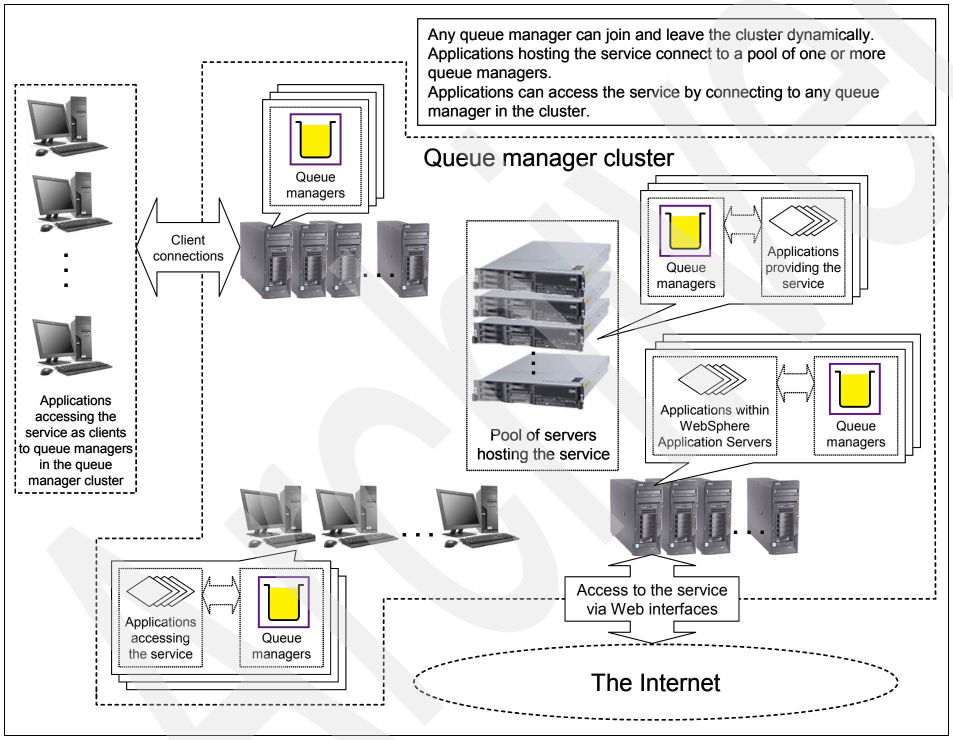

Any queue manager can join and leave the cluster dynamically.
Applications hosting the service connect to a pool of one or more queue managers.
Applications can access the service by connecting to any queue manager in the cluster.
Queue manager cluster
Queue managers
Client connections
Applications accessing the service as clients to queue managers in the queue manager cluster
Queue managers
Applications providing the service
Pool of servers hosting the service
Applications within WebSphere Application Servers
Queue managers
Applications accessing the service
Queue managers
Access to the service via Web interfaces
The Internet

Figure 3-3 Example WebSphere MQ architecture based on a queue manager cluster

3.4 Reliability and data integrity

WebSphere MQ queue managers, which together form a WebSphere MQ infrastructure, have built an excellent reputation for reliability through more than 10 years of development. Queue managers can remain running for long periods of time, and applications can remain connected to a queue manager for long periods of time.

The intercommunication performed between queue managers across channels is tolerant to network communication failures, and WebSphere MQ assures exactly once delivery of messages.

3.4.1 Persistent and nonpersistent messages

Some interactions between applications in a system relate only to query data, where the loss of the data might be inconvenient but does not affect the integrity of the information held within the system. In these cases, performance can be considered more important than data integrity.

For example, an application requesting the status of an order through a graphical interface can be accessed from many locations or across the Internet. The user accessing this information is only likely to wait for a finite period of time for this information to be available. If the information is not available, the request message or reply message for this information might no longer be required. The loss of a message in this circumstance is less significant than an inability to process that message efficiently.

However, messages containing business-critical data, such as receipt of payment for an order, need to be reliably maintained within that system. If a failure occurs on a machine within the infrastructure, or planned maintenance of some machines within the infrastructure makes the destination unavailable, messages must wait within the infrastructure until they can be delivered.

Messages that flow through a WebSphere MQ infrastructure are marked as one of the following messages to reflect the data contained:

- Nonpersistent messages:
  WebSphere MQ optimizes the actions performed on nonpersistent messages for performance. Nonpersistent messages can be lost if network communication between queue managers fails, a queue manager is restarted to perform maintenance, or an abrupt failure occurs that ends a queue manager uncleanly.

  Because the loss of some query data can be inconvenient, because it might need to be requested again, a queue manager can be configured to perform less optimization on certain nonpersistent messages. This includes transferring nonpersistent messages across channels in a failure-tolerant manner and maintaining nonpersistent messages on queues during a planned restart of the queue manager. However, exactly once delivery is not assured for nonpersistent messages regardless of the configuration of the queue manager. Always mark messages containing business-critical data as persistent.

- Persistent messages:
  Messages containing business-critical data are marked as persistent. WebSphere MQ assures exactly one delivery of persistent messages. This means that WebSphere MQ does not discard a persistent message through network failures, delivery failures, or planned restarts of the queue manager. Each queue manager keeps a failure tolerant *log*, sometimes referred to as a *journal*, of all actions performed on persistent messages. This protects against unplanned abrupt failures causing persistent messages to be lost. Incomplete actions on persistent messages are undone. This includes maintaining the integrity of the units of work described in 3.4.2, “Units of work” on page 42.

  Note: Always keep the log data held by queue managers on reliable storage. If data on storage is lost or corrupted, due to a hard disk failure, for example, WebSphere MQ might not be able to recover messages.

Persistent messages can allow the design of applications to be simplified by relying on the exactly once delivery assurance of WebSphere MQ. A message that is sent to perform an action is assured to arrive at its destination to be processed. The application that actually performs the action on the message does not need to be available at the time that the message requesting that action is sent.

3.4.2 Units of work

Many actions 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, should any messages be sent or received.

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

These actions are considered to be within a *unit of work*. Units of work performed by applications accessing a WebSphere MQ infrastructure can include sending and receiving messages and updates to databases. 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.

WebSphere MQ can also participate in units of work that are coordinated by other products. For example, actions against a WebSphere MQ infrastructure can be included in units of work that are coordinated by WebSphere Application Server.

3.5 Security

WebSphere MQ provides features to assure security of access, authentication of identity, and security and integrity of communication.

3.5.1 The Object Authority Manager (OAM)

All actions performed by an application connected to a queue manager are authenticated by that queue manager by a component called the Object Authority Manager (OAM).

The configuration of a queue manager is performed by defining WebSphere MQ objects within that queue manager. In order to perform any action, such as connecting to a queue manager, sending a message, or retrieving a message from a queue, an application must access WebSphere MQ objects.

Every time an application attempts any action against a WebSphere MQ object, the OAM ensures that the identity under which that application is connected to the queue manager has been set to allow the type of access it is requesting on the object it is performing an action against.

3.5.2 Secure Sockets Layer (SSL) and Transport Layer Security (TLS)

Applications which connect to WebSphere MQ do not need to be hosted on the same machine as the queue manager to which they connect to access the infrastructure. They can connect as a client to that queue manager over a network.

Extending access in this way is very flexible, but reduces the assurance that a queue manager has established the true identity of each application connecting to it.

Secure Sockets Layer (SSL) and Transport Layer Security (TLS) are industry-standardized technologies that provide assurance of identity.

SSL and TLS provide similar capabilities and build on similar principles for establishing identity. TLS is often considered the successor of SSL, because it provides some enhanced security features. WebSphere MQ V6.0 provides TLS capabilities, as well as the SSL capabilities provided in WebSphere MQ V5.3.

SSL or TLS can be used for all communication performed over a network within a WebSphere MQ infrastructure.

Using these technologies, WebSphere MQ can verify the identity of applications connecting to a queue manager and can also verify the identity of other queue managers within the infrastructure with which it exchanges messages.

3.5.3 Securing communication using SSL or TLS

In addition to allowing the identity of entities to be verified, and thus allowing authentication of that entity to be performed by the OAM of a queue manager, SSL and TLS also provide industry standardized security of communication.

Any communication over a network within a WebSphere MQ infrastructure for which identity has been verified using SSL or TLS can then be encrypted using a variety of different algorithms within the SSL and TLS standards. This assures the privacy of that communication.

SSL and TLS can also provide verification of the integrity of data flowing across a network connection. This protects against both malicious modification of the data and corruption of the data by the network communication link.

3.6 High availability

The reliability of the WebSphere MQ product makes it an excellent choice for providing a message queuing infrastructure for services that need to be highly available.

This should be combined with a quality assurance (QA) environment, which closely matches the production environment, for development and testing of applications and testing of changes to applications and the infrastructure before these changes are undertaken in production.

Investing in the development of such a QA environment allows testing of the services provided by a system to ensure that they match the external usage of that system as closely as possible. This minimizes the possibility that software or logic failures affect the availability of the services provided by the system.

For services where high availability is of critical importance, consideration should be made for the planned and unplanned outages that occur. This includes hardware failures and the maintenance and upgrading of applications and infrastructure components.

For more information about all of the topics discussed in this chapter, see the *Understanding high availability with WebSphere MQ* white paper available from the following Web page:

`http://www.ibm.com/developerworks/websphere/library/techarticles/0505_hiscock/0505_hiscock.html`

3.6.1 The role of queue manager clusters in high-service availability

Joining queue managers together within a queue manager cluster, as discussed in 3.3.4, "Flexibly scaling capacity using queue manager clusters" on page 39, allows multiple independent instances of a service to be implemented and deployed dynamically within a system.

The queue managers within a queue manager cluster automatically route messages to all available instances of the service. If a queue manager is marked as unavailable, for example, to apply maintenance to WebSphere MQ or the applications and other resources providing the service, new messages are no longer sent to the instance of the service hosted through that queue manager.

Equally, if a queue manager becomes unavailable unexpectedly, for example, due to a network failure or a hardware failure affecting that queue manager, requests are automatically routed to the other instances of the service that are still available. Queue manager clusters enable the infrastructure to automatically route around failures.

WebSphere MQ V6.0 introduces additional features for queue manager clusters, which allow more control over this automatic routing of requests. Queue managers hosted on higher-capacity servers can route more messages than those on lower-capacity servers. Queue managers can also be designated as backups for a service so that they are only routed messages if the primary queue managers are unavailable.

3.6.2 Queue sharing groups on WebSphere MQ for z/OS

IBM WebSphere MQ for z/OS builds on the facilities provided by joining z/OS systems together in a *sysplex*. Queue managers in the same sysplex can be members of a WebSphere MQ *queue sharing group*. These queue managers can all access messages on the same shared queues.

This can address the problem of message availability, where a queue manager becomes unavailable while messages are contained on its queues. Queue manager clusters automatically route new requests around that queue manager, but the messages on the queues of that queue manager cannot be accessed until it is made available again.

If the queue manager is a member of a queue sharing group, other queue managers within the queue sharing group that are still available can access the messages on the queue, preventing those messages from being unavailable.

3.6.3 High availability clusters

Another solution for the problem of providing message availability if hardware failures occur or during planned maintenance, such as applying updates to software components, is the use of *high availability (HA) clusters*.

These enable a primary and secondary hardware server to have all of the software components installed that are required to provide a service. *Failover* occurs between the primary and secondary server if requested manually or if any component providing that service fails on the primary server.

The reliable storage, which is used by all of the software components on the primary server to store data, is also failed over to the secondary server. This allows the same components on this secondary server to have access to the *exact* state of the data at the time of failure.

Note: HA clusters are not a feature of the WebSphere MQ product itself, but a wider concept that multiple products, including many operating system vendors, implement. Do not confuse HA clusters with queue manager clusters, which are a feature of the WebSphere MQ product.

IBM HACMP™ is one example of a product that provides HA clustering functionality for servers using the IBM AIX® 5L™ operating system.

In a WebSphere MQ context, this means that a WebSphere MQ installation exists on both servers, and both servers have a queue manager configured to store data on the reliable storage that is failed over between the servers.

These two queue managers cannot run concurrently, accessing the same data, but the HA clustering software switches the reliable storage between the two servers if a failure is detected.

It is significant that this functionality is provided separately to WebSphere MQ. This is because the resources that need to be failed over between the machines are not limited to WebSphere MQ resources. All software components required to implement a service, including the applications themselves and any databases, application servers, or other infrastructure software, must be available after the failover occurs. These software components must all have access to the exact data from the time of the failure.

In order for this to be transparent to the operation of the service, the identity of the server within any network must also be failed over.

By being supplied within a separate product, the HA clustering software is able to independently establish the availability of all of the software components under its control and perform failover of all of those components between the machines.

A HA clustering solution is a combination of HA clustering software, which can perform full coordination of the resources, combined with a reliable storage solution that can be switched between the primary and secondary servers. A complete installation of all of the software required, including the custom application software, is required on both servers to use the storage switched over to that machine when failover occurs.

WebSphere MQ does not limit support to specific HA clustering solutions. A business can choose a solution that matches their individual requirements, operating systems, and hardware choices.

3.6.4 Disaster recovery

Recovering from disastrous outages, which cause significant loss of data or multiple machines to become unavailable, requires careful consideration and planning. Such events are unpredictable by their nature.

Full snapshots of all data for a queue manager can be taken to provide a backup of that queue manager at a particular point.

Disaster recovery techniques cannot usually assure the most up-to-date information is available after a failure. The action of backing up the data is not generally performed synchronously as it is written due to the performance costs of doing so.

WebSphere MQ V6.0 provides some facilities to allow a remote copy of a queue manager to exist at a geographically remote location based on a snapshot of a queue manager. Actions on the data of the original queue manager, as logged by the original queue manager, can be transferred to the remote location in segments as each segment is finished with by the queue manager. This can be performed manually or using third-party products.

These actions can then be replayed by the remote copy of the queue manager to make it a more recent reflection of the original queue manager. This can allow the most recently backed up and transferred data to become more quickly available after a disastrous outage. However, the asynchronous nature of performing backups over long distances is unlikely to make the exact data from the time of the failure available.

3.7 Monitoring and accounting

WebSphere MQ V6.0 introduces new functionality to enable you to gather performance monitoring and accounting information.

This section provides a high-level overview of the features provided. Refer to *Monitoring WebSphere MQ*, SC34-6593, for details about these features, their purpose, and use.

3.7.1 Performance monitoring

WebSphere MQ V6.0 can provide real-time performance information about the flow of messages through queues hosted by a queue manager and the flow of messages across channels between queue managers.

You can also generate summary reports regular intervals, containing statistics about the usage of queues hosted by the queue manager or channels connecting queue managers.

3.7.2 Accounting

WebSphere MQ V6.0 can generate reports about the usage of a queue manager for each application that connects to a queue manager. These reports are generated as each application disconnects, or at regular intervals for longer running applications.

The reports are designed to provide enough information to identify the application or the entity that invoked that application. Enough information is contained about the actions performed by that application to analyze its individual performance or to determine charging based on those actions.

3.7.3 Trace-route messaging

WebSphere MQ V6.0 provides facilities to identify the route that messages take through a WebSphere MQ infrastructure or interconnected WebSphere MQ infrastructures.

4

Designing applications that access a WebSphere MQ infrastructure

This chapter describes how applications can be designed to build on the facilities provided by a WebSphere MQ infrastructure to provide and access the services of a system. It introduces the fundamental concepts of point-to-point and publish/subscribe messaging in WebSphere MQ and describes the available application programming interfaces (APIs).

We discuss the following topics:

- Cross-platform support
- Application programming interfaces (APIs)
- WebSphere MQ messages
- Interacting with a WebSphere MQ infrastructure
- Units of work and transactions
- Point-to-point messaging with WebSphere MQ
- Publish/subscribe messaging

4.1 Cross-platform support

The queue managers that form a WebSphere MQ infrastructure, and the applications that access that infrastructure, can be hosted on a wide variety of different hardware types and operating systems. The combination of a particular hardware type and a particular operating system is called a *platform*.

See the following Web page for information about the platforms supported by IBM for hosting queue managers:

`http://www.ibm.com/software/integration/websphere/mqplatforms/supported.html`

This Web page provides a statement of environment (SOE) for each platform on which WebSphere MQ is supported by IBM. The SOE provides details about the levels supported for each software component with which WebSphere MQ can interact and any maintenance required by WebSphere MQ for those software components. On some platforms, the supported version of WebSphere MQ might not be the latest version.

Additional platforms, which are not listed and not supported by IBM, might be supported by IBM Business Partners.

A queue manager does not need to be hosted on the same hardware server as the applications that connect to that queue manager. When a queue manager is hosted on a different hardware server, a WebSphere MQ client product is required for the application to connect to the WebSphere MQ queue manager. WebSphere MQ client products might be supported on additional platforms, by IBM and IBM Business Partners, than those supported for hosting queue managers.

4.2 Application programming interfaces (APIs)

In order for applications to connect to a queue manager and interact with the WebSphere MQ infrastructure of which that queue manager is a part, an application programming interface (API) is required.

4.2.1 The message queue interface (MQI)

The core API provided by WebSphere MQ is the message queue interface (MQI).

The MQI is a *procedural* API and as such is suitable for applications developed within procedural programming languages. A procedural API is one in which the context and data required by each function is passed to that function when it is invoked. The application itself must track all context and provide areas in which to store each item of data.

The MQI also defines all structures, constants, and basic data types required to interact with WebSphere MQ. Manipulation of these structures and types must be performed explicitly when developing applications using the MQI.

Applications developed using the following programming languages are likely to use the MQI directly:

- C
- COBOL

The other APIs described in this section build on the flexibility of the MQI to provide interfaces that enable a business to exploit the benefits of modern programming languages and approaches.

4.2.2 APIs based on the WebSphere MQ object model

Object-orientated programming languages allow actions, states, and data to be associated with the logical objects on which those actions are performed. Using this approach when developing applications can allow the structure of applications to more logically match their function.

WebSphere MQ provides object-oriented APIs for a number of object-oriented programming languages. Although programming using these APIs differs among individual programming languages, all of these interfaces conform to a single design, called the WebSphere MQ *object model*.

These interfaces wrap the functionality and data structures provided by the MQI in classes from which objects can be instantiated. Each class provides methods that can be performed on the objects instantiated from that class. This allows an application to focus more on the business logic being performed by the application and less on manipulating and tracking context, maintaining data structures, or allocating and de-allocating memory.

WebSphere MQ provides the following object-oriented APIs:

- WebSphere MQ C++:
  WebSphere MQ C++ provides an API for the C++ language, which conforms to the WebSphere MQ object model. Developing applications to access WebSphere MQ using the C++ language allows use of an object-oriented interface, while still allowing direct access to operating system and hardware functionality using C functions if required. For more information about the C++ API, refer to *WebSphere MQ Using C++*,SC34-6592.

- WebSphere MQ base Java API:
  The WebSphere MQ base Java API provides an API for the base Java language, which conforms to the WebSphere MQ object model. Developing applications to access WebSphere MQ using the Java language allows applications to gain from the portability between platforms provided by the Java platform. The functionality provided within the Java platform can simplify other aspects of an application. For more information about the Java API, refer to *WebSphere MQ Using Java*, SC34-6591.

  Note: See 4.2.3, "Standardized APIs available for WebSphere MQ" on page 52, for an alternative API for the Java language.

- WebSphere MQ classes for .NET:
  The WebSphere MQ classes for .NET provide an API for the .NET environment, which conforms to the WebSphere MQ object model. Microsoft Windows applications can be developed using the WebSphere MQ classes for .NET in multiple programming languages. For more information about the .NET environment and the WebSphere MQ classes for .NET, refer to *WebSphere MQ Using .Net*, GC34-6605.

  Note: See 4.2.3, "Standardized APIs available for WebSphere MQ" on page 52, for an alternative API for the .NET environment.

- WebSphere MQ component object model (COM) interface:
  The WebSphere MQ COM interface contains a set of Microsoft ActiveX® components called the WebSphere MQ Automation Classes for ActiveX (MQAX). The components of MQAX conform to the normal conventions expected of an ActiveX component, as well as the WebSphere MQ object model. For more information about COM and MQAX, refer to *WebSphere MQ for Windows V6.0, Using the Component Object Model Interface*, SC34-6594.

4.2.3 Standardized APIs available for WebSphere MQ

As discussed in 3.2.7, "Standardized application programming interfaces (APIs)" on page 31, using standardized APIs to access WebSphere MQ can provide added flexibility.

Note: Standardized APIs can simplify the logic of applications through the use of simplified and standardized messaging concepts and hiding implementation details. The layer provided between the standardized API and the WebSphere MQ infrastructure uses relevant functionality provided by WebSphere MQ to implement the functionality provided by the standardized API.

Access to the full range of functionality provided by a WebSphere MQ infrastructure might require the use of a WebSphere MQ proprietary API.

Examples of standardized APIs that can be used to access a WebSphere MQ message queuing infrastructure include:

- Java Message Service (JMS):
  JMS is part of the Java 2, Platform Enterprise Edition (J2EE) standard.

  JMS encapsulates both the point-to-point messaging model and the publish/subscribe messaging model. As such, applications developed in Java, using the JMS interface, can be independent of the broker that provides the publish/subscribe capabilities to WebSphere MQ. This provides the flexibility to upgrade the WebSphere MQ publish/subscribe broker without the costly redevelopment of applications.

  Applications using the JMS API are not required to build and issue WebSphere MQ publish/subscribe commands when accessing the publish/subscribe messaging capabilities.

  JMS is an industry standard for messaging within the J2EE standard. J2EE standardizes interfaces to a broad range of common functionality, of which messaging using JMS is one.

  The WebSphere MQ product is supplied with all of the tools required to allow an application, which is developed to perform messaging with the JMS API, to access a WebSphere MQ infrastructure. WebSphere MQ becomes a *provider* of JMS to that application.

  Note: The messages placed in a WebSphere MQ infrastructure using the JMS API by default have some additional meta information attached to them. This information enables a remote application to receive those messages through the JMS API. This information can be disabled for interoperability with applications using other APIs to access a WebSphere MQ infrastructure.

  Applications accessing a WebSphere MQ infrastructure through a JMS interface can be deployed in WebSphere Application Server. This enables the application to benefit from the full functionality provided by the J2EE standard, as well as the deployment and scalability functionality provided by WebSphere Application Server.

The embedded JMS supplied with WebSphere Application Server V5 is based on the WebSphere MQ V5.3 product. WebSphere Application Server V6 provides a new JMS messaging provider called WebSphere Platform Messaging. WebSphere MQ can be configured as the JMS provider for WebSphere Application Server V6. A WebSphere Platform Messaging infrastructure can be interconnected with a WebSphere MQ infrastructure.

When implementing a new application in Java, performing point-to-point or publish/subscribe messaging, consider the use JMS to interface with WebSphere MQ.

Java, JMS, J2EE, and WebSphere Application Server are significant topics in their own right.

- IBM Message Service Client (XMS):
  XMS is a messaging API for the C and C++ programming languages and the .NET environment. XMS provides a very similar approach to messaging as is used by the JMS API, and thus benefits from the industry standardization that the JMS API has achieved.

  XMS encapsulates both the point-to-point messaging model and the publish/subscribe messaging model. Applications developed in C, C++, or within the .NET environment using the XMS interface can be independent of the broker that provides the publish/subscribe capabilities to WebSphere MQ. This provides the flexibility to upgrade the WebSphere MQ publish/subscribe broker without the costly redevelopment of applications.

  Applications using the XMS API are not required to build and issue WebSphere MQ publish/subscribe commands when accessing the publish/subscribe messaging capabilities.

  The messages placed into a WebSphere MQ infrastructure by applications using an XMS API can be accessed by applications using the JMS API to interact with the WebSphere MQ infrastructure.

  Currently, the tools required to facilitate development of an application using the XMS interface and to allow that application to perform messaging across a WebSphere MQ message queuing infrastructure are provided in SupportPac IA94. See the following Web page for more information:

  `http://www.ibm.com/support/docview.wss?rs=171&uid=swg24007092&loc=en_US&cs=utf-8&lang=en`

  Note: SupportPac IA94 is currently Category 2. Therefore, support is not available for this SupportPac through IBM product service channels. See the Web page for more detailed information.

When implementing a new application in C, C++, or within the .NET environment, performing point-to-point or publish/subscribe messaging, consider using XMS to interface with WebSphere MQ.

4.2.4 Custom adapters

If WebSphere MQ does not provide an interface into a particular programming language or software infrastructure component that is required for an application, a *custom adapter* can be developed.

A custom adapter provides an interface between one of the previously described APIs, such as MQI, and the programming language or software infrastructure component within which the application has been developed.

A custom adapter is a general example of a proxy, taking an existing interface into a service and extending it to benefit from the capabilities of the WebSphere MQ message queuing infrastructure.

An example of such a custom adapter is SupportPac MA89 Perl language support for MQSeries®. This custom adapter provides an interface between WebSphere MQ and the Perl programming language. See the following Web page for more information about SupportPac MA89:

http://www.ibm.com/support/docview.wss?rs=171&uid=swg24000208&loc=en_US&cs=utf-8&lang=en

Note: SupportPac MA89 is Category 4. It is provided by a third-party supplier, not from IBM. See the Web page for more detailed information.

4.3 WebSphere MQ messages

The messages sent through a WebSphere MQ infrastructure can contain data of many forms, as per the individual requirements of the applications that access that infrastructure.

In order for processing of an individual message to be performed, information is required about the message, which is not part of the message data itself, for example, whether the message requires a reply to be generated by an application processing that message.

4.3.1 The message descriptor

Every message that passes through a WebSphere MQ infrastructure has a *message descriptor* associated with that message.

The message descriptor contains the *meta information* about the message. This information describes the message, but does not constitute part of the data that the message contains.

Some of this information is specified by the application that sends the message, and some is generated and updated automatically by WebSphere MQ as the message passes through the message queuing infrastructure. All of the information is available to the application that receives the message.

Examples of this meta information, which we discuss throughout the rest of this chapter, include:

- A uniquely generated, or custom, identity for the message
- The type of message, which determines whether a reply is required
- Details of where to send any replies to the message
- Whether this message has a limited life, called an *expiry interval*
- Information about how the message was produced, when, and by whom
- Information about the representation of data in the message body

4.3.2 Data conversion

The individual machines that host the queue managers of a WebSphere MQ infrastructure can represent characters and numbers in different ways. Different byte values can be used to represent a particular character and to represent a particular number.

Each individual queue manager, running on a particular machine, has an understanding of the way character and numeric data is stored on that machine. This means that it can perform *data conversion* to convert data in other representations to a representation understood by the applications running on that machine.

WebSphere MQ can be configured to perform this in two ways:

- As the message passes through the WebSphere MQ infrastructure, data conversion can be performed each time the message reaches a new queue manager over a channel. This can be inefficient, because the message might pass through multiple queue managers before reaching its destination.
- When an application receives a message, it can specify that it wants data conversion to be performed. This data conversion is performed in the local representation used by the machine on which that application is running, or the application can request a specific representation. This is the recommended method.

4.3.3 Message formats

In order for a queue manager to be able to perform this data conversion for messages, it must be told about the data contained within that message. This is specified by the application that sends the message by setting a *message format* in the message descriptor. If this is not specified, WebSphere MQ treats the message body as binary, and no data conversion can be performed.

The message format specifies *one* type of data.

Often messages contain only character data, because character data can be used to represent both numbers and characters. For example, the Extensible Markup Language (XML) can be used by applications sending and receiving messages to provide a consistent structure for messages. XML messages represent all the data they contain using characters, so WebSphere MQ can perform character data conversion on the whole of those messages.

WebSphere MQ also allows messages to be structured in flexible ways, containing combinations of binary, character, and numeric data, as well as containing extra meta information that is understood by WebSphere MQ to describe the structure of the data.

4.3.4 Chaining portions of a message together

Flexibility is gained by *chaining together* portions of the message in a way understood by the data conversion algorithms of queue managers. Each portion has a length, and the length of all of the portions totals the full length of the message.

The format of each portion and information about its original representation are specified by the previous portion of the message. The *first* portion of the message is identified by fields in the message descriptor. For messages containing only character or binary data, the message descriptor specifies the details of the whole message.

The following points summarize the types of portions:

- One that can be considered as all one type of data by WebSphere MQ: Data that can be considered by WebSphere MQ to be of all one type is categorized as follows:
  - Binary data: WebSphere MQ does not perform data conversion.
  - Character data: WebSphere MQ can perform character data conversion.

- A WebSphere MQ structure:
  These structures can contain multiple different types of data, and some of these structures contain extra meta information about the message. These structures can be automatically added to, and removed from, messages by WebSphere MQ as they pass through the WebSphere MQ infrastructure.

  Where another portion of the message is allowed after the structure, the structure contains enough information to specify the details of that portion. This is a subset of the information contained in the message descriptor.

  Some examples of such structures include:

  – The transmission queue header and dead letter queue header:
    These are added to a message automatically by a queue manager, in circumstances described later in this book. It is not usual for an application to add these structures to a message themselves, but it is useful to know that these structures might be added to a message when looking at a message on a queue within a queue manager.
  – The rules and formatting header, and rules and formatting header 2:
    These can be used by applications to describe the contents of a message containing multiple different types of data, such as combinations of character and numeric data. Chaining together multiple portions of a message using these structures can be useful in allowing data conversion to occur differently for the different portions of the message.

    Note: Messages placed into a WebSphere MQ infrastructure using standardized APIs, such as the Java Message Service (JMS), often have rules and formatting headers added to them automatically to contain the extra meta information required by the standardized API.

- A WebSphere MQ command:
  Some messages are sent to WebSphere MQ components to perform actions. Examples include the WebSphere MQ publish/subscribe broker and the WebSphere MQ command server, discussed later in this book. WebSphere MQ defines the structure of the messages that contain these commands.

  Note: Using standardized APIs, such as JMS, to perform publish/subscribe messaging can remove the requirement to generate and issue commands to the WebSphere MQ publish/subscribe broker.

4.4 Interacting with a WebSphere MQ infrastructure

Using any of the APIs discussed, applications connect to a queue manager in order to access a WebSphere MQ infrastructure.

Applications can connect to a queue manager hosted on the same machine on which they are running. This is the most efficient connection method to a queue manager and is called a *bindings* connection to the queue manager.

Applications can also connect to a queue manager hosted on a different machine using a *client* connection over a network. There is a performance effect of operating a connection as a client, and the remote queue manager must be available in order for the connection to succeed. However, a WebSphere MQ server installation is not required on the machine where the client is hosted.

4.4.1 WebSphere MQ client products

A smaller amount of WebSphere MQ software is required on a machine that hosts applications connecting to a WebSphere MQ infrastructure as clients to a queue manager.

Note: Some WebSphere MQ client products are available as SupportPacs, and some are supplied on the distribution media for a WebSphere MQ server installation.

WebSphere MQ server licenses are currently not required for machines that have *only* a WebSphere MQ client installation.

4.4.2 Core facilities provided to a WebSphere MQ application

The following list provides a summary of the core facilities provided to an application that connects to a WebSphere MQ infrastructure through an individual queue manager:

- An application can retrieve messages from queues hosted on that queue manager in order to process those messages and perform processing, for example, to host a service with a request/reply, or send and forget, interface on a queue.
- It can gain an identity within the infrastructure based on a queue on the queue manager to which it is connected. This identity is provided to other applications in order for them to send messages to this application. In point-to-point messaging with WebSphere MQ, this is called a *reply-to queue*, or *response queue*. This identity can exist beyond the life of the application if required, and multiple applications can share the same queue by retrieving only messages designated for that application, using a *correlation identifier*. We discuss the options for gaining this identity in 4.6.14, “Reply-to queue considerations” on page 75.

- An application can send messages to queues anywhere within the infrastructure, as long as the queue manager to which it is connected has been configured to have knowledge of that destination.

  Destination queues can be hosted on the same queue manager for asynchronous intercommunication with other applications connected to that queue manager.

  The infrastructure can be configured to route messages based only on the name of the queue. This is recommended when sending a message to a service so that the infrastructure can be reconfigured without affecting the interface into that service.

  An application can specify a unique destination queue within the infrastructure based on the queue name and queue manager name. This is used when generating replies, or reports, in response to a request for a service from an application.

  In either case, the queue manager to which the application is connected must have knowledge of how to route messages to that destination queue. An individual decision is made by each queue manager on route to the final destination using a process called *queue name resolution*, so a queue manager only requires knowledge of the next step on route to the destination.

  The knowledge of routes to queues and queue managers, held by each queue manager, can be configured using WebSphere MQ *queue objects* defined on that queue manager. Queue manager clusters allow a queue manager to have automatic knowledge of other routes to queues and queue managers within the infrastructure, as well as performing workload balancing across multiple queues shared within the cluster with the same name.

  For a further discussion of queue name resolution and the technical configuration of queue managers that control queue name resolution, see 6.2.1, “Queue name resolution” on page 134.

- An application can access the publish/subscribe functionality provided by the infrastructure to publish messages and subscribe to topics.

  When using WebSphere MQ publish/subscribe directly, commands are sent to a request/reply interface provided by a WebSphere MQ publish/subscribe broker in the infrastructure. This allows an application to register as a subscriber on a topic, or publish messages on a topic, and issue other commands against the broker.

  Subscriptions are received onto a queue hosted on the queue manager to which the application is connected in a similar way that replies are received from a request/reply interface.

  For more information about WebSphere MQ publish/subscribe and extending the publish/subscribe capabilities of a WebSphere MQ infrastructure, see 4.7, “Publish/subscribe messaging” on page 78.

Note: Using a standardized API, as described in 4.2.3, “Standardized APIs available for WebSphere MQ” on page 52, can simplify the interface for the WebSphere MQ publish/subscribe.

4.5 Units of work and transactions

Units of work allow multiple actions performed by an application to be grouped together so that any individual action within that unit of work can only complete successfully if all actions within that unit of work complete successfully.

The basic construct that allows actions within a unit of work to complete or fail as a group is a *transaction*. The terms transaction and unit of work are often used interchangeably.

WebSphere MQ provides transactions that allow multiple operations on messages to complete, or fail, within a unit of work. Two basic actions can be performed on messages in WebSphere MQ:

- A message can be placed on a queue. This occurs as the first action performed by an application sending a message through the message queuing infrastructure and each time a message is placed on a transmission, or final destination, queue by a message channel.

  This action is called *putting a message* to a queue, or a *put*.
- A message can be received from a queue by an application, or by a message channel transferring a message from a transmission queue over a communication link.

  This action is called *getting a message* from a queue, or a *get*.

4.5.1 Local units of work

By default, a unit of work can only contain the WebSphere MQ put and get actions. The unit of work is begun automatically by WebSphere MQ, and the transaction is created automatically by WebSphere MQ. The unit of work is said to be *local* to WebSphere MQ.

The transaction that controls a local unit of work is *coordinated* by WebSphere MQ, because WebSphere MQ owns the transaction that controls the unit of work.

4.5.2 Syncpoint

The automatic creation of local units of work occurs the first time a put or get is performed by an application, provided that WebSphere MQ is told to perform that action *under syncpoint*.

Under syncpoint means that a put or get should be contained within the current unit of work. All subsequent actions performed under syncpoint, which can be on a variety of queues, continue to be part of that unit of work until the unit of work ends.

4.5.3 Commit and back out

A local unit of work can end in three circumstances:

- The application chooses to *commit* the unit of work. This means that all actions within that unit of work should be completed. The application is informed by WebSphere MQ whether the action of committing a unit of work was successful or not.
- The application chooses to *back out* the unit of work. This means that all actions within that unit of work are undone so that each message returns to the queue from which a get was performed in that unit of work, or a message never arrives on the queue to which a put was performed within that unit of work.
- The application disconnects from WebSphere MQ. If this disconnection is performed in a controlled way by the application, a local unit of work is committed by WebSphere MQ on behalf of the application and the application is informed if this commit fails. However, if the application terminates without disconnecting from WebSphere MQ in a controlled way, on detection of this termination, WebSphere MQ backs out any in progress unit of work for that application.

4.5.4 Uncommitted messages

When messages are put to a queue within a unit of work (under syncpoint), these messages are not available for other applications, or the application that put the message, to get from that queue until the unit of work has been committed. If a unit of work is backed out, any messages put to queues within that unit of work are never made available for other applications to get.

When messages are got from a queue within a unit of work (under syncpoint), these messages are made unavailable to other applications as soon as the get action completes. Therefore, two applications cannot get the same message from a queue. However, messages got from a queue within a unit of work are not actually removed from that queue until the unit of work is committed. If a unit of work is backed out, any messages got from queues within that unit of work are made available for all applications to get from the queue again. Therefore, if the same application retries a get within a subsequent unit of work, it might get the same message.

Messages that have been put to a queue within a unit of work that has not yet been committed or backed out, or messages which that been got from a queue within a unit of work that has not yet been committed or backed out, are called *uncommitted messages*.

Only one unit of work can be in progress at any time for a single connection from an application to a queue manager. However, an application can maintain multiple connections to a queue manager by using different *threads* within that application. A thread is a facility provided by an operating system or Java Virtual Machine (JVM™) to allow multiple actions to be performed concurrently by an application.

4.5.5 Global units of work

One of the most powerful features of a unit of work is that it can contain actions performed against multiple resources, not only WebSphere MQ. The most common example of such a resource is a database.

One common set of actions that can be performed by a service within a system is:

1. Get a request message from a queue.
2. Perform database queries and database updates based on that request.
3. Send messages to other services within the system to perform additional processing of the request.
4. Send a reply message back to the requesting application.

If the application providing the service fails abruptly, for example, after step 2 in the previous example, the consistency of the system might be dependent on all actions within the unit of work being backed out. If these actions are contained in a global unit of work, a failure at any stage can be resolved by backing out the unit of work. The original request message then returns to the queue for processing as though it had not been retrieved from the queue at all.

Units of work that include WebSphere MQ actions and actions on other resources are called *global* units of work.

4.5.6 Coordination of global units of work

The transaction that controls a global unit of work is coordinated by a *transaction manager*, which must be able to communicate with all participants within that unit of work. The participants within a global unit of work are called *resource managers*. In the previous example, WebSphere MQ and the database are the resource managers participating in the global unit of work.

WebSphere MQ can act as the transaction manager coordinating global units of work that include database products as resource managers.

When an application needs to *begin* a global unit of work, it must inform the transaction manager for that unit of work. This includes informing WebSphere MQ to begin a global unit of work. This is because the transaction manager does not know when the first action within a global unit of work has been performed by the application. For example, the first action in a global unit of work coordinated by WebSphere MQ might be an update to a database.

WebSphere MQ can also act as a resource manager within global units of work which are coordinated by external transaction managers. Examples of such transaction managers are IBM CICS®, TXSeries® and WebSphere Application Server.

For information about configuring WebSphere MQ as a transaction manager, or resource manager, within global units of work, refer to Chapter 9, “Configuring WebSphere MQ,” in *WebSphere MQ System Administration Guide*, SC34-6584.

Note: IBM only supports the operation of WebSphere MQ as a transaction manager with certain resource managers. Equally, IBM only supports the operation of WebSphere MQ as a resource manager with certain transaction managers. See the following Web page for details:

http://www.ibm.com/software/integration/websphere/mqplatforms/supported.html

4.5.7 Two-phase commit

Coordination must be performed between all of the participants within a global unit of work to allow any resource manager to report a failure at any time. The most important form of coordination is called *two-phase commit*.

The two-phase commit process consists of the following two steps:

1. Prepare:
   All resource managers participating in a global unit of work are queried by the transaction manager to ensure that they are able to commit the unit of work. After a resource manager has successfully completed the prepare phase, it

can no longer signify that a unit of work must be backed out. From that point, only the transaction manager can back out the unit of work. Therefore, a resource manager must retain knowledge of a prepared unit of work indefinitely until the transaction manager tells it to commit or back out that unit of work.

2. Commit:
   If the prepare phase completed successfully for all resource managers, the transaction manager informs all resource managers to commit the unit of work.

4.5.8 The XA specification

The Open Group publishes the XA Specification. This defines the interactions that can occur between a transaction manager and resource managers. WebSphere MQ conforms to this specification. See the following Web page for more information or to download the XA Specification:

`http://www.opengroup.org/bookstore/catalog/c193.htm`

4.5.9 The extended transactional client

WebSphere MQ allows applications that connect to a queue manager to be remote to that queue manager by connecting as a client to that queue manager over a network.

However, during the coordination of global units of work, all resource managers being controlled by a transaction manager must reside on the same machine as the transaction manager itself.

Because of this, an application using a standard client product to connect to a queue manager cannot include WebSphere MQ actions within global units of work.

WebSphere MQ provides a product called the extended transactional client. This product enables applications that connect to a remote queue manager to include WebSphere MQ actions within global units of work.

Note: This product is not supplied under the same terms as the WebSphere MQ client installation. See the following Web page for more information:

`http://www.ibm.com/software/integration/wmq/transclient.html`

In these circumstances, WebSphere MQ cannot act as the transaction manager for the global unit of work, because only a queue manager can coordinate global units of work within WebSphere MQ. However, WebSphere MQ can act as a resource manager within that global unit of work. For example, the application can participate in global units of work coordinated by WebSphere Application Server.

4.5.10 Failure handling and tolerance

By performing WebSphere MQ actions within a unit of work (under syncpoint) and then committing those actions only when related processing has been completed successfully by an application, that application can become tolerant to failures.

Note: Some actions performed by an application cannot be included within a global unit of work, for example, updating a file on the file system or sending an e-mail to an administrator. An application developer needs to consider that WebSphere MQ is not able to back out these actions if an abrupt termination of the application occurs.

If an application providing a service experiences problems while performing processing on a message, it can back out the current unit of work and return that message to a queue. That message then becomes available for processing again by that same application or another application waiting for messages on the same queue.

If an application needs to put and get a number of messages as part of a particular piece of processing, containing those separate actions within a unit of work means that each individual action only completes if the whole unit of work is committed successfully. If the application experiences a problem at any point during its processing, it can inform WebSphere MQ to back out the current unit of work.

If an application experiences an abrupt failure while performing processing on a message, and WebSphere MQ is coordinating a unit of work for that application, WebSphere MQ automatically backs out the unit of work when it detects the failure of the application, returning the message to the queue.

Note: WebSphere MQ can only detect the failure of a whole process, not threads within that process. This means that if a single thread terminates unexpectedly, an in progress unit of work for that thread is not resolved by WebSphere MQ until the process ends.

An example of such a process is a Java Virtual Machine (JVM) within a J2EE application server. If the application server detects a failure in an application it hosts and terminates that application, which is running as a Java thread within the virtual machine, WebSphere MQ does not detect termination of that thread until the whole JVM ends. However, J2EE application servers provide transaction management support, which might be able to account for these circumstances. WebSphere MQ can be configured as a resource manager for units of work coordinated by some J2EE application servers, such as WebSphere Application Server, as described in 4.5.5, "Global units of work" on page 63.

4.6 Point-to-point messaging with WebSphere MQ

The core point-to-point messaging capabilities, including the exactly once delivery assurance provided by WebSphere MQ when using persistent messages, allow reliable performance of point-to-point messaging to occur between applications that access a WebSphere MQ infrastructure.

This section expands on the facilities provided for send and forget and request/reply point-to-point messaging by a WebSphere MQ infrastructure. This includes a design-level view of how services are provided and accessed by applications.

4.6.1 Retrieving messages from queues

Applications retrieve messages from the queues hosted by the queue manager to which they are connected.

When retrieving messages, an application can specify that it has exclusive access to all messages that arrive on a certain queue, or multiple applications might wait for messages to arrive on the same queue.

When retrieving messages from a queue, applications can *browse* the messages on a queue without removing them and without making them unavailable for applications.

Alternatively, they can destructively *get* messages from the queue, making them unavailable to other applications. WebSphere MQ ensures that no two applications can successfully get the same message.

Applications can choose to wait for messages to become available on the queue for a specified period of time, or indefinitely, to browse or get available messages as they arrive.

Applications can qualify that they only want to retrieve messages with certain identity information, or *match options*. This is usually a correlation identifier, which can be used by applications to link a message to a single application on a queue shared by multiple applications.

4.6.2 Hosting services on queues

An application can provide a queue as an interface into the service it provides by retrieving messages from that queue when they arrive and processing them individually.

Any number of applications connected to the message queuing infrastructure can submit requests to a service hosted on a queue without knowing the availability of the service when sending the message. These requests are buffered in the queue in the order in which they are received onto that queue and can be processed in turn by the service. This is called first-in first-out (FIFO).

A priority can be associated with each message, and WebSphere MQ retrieves messages with a higher priority when getting a message before messages with a lower priority. This facility can be used to provide different qualities of service to different applications requesting a service.

Multiple instances of an application providing a service might wait for messages on the same queue, because for some services, it is more efficient for a number of requests to be processed in parallel. These applications can be hosted on the same machine as the queue manager, or connect to the queue manager from remote machines as clients. Adding more instances of applications providing a service can allow the capacity and performance of a service to be increased.

By using only a queue name, and not a specific queue manager name, when sending messages to a service, the configuration of the infrastructure can be changed without affecting applications requesting a service. For example, the infrastructure can be changed to use the facilities of queue manager clusters and provide multiple instances of the same service on queues of the same name on different queue managers, thus on different machines.

The queue managers to which applications accessing the service are connected then *workload balance* their requests to all available instances of the service.

After completing the processing of a message, the service can generate replies on the request of an application accessing that service, or generate a report, depending on the outcome of the processing. We discuss this in 4.6.15, “Processing of messages by a service” on page 77.

4.6.3 Backout queues and backout counts

If the processing of a message fails, consistent action should be taken. One option is to move the message to a different queue for special attention, possibly adding extra information to the message based on the nature of the failure.

WebSphere MQ allows queues from which messages are retrieved to identify a backout queue. An application can check the name of this queue and use this as the destination of messages for which processing fails.

Some failures might not be permanent, and retrying the processing might be appropriate. If an application gets a message within a unit of work and then backs out that unit of work, or fails while processing, WebSphere MQ increases a *backout count* within the message descriptor of that message.

Applications can check this backout count when retrieving messages from the queue in order to determine whether that message should be processed or sent to the backout queue.

A backout threshold can also be specified on a queue to determine how many times processing of a particular message should be attempted.

Note: WebSphere MQ does not move messages to a backout queue automatically.

4.6.4 Event-driven services

An application providing a service can efficiently wait for messages to arrive on a queue indefinitely using WebSphere MQ, without placing undue loads on the infrastructure.

This allows services to be driven by events that occur within the system, or from external interactions with the system by users. These services can react to external actions quickly and effectively; a service can process events as they occur, rather than periodically checking whether events have occurred.

It might not be desirable for an application providing a service to be inactive for long periods of time while waiting for messages to arrive, for example, due to the resources used by that application while idle.

WebSphere MQ provides a *triggering* mechanism to allow these applications to be started automatically when messages become available on a queue. The application can then process all messages on the queue before exiting. Alternatively, an application can process all currently available messages and then wait for messages to arrive for a short period before exiting.

WebSphere MQ also allows triggering to occur when a certain number of messages have arrived on a queue, or for every message that arrives on a queue. This allows for batch processing of messages and simple applications that are designed to process a single message each time they run.

4.6.5 Send and forget messaging

Each message sent by an application only needs to be transferred to a queue hosted by the queue manager to which it is connected before the application can continue processing. This enables the application to begin processing quickly after producing each message, especially if it is using a bindings connection to the queue manager, rather than connecting as a client over a network.

Queue name resolution occurs based on the knowledge of the WebSphere MQ infrastructure held by the queue manager to which the application is connected to determine whether the message can be sent. If this is successful, the message is delivered to a queue. If the destination queue is determined to be local, it is delivered directly. Otherwise, it is placed on an intermediate queue, called a *transmission queue*, to be transferred to another queue manager in the infrastructure.

When placing a message on the transmission queue, the queue manager adds enough information to allow queue name resolution to occur again when the message reaches the next queue manager. That queue manager can host the queue itself, or perform the same steps to pass the message to the next queue manager on route to the destination.

Note: We provide the details of how to implement a WebSphere MQ infrastructure, how the knowledge of a queue manager is configured using WebSphere MQ objects, how this is affected by queue manager clusters, and how message transmission occurs over channels in the subsequent chapters of this book.

This asynchronous messaging has the benefit of allowing the application to continue performing business logic, while WebSphere MQ performs the communication required to deliver each message to the destination queue.

If the application connects to the service directly, for example, by using a direct communications protocol, the operation is synchronous: The application needs to wait for the service to be available and ready to accept the information, connect to it, send the information, and then confirm the information has arrived before it continues processing.

When an application sends a message, it can choose the persistence of that message. If it does not do so, the queue manager sets an appropriate default based on the configuration of the WebSphere MQ queue objects used during queue name resolution.

A message that contains a request for an action from an application providing a service, but does not require a reply, is marked as a *datagram* in WebSphere MQ. Applications can use services by sending datagrams to queues that provide interfaces into those services.

4.6.6 Distribution lists

An application can send the same message to multiple destinations using a single WebSphere MQ operation with distribution lists. If multiple destinations specified in a distribution list are reached from a queue manager by transferring the message over the same channel to an intermediate queue manager or the destination queue manager, the message is only sent once.

Note: This is functionality is not available in WebSphere MQ for z/OS. Refer to *WebSphere MQ Application Programming Guide*, SC34-6595, for more information.

4.6.7 Segmentation of messages

The maximum length of an individual message allowed in a WebSphere MQ infrastructure is 100 MB. However, by default, queues do not accept messages larger than 4 MB.

Messages that are larger than 100 MB, or larger than 4 MB where a queue that a message passes through on route to its destination is not configured to accept messages of a larger size, can be broken into smaller segments.

Segmentation when sending messages and reassembly of the whole message when receiving messages can be performed manually by an application or automatically by a queue manager.

Note: This is functionality is not available in WebSphere MQ for z/OS. Refer to *WebSphere MQ Application Programming Guide*, SC34-6595, for more information.

4.6.8 Logical grouping of messages

WebSphere MQ cannot, under certain circumstances, assure the ordering in which messages arrive on a queue to be processed, or which messages are retrieved from a queue by which applications. Messages are usually delivered in the order in which they are sent, but this can be affected by a failure in transmission of a message over a channel, multiple applications getting messages from the same queue, and units of work.

An application can mark messages as being logically grouped together. The messages within a group are numbered to allow applications to deliver these messages in the order in which they were sent within the group. A queue manager can automatically deliver messages within a group in the order they were sent. The individual messages within a group can be segmented if required, and WebSphere MQ does not require any relationship between the contents of the messages.

4.6.9 Reports

Some messages are sent to signify that an event has occurred, for example, in response to an unexpected event while processing a datagram message. These messages are marked as a *report* in WebSphere MQ, and the message descriptor for these messages contains a *feedback* field to identify the reason for the report.

If a service receives a datagram, but experiences a failure while processing that request, there are a number of possible actions that can be performed by the service at this stage. One of these is to generate a report message and send it back to the originator of the datagram, or send it to another queue for special processing.

An application can explicitly request that a service generates a report message only in the event of a failure, or only in the event of a success. WebSphere MQ passes this request to the service within a field of the message descriptor, and that service can be implemented to return a report in those circumstances.

4.6.10 Confirmation of arrival and confirmation of delivery reports

WebSphere MQ queue managers can also automatically generate reports in the following circumstances:

- Confirm on arrival (COA):
  A message arrives on its target queue on the target queue manager.
- Confirm on delivery (COD):
  A message is retrieved from its target queue by an application.

These are delivered to the destination specified by the application sending the message. This application does so using the same mechanism as used for request/reply messaging.

4.6.11 Synchronous request/reply messaging

The actions of sending a request message and waiting for a reply message in WebSphere MQ are separate asynchronous actions. If required by an application, these two actions can be joined together into a synchronous operation.

The action of sending a request is the same as the action of sending a datagram, but the application must provide some additional information to allow it to receive the reply.

The application provides a *reply-to queue*, on which it waits for replies in response to its requests. This reply-to queue can be unique to the application, or shared between multiple applications. We discuss this in 4.6.14, "Reply-to queue considerations" on page 75. The name of this queue is specified in the message descriptor of the request.

A reply-to queue manager name can also be specified in the message descriptor. However, this is filled in automatically by WebSphere MQ as the name of the queue manager to which an application is connected.

A message that contains a request for an action from a service and requires a reply to be returned to the requesting application is marked as a *request* in WebSphere MQ.

A message that contains a reply from a service in response to a request message from an application is marked as a *reply* in WebSphere MQ.

4.6.12 Partially synchronous request/reply messaging

The separation of generating a request and waiting for a reply into two separate asynchronous actions can add flexibility to the requesting application.

The application might not check for a reply immediately. Instead, it might perform other processing that is not dependent on the reply and then check for a reply message at a later time. This can be used to reduce the time during which an application is idle while waiting for a service to process a request. It can also allow processing of the reply to be deferred until it is requested by a user, or required for further processing by the application.

Some requests performed synchronously by an application can query data that is already stored reliably within the system. For these requests, an application might not want to wait indefinitely for a response from the service; it might *time out* after a specified interval. The asynchronous nature of messaging with WebSphere MQ makes it possible for an application to time out and discard a request before the reply has been received.

Note: During the time between sending a request message to a service and receiving a reply message, an application cannot determine the state of that request. After performing a timeout, the actual state of a request must not affect any later processing, because it might still complete successfully or have already completed.

In order to perform a timeout, an application can specify that it will wait for messages to arrive on the reply-to queue for a particular *wait interval*. After this time, WebSphere MQ returns to the application that no messages were available on the reply-to queue, and the application can continue processing.

If required, an application can wait multiple times for messages to arrive on the reply-to queue, performing processing between each attempt. An application can also repeatedly check for messages arriving on the queue, specifying to WebSphere MQ that is does not want to wait at all if no messages are available in order to *poll* WebSphere MQ for a reply.

4.6.13 Message expiry

If an application implements a timeout, it should specify an *expiry time* in the message descriptor of the request message. This expiry time is specified in tenths of a second by the requesting application and is reduced by WebSphere MQ to reflect the time it spends within the infrastructure. This includes time spent on the destination queue before being processed by the service and time spent on all intermediate queues. After the expiry time reaches zero, the message can no longer be received by applications and is eligible to be discarded by the infrastructure.

It is usual for a service to duplicate the current expiry time of the request message in the message descriptor of the reply message. However, a service can be implemented to reset the expiry time to a preset value, or to never set an expiry on reply messages.

An application can request to receive an *expiry report*, indicating when an expired message has been discarded by the infrastructure.

Note: WebSphere MQ does not discard messages from a queue immediately when the expiry time reaches zero. Expired messages continue to contribute to a queue's depth until an application attempts to retrieve those messages. An expiry report is not generated until this occurs.

In WebSphere MQ Version 6.0, a periodic check can be made of all queues hosted by a queue manager to cause expired messages to be removed automatically. This is enabled by default.

4.6.14 Reply-to queue considerations

The reply-to queue, provided by an application that generates a request, is hosted on the queue manager to which the requesting application is connected.

In a large system, many applications connected to the same queue manager can be requesting a service. WebSphere MQ scales efficiently in these circumstances and provides some design choices for the requesting applications when providing a reply-to queue in a request message:

- Use a single reply-to queue for all requests for a service:
  Each message in WebSphere MQ has a *message identifier* in its message descriptor. WebSphere MQ can generate this message identifier to be unique throughout the infrastructure.

  The message descriptor also contains a *correlation identifier*, which can be used by applications to correlate replies with requests.

  When sending a reply message in response to a request, a service copies the message descriptor from the request into the correlation identifier of the reply. This allows the reply message to have a unique message identifier, while allowing correlation between the reply message and the original request. An application requesting a service can then wait for a message with the same correlation identifier as the message identifier for the request it produced.

  This approach can simplify administration and reduce loads on a queue manager, because only one reply-to queue is required, and this queue is manually defined. This approach can be used for persistent messages.

  If an application terminates without receiving a reply to a request it generated, and the reply does not have an expiry time, the message needs to be removed or processed manually.

This can cause the queue to grow, but if these messages contain business-critical data, some action might need to be performed on messages that enter this state. For example, the requesting application can keep a record of its requests in order to recover after a failure.

Note: WebSphere MQ is optimized for applications that wait for messages with a particular correlation identifier. We do not recommend developing applications that wait for a message with a particular message identifier. This is not as efficient if a queue grows to contain a large number of messages.

- Use a temporary dynamic queue:
  WebSphere MQ allows queues to be created dynamically by applications to provide a unique identity for the application within the infrastructure. A *temporary dynamic queue* only exists as long as a particular application accesses it. Temporary dynamic queues are automatically removed from the queue manager by WebSphere MQ when an application specifies that it has finished processing related to that queue, or the application that made the request terminates.

  This approach is only suitable for query data contained in nonpersistent messages. Temporary dynamic queues cannot contain persistent messages for business-critical data because they are automatically removed from the queue manager by WebSphere MQ when an application terminates, even if this is due to a failure and the queue contains messages.

  Using this approach, an application can wait for any message to arrive on the temporary dynamic reply-to queue, rather than requiring a message with a particular correlation identifier. This provides strong isolation between individual applications accessing a service so that an application is less likely to affect the replies sent to other applications through logic errors.

  A small quantity of system resource is required for each temporary dynamic queue that is created or removed.
- Use a permanent dynamic queue:
  A *permanent dynamic queue* is created in the same circumstances as a temporary dynamic queue and can be used in the same way. However, it is not removed automatically from the queue manager by WebSphere MQ, including when the requesting application terminates. Instead, the requesting application must manually remove the queue from the queue manager after it has completed processing.

  This approach provides the same benefits as the use of temporary dynamic queues, but is suitable for use where the reply from a service contains business-critical data.

Similar to the use of a single reply-to queue, if an application terminates without receiving the reply for a request it generated, the reply message and the permanent dynamic queue remain until action is taken to process the message and remove the queue.

4.6.15 Processing of messages by a service

The following steps illustrate an outline of actions of a service in order to provide consistent processing of messages to generate replies and reports in response to datagrams and request messages:

1. If global units of work are being used, begin the global unit of work.
2. Retrieve a message from a queue. It might be necessary to consider message segmentation, or the order of logically grouped messages, when doing this. If so, inform WebSphere MQ that the whole message must be retrieved or that the messages within a group must be retrieved in order. Also, consider whether data conversion is required; if so, request that it is performed by the queue manager. Specify that the action is performed under syncpoint if using a unit of work.
3. Check the type of the message, the format of the message, and the requested report options in the message descriptor. If they do not match the functionality provided by the service, perform a consistent action. This might be to check whether a backout queue has been specified and put it to that queue.
4. If units of work and backout queues are being used, the backout count can be checked. If this is above the backout threshold for the queue, put the message to the backout queue.
5. Perform business logic on this message in order to provide the requested service. This can include interactions with other products, such as databases, and sending messages to other services provided by the system.
6. Decide a result based on the outcome of the processing to determine whether it completed successfully.
7. If a reply or a report is required based on the type of message received and the outcome of the processing, generate a reply. The following steps provide an overview of the normal steps involved:
    a. Create the data for the reply message.
    b. Create a message descriptor for the reply/report message, or use the original message descriptor.

c. Ensure that all of the fields in the message descriptor relating to the content of the message, such as the message format and details of the way data is represented, are set appropriately. Setting these to the defaults causes WebSphere MQ to choose values appropriate to the local representation of data.

d. Set the message type to either reply or report as appropriate.

e. Copy the message identifier from the original message descriptor into the correlation identifier of the reply/report message descriptor.

f. Clear the message identifier in the reply/report message descriptor. This causes WebSphere MQ to generate a new message identifier for the reply.

g. Choose whether to use the expiry time of the original message descriptor, or reset this to a particular value.

h. If a report message is being generated, specify an appropriate feedback code in the message descriptor of the report.

i. Ensure that the persistence and priority of the reply/report is the same as the original message.

j. Send the reply/report message using the reply-to queue manager name and reply-to queue name from the original message. Specify that the action is performed under syncpoint if using a unit of work.

Note: These actions can be contained in a unit of work. We recommend this for persistent messages containing business-critical data.

8. Commit any unit of work.

4.7 Publish/subscribe messaging

Providing publish/subscribe messaging capabilities requires a publish/subscribe broker.

The publish/subscribe broker tracks subscriptions to individual topics and provides the facilities for a publisher to publish messages on a given topic.

4.7.1 WebSphere MQ publish/subscribe broker

WebSphere MQ is supplied with an integrated broker, which builds on the core capabilities of WebSphere MQ to provide the publish/subscribe capability.

Note: The publish/subscribe broker was integrated into the WebSphere MQ product in Fix Pack 8 for WebSphere MQ Version 5.3 and is included in WebSphere MQ Version 6.0. Previously, it was supplied in SupportPac MA0C.

The WebSphere MQ publish/subscribe broker uses the message queuing infrastructure of WebSphere MQ to accept and process commands, track subscriptions, hold current state information, and as a delivery mechanism to send publications to subscribers. By doing this, the broker inherits the exactly once delivery assurance of WebSphere MQ.

Note: We provide an outline of the functionality provided by the WebSphere MQ publish/subscribe broker.

See the following guides for more information about the publish/subscribe capabilities provided by WebSphere MQ:

- *WebSphere MQ Publish/Subscribe User's Guide*, SC34-6606
- *WebSphere Business Integration Pub/Sub Solutions*, SG24-6088
- *MQSeries Publish/Subscribe Applications*, SG24-6282

Each queue manager can host a maximum of one broker, and this broker shares the same name within the infrastructure as the queue manager that hosts it. Brokers can be connected together in a *broker network* to allow applications connected to one queue manager to receive publications made to a broker hosted on another queue manager.

4.7.2 Interacting with the WebSphere MQ publish/subscribe broker

The WebSphere MQ publish/subscribe broker provides a set of commands that can be sent to the broker using a request/reply interface. These commands perform functions such as registering an application as a subscriber and publishing a message on a particular topic

The request/reply interface of the broker for each queue manager is hosted on a queue called the *broker control queue*.

Standardized APIs, such as the Java Message Service (JMS), can simplify the interface into the broker by removing the requirement to generate and issue these commands explicitly.

4.7.3 Streams

The broker can separate the flow of information about topics into *streams*. A publication made for a topic on one stream is not published to subscribers for that topic registered on other streams. The subscriptions, and other required information, are held on a separate WebSphere MQ queue for each stream.

4.7.4 Registration

Each subscriber must *register* with the broker before it can receive publications. This registration process requires that the subscriber provides a queue onto which suitable publications can be placed as they are received by the broker.

This queue can be individual to the subscriber or shared between multiple subscribers. If a subscription queue is shared between multiple subscribers, a correlation identifier must be provided by the subscriber during registration. This correlation identifier is used to identify the ownership of messages on the queue. Similar considerations exist for choosing a subscription queue, as discussed in 4.6.14, “Reply-to queue considerations” on page 75.

A publisher is not required to perform any registration with the broker before publishing messages. However, a publisher can choose to register with the broker so that some default behavior is configured regarding the publications that it makes and to notify the broker of which topics it publishes. A publication made by a publisher can contain any data that a WebSphere MQ message can contain within the message body, as described in 4.1, “Cross-platform support” on page 50.

4.7.5 Topics

Topics are identified within the broker by character strings, called *topic strings*. When each publication is made by a publisher, a topic string is specified.

When each subscriber registers with the broker, it also specifies a topic string. After the subscriber is registered, the topic string is matched against the topic of each publication. If the two strings match, the publication is placed on the queue specified by the subscriber during registration. The strings do not need to match exactly; wildcard characters can be used in the topic string supplied by the subscriber during registration in order to subscribe to a range of topics.

Registrations do not end when the subscribing application ends. This means that publications build up on a subscriber’s queue while it is not active.

4.7.6 Publications

WebSphere MQ provides two forms of publications:

- Non-retained publications:
  By default, the WebSphere MQ broker discards a publication after it has been delivered to the queues of all matching registered subscribers.
- Retained publications:
  A publisher can specify that an individual publication, or all publications that it publishes, are retained by the broker. This means that each time a publication is processed by the broker, it is delivered to the queues of all matching registered subscribers, and a copy is retained by the broker. The broker only keeps the most recent publication on a particular topic.

One particular use for retained publications is for state information. By publishing state information as a retained publication, a subscriber can request the current state of information about a particular topic when it begins processing. This means that the subscriber does not need to wait for a publication on a topic in order to determine the current state of the information.

4.7.7 Extending the WebSphere MQ publish/subscribe capabilities

The WebSphere MQ publish/subscribe broker provides the core functionality required to use the publish/subscribe messaging model. However, the power of decoupling the provider of information from the producer of that information can be further exploited to simplify the design and implementation of business services.

Fully exploiting the potential provided by the publish/subscribe messaging model can provide a messaging infrastructure where diverse applications exchange information in disparate forms and the flow of this information is controlled, routed, and transformed using business rules, rather than proprietary logic.

IBM WebSphere Business Integration Message Broker and WebSphere Business Integration Event Broker are separate broker products that build on the core messaging capabilities of WebSphere MQ to fully exploit the potential of the publish/subscribe messaging model.

Tip: Using standardized interfaces to access publish/subscribe services can provide the flexibility to upgrade from the core WebSphere MQ publish/subscribe messaging capabilities to a solution based on the WebSphere Business Integration Message Broker or WebSphere Business Integration Event Broker product without costly redevelopment to applications.

5

Understanding and configuring queue managers

In this chapter, we describe the administration interfaces used to create and administer WebSphere MQ queue managers and also provide an overview of the operation of queue managers and the facilities they provide for data integrity.

We discuss the following topics:

- Installation information
- WebSphere MQ administration interfaces
- The queue manager

5.1 Installation information

The *WebSphere MQ Quick Beginnings* guides provide information for performing the installation of WebSphere MQ on Microsoft Windows, UNIX, and IBM @server® iSeries platforms. The guides also provide the steps to validate your WebSphere MQ installation. Refer to the *WebSphere MQ V6.0 Quick Beginnings* guide relevant to your platform:

- *WebSphere MQ for Windows V6.0 Quick Beginnings*, GC34-6476
- *WebSphere MQ for Linux V6.0 Quick Beginnings*, GC34-6480
- *WebSphere MQ for AIX V6.0 Quick Beginnings*, GC34-6478
- *WebSphere MQ for Solaris V6.0 Quick Beginnings*, GC34-6477
- *WebSphere MQ for HP-UX V6.0 Quick Beginnings*, GC34-6479
- *WebSphere MQ for iSeries V6.0 Quick Beginnings*, GC34-6481

Refer to *WebSphere MQ for z/OS V6.0 Concepts and Planning Guide*, GC34-6582, for installation instructions for z/OS and an introduction to concepts specific to using WebSphere MQ on the z/OS platform.

5.1.1 Review available WebSphere MQ maintenance

We recommend that you apply the latest maintenance to the WebSphere MQ product after installation. Refer to 12.2.1, "The WebSphere MQ support Web site" on page 326 for more information.

Note: The special information section of the *readme* file supplied with the latest maintenance delivery to WebSphere MQ might contain information relevant to your platform.

5.1.2 Statement of environment

Details of the supported versions of operating systems, compilers, and other software components that interact with the WebSphere MQ product, including required maintenance updates, are available in a statement of environment (SOE) for each platform supported by WebSphere MQ.

For these SOEs, refer to the following Web page:

http://www.ibm.com/software/integration/websphere/mqplatforms/supported.html

5.2 WebSphere MQ administration interfaces

The administration of WebSphere MQ can be performed through a variety of interfaces. This section provides an introduction to these interfaces.

5.2.1 WebSphere MQ Explorer

The graphical user interface (GUI) provided by WebSphere MQ for the administration of queue managers and the objects they contain, as well as the WebSphere MQ installation hosted on the same machine as the GUI, is the WebSphere MQ Explorer.

The WebSphere MQ Explorer is new in WebSphere MQ V6.0. Previous versions of WebSphere MQ for Windows are supplied with a Microsoft Management Console (MMC)-based GUI. This consists of a set of MMC snap-ins, called the WebSphere MQ Explorer snap-in and the WebSphere MQ Services snap-in.

The WebSphere MQ Explorer provides a GUI environment, which is similar in appearance to the MMC snap-ins of previous releases of WebSphere MQ, for the administration of queue managers and the WebSphere MQ installation. However, the functionality provided by the WebSphere MQ Explorer has expanded on the functionality provided in previous releases. This functionality continues to grow due to the benefits of building the WebSphere MQ Explorer on the *Eclipse* technology, as described in “WebSphere MQ Explorer and the Eclipse project” on page 92.

Starting the WebSphere MQ Explorer

At the time of writing this book, the WebSphere MQ Explorer was run for installations of the following WebSphere MQ products:

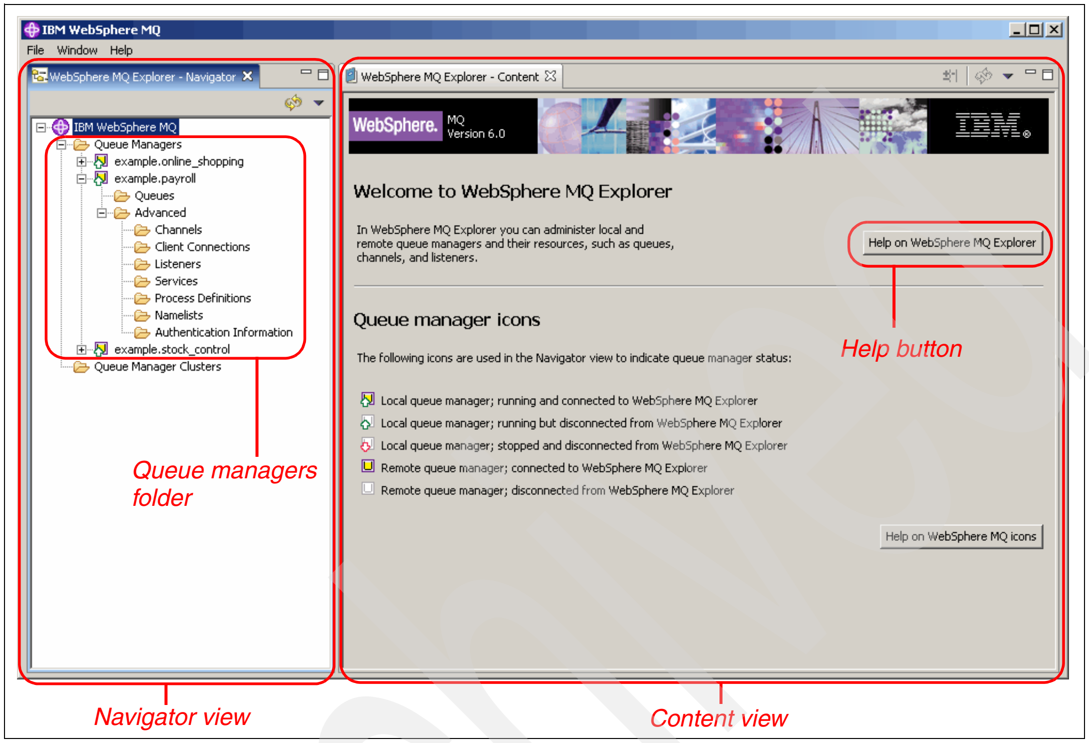
- WebSphere MQ for Windows
- WebSphere MQ for Linux® (x86)

After installing WebSphere MQ, including all the prerequisites, you can start the WebSphere MQ Explorer by using one of the following methods:

- Using the `strmqcfg` WebSphere MQ control command.
- Clicking the **WebSphere MQ Explorer** icon. This method is only available on WebSphere MQ for Windows. The icon is located in the IBM WebSphere MQ program group, from the Windows Start menu.

Working with WebSphere MQ and local queue managers

Figure 5-1 on page 86 illustrates the WebSphere MQ Explorer with three queue managers defined on the same machine as the WebSphere MQ Explorer. These are called example.payroll, example.stock_control, and example.online_shopping. The figure shows the highlighted sections of the layout that we discuss in this section.



Welcome to WebSphere MQ Explorer

In WebSphere MQ Explorer you can administer local and remote queue managers and their resources, such as queues, channels, and listeners.

Queue manager icons

The following icons are used in the Navigator view to indicate queue manager status:

- Local queue manager; running and connected to WebSphere MQ Explorer
- Local queue manager; running but disconnected from WebSphere MQ Explorer
- Local queue manager; stopped and disconnected from WebSphere MQ Explorer
- Remote queue manager; connected to WebSphere MQ Explorer
- Remote queue manager; disconnected from WebSphere MQ Explorer

Figure 5-1 WebSphere MQ Explorer layout

The layout of the WebSphere MQ Explorer is separated into two panes:

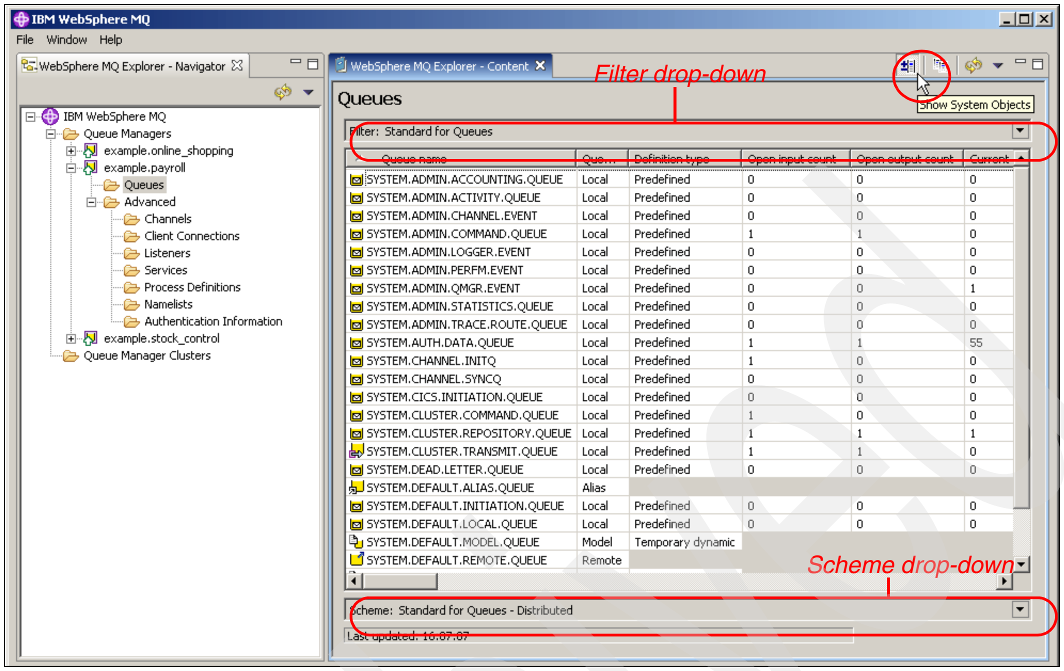
- The navigator view:
  This contains a tree view of the WebSphere MQ resources that can be administered from the WebSphere MQ Explorer. The resources shown in the navigator view are separated into folders under the IBM WebSphere MQ root icon. Click the **+** symbol next to an item in the tree to expand that item and see the items it contains, and click the **-** symbol to collapse that item. Highlight an item to view the content page for that item in the content view.
- The content view:
  Selecting an item in the navigator view displays a corresponding content pane in the content view. This page shows a table of all the objects that correspond to that item or information describing the item. The content pane might also provide actions.

Note: Press F1 at any time to get a help summary of the current content pane or the navigator view. Select **Help** → **Help Contents** for the full help system.

The Queue Managers folder is used to perform the administration of the queue managers hosted on the system and any queue managers that have been connected remotely to the WebSphere MQ Explorer. For each queue manager to which a connection exists, an item for that queue manager is listed under the Queue Managers folder. This item also contains a set of folders that are selected to access and configure the objects in the queue manager.

Some of the folders for a queue manager are located under an Advanced folder. The Advanced folder can be eliminated so that these folders are shown directly under the Queue Managers folder. To do this, click the **Advanced** folder and follow the instructions shown in the content view.

Clicking a subfolder for a queue manager opens a content page containing a table of those types of objects defined on that queue manager. The columns of this table show the attributes of each object in the table with a different icon for each type of object. If an attribute does not apply to a particular item in the table, that cell is gray. The system objects can be hidden, but are shown in Figure 5-2 on page 88.



Queues

Filter: Standard for Queues

Queue name	Que...	Definition type	Open input count	Open output count	Current
SYSTEM.ADMIN.ACCOUNTING.QUEUE	Local	Predefined	0	0	0
SYSTEM.ADMIN.ACTIVITY.QUEUE	Local	Predefined	0	0	0
SYSTEM.ADMIN.CHANNEL.EVENT	Local	Predefined	0	0	0
SYSTEM.ADMIN.COMMAND.QUEUE	Local	Predefined	1	1	0
SYSTEM.ADMIN.LOGGER.EVENT	Local	Predefined	0	0	0
SYSTEM.ADMIN.PERFM.EVENT	Local	Predefined	0	0	0
SYSTEM.ADMIN.QMGR.EVENT	Local	Predefined	0	0	1
SYSTEM.ADMIN.STATISTICS.QUEUE	Local	Predefined	0	0	0
SYSTEM.ADMIN.TRACE.ROUTE.QUEUE	Local	Predefined	0	0	0
SYSTEM.AUTH.DATA.QUEUE	Local	Predefined	1	1	55
SYSTEM.CHANNEL.INITQ	Local	Predefined	1	0	0
SYSTEM.CHANNEL.SYNCQ	Local	Predefined	0	0	0
SYSTEM.CICS.INITIATION.QUEUE	Local	Predefined	0	0	0
SYSTEM.CLUSTER.COMMAND.QUEUE	Local	Predefined	1	0	0
SYSTEM.CLUSTER.REPOSITORY.QUEUE	Local	Predefined	1	1	1
SYSTEM.CLUSTER.TRANSMIT.QUEUE	Local	Predefined	1	1	0
SYSTEM.DEAD.LETTER.QUEUE	Local	Predefined	0	0	0
SYSTEM.DEFAULT.ALIAS.QUEUE	Alias				
SYSTEM.DEFAULT.INITIATION.QUEUE	Local	Predefined	0	0	0
SYSTEM.DEFAULT.LOCAL.QUEUE	Local	Predefined	0	0	0
SYSTEM.DEFAULT.MODEL.QUEUE	Model	Temporary dynamic			
SYSTEM.DEFAULT.REMOTE.QUEUE	Remote				

Scheme: Standard for Queues - Distributed

Last updated: 16:07:07

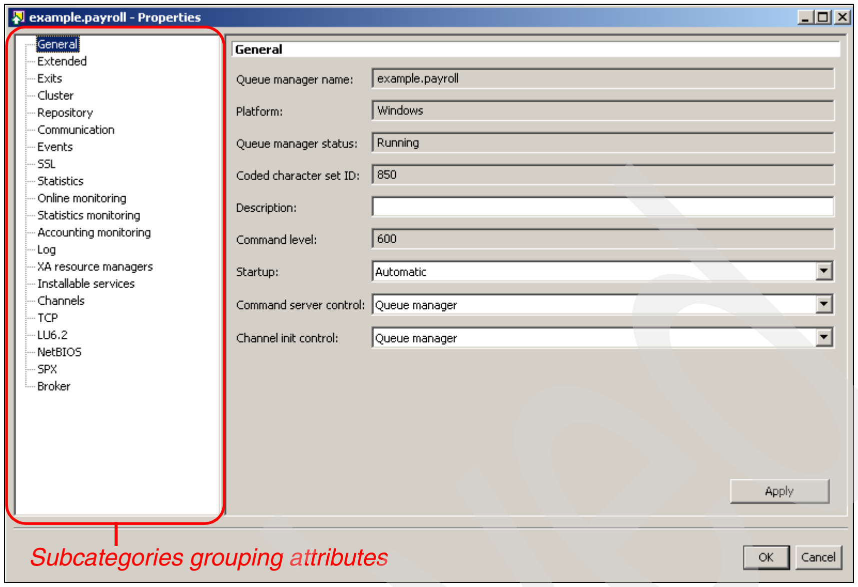
Figure 5-2 WebSphere MQ Explorer: Queues content view for a queue manager, showing system objects

The Filter drop-down list shown in Figure 5-2 can be used to display only objects that match specified criteria in the table, for example, only queues that contain more than 10 messages. Some example filters are provided, and the Manage Filters window available from this drop-down list can be used to configure custom filters and add these filters permanently to the list of filters available.

The Filter drop-down list shown in Figure 5-2 can be used to change the order of the attribute columns shown in the table or to add or remove columns for particular attributes. A default column scheme is provided, and the Manage Schemes window available from this drop-down list can be used to configure custom schemes and add these schemes permanently to the list of schemes available.

Functionality is generally accessed in the WebSphere MQ Explorer by right-clicking an item in the navigator view, or a row in a table, and choosing the required action from the menu displayed.

For example, to display the properties of a queue manager, right-click the queue manager and select **Properties**. Figure 5-3 on page 89 shows the properties window that is displayed for a queue manager hosted on the local machine.


example.payroll - Properties

General
Extended
Exits
Cluster
Repository
Communication
Events
SSL
Statistics
Online monitoring
Statistics monitoring
Accounting monitoring
Log
XA resource managers
Installable services
Channels
TCP
LU6.2
NetBIOS
SPX
Broker

General

Queue manager name: example.payroll
Platform: Windows
Queue manager status: Running
Coded character set ID: 850
Description:
Command level: 600
Startup: Automatic
Command server control: Queue manager
Channel init control: Queue manager

Figure 5-3 Properties window for a queue manager

This properties window has the same layout as other properties windows provided by the WebSphere MQ Explorer. The tree on the left side can be used to access subcategories into which the available properties are grouped.

The Queue Manager Clusters folder can be used to access information related to queue manager clusters. An item is shown under this folder for each queue manager cluster for which any queue manager shown under the Queue Managers folder is a full repository. We discuss this folder in 8.2.2, “Viewing repository information in WebSphere MQ Explorer” on page 196.

Considerations for migrated queue managers

Queue managers that were originally created with WebSphere MQ V5.3 or a previous version of WebSphere MQ and then started after WebSphere MQ V6.0 is installed are called *migrated* queue managers.

The migration stage updates the queue manager data, including all WebSphere MQ objects, and queue manager logs for that queue manager to WebSphere MQ V6.0 data and logs. Migration maintains the existing configuration of the queue manager.

Queue managers prior to WebSphere MQ V6.0 did not automatically start the command server for the queue manager, which the WebSphere MQ Explorer uses when performing the administration of all queue managers. This includes queue managers that are local to the machine running the WebSphere MQ Explorer.

The WebSphere MQ Explorer also requires that a particular WebSphere MQ system object is defined on queue managers in order to perform the administration of those queue managers. This is not created during the migration process.

In order to allow the administration of a migrated queue manager to occur, perform the following steps:

1. End the queue manager if it is running.
2. Issue the following command to create the system objects of that queue that are new for WebSphere MQ V6.0:

   ```
   strmqm -c Queue_Manager_Name
   ```

3. Issue the following command to alter the queue manager so that it will start a command server automatically on startup:
   - Windows:

     ```
     echo ALTER QMGR SCMDSERV(QMGR) | runmqsc Queue_Manager_Name
     ```

   - UNIX:

     ```
     echo "ALTER QMGR SCMDSERV(QMGR)" | runmqsc Queue_Manager_Name
     ```

4. Either restart the queue manager, or issue the following command to avoid a restart now:

   ```
   strmqcsv Queue_Manager_Name
   ```

Introduction to remote queue manager administration

The WebSphere MQ Explorer is capable of connecting to remote queue managers and administering them under the Queue Managers folder.

These remote queue managers do not need to be running on the same platform as the WebSphere MQ Explorer or be the same version of WebSphere MQ.

Remote administration of WebSphere MQ for z/OS queue managers is a new feature available in the WebSphere MQ Explorer. However, the remote WebSphere MQ for z/OS queue manager must be WebSphere MQ V6.0.

The WebSphere MQ Explorer uses a client connection to connect to remote queue managers. It uses the programmable command formats (PCFs) interface described in "MQSC syntax" on page 95.

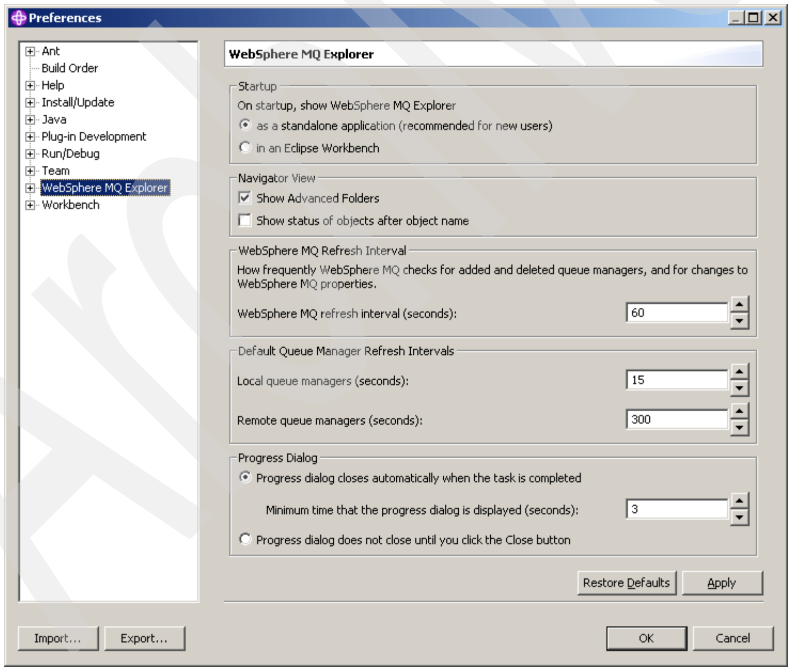
We demonstrate performing remote administration, including the steps required to enable a queue manager for remote administration, in 10.2, “Connect as a client to a queue manager” on page 266.

Note: The WebSphere MQ Explorer is capable of connecting to remote queue managers using a Secure Sockets Layer (SSL) secured client connection. This uses the SSL connection facilities provided by the WebSphere MQ Java API. Details of performing this connection are beyond the scope of this book.

WebSphere MQ Explorer preferences

To configure the options for the WebSphere MQ Explorer, select **Window** → **Preferences** from the menu bar. This opens a Preferences window, which contains a number of different sections containing preferences for the Eclipse workbench within which the WebSphere MQ Explorer runs.

To change the WebSphere MQ properties, select the **WebSphere MQ Explorer** category from the list of categories on the left side of the window, as shown in Figure 5-4.



WebSphere MQ Explorer

Startup
On startup, show WebSphere MQ Explorer
[✓ as a standalone application (recommended for new users)]
in an Eclipse Workbench

Navigator View
[✓ Show Advanced Folders]
Show status of objects after object name

WebSphere MQ Refresh Interval
How frequently WebSphere MQ checks for added and deleted queue managers, and for changes to WebSphere MQ properties.
WebSphere MQ refresh interval (seconds): 60

Default Queue Manager Refresh Intervals
Local queue managers (seconds): 15
Remote queue managers (seconds): 300

Progress Dialog
[✓ Progress dialog closes automatically when the task is completed]
Minimum time that the progress dialog is displayed (seconds): 3
Progress dialog does not close until you click the Close button

Figure 5-4 WebSphere MQ Explorer Preferences window

WebSphere MQ Explorer and the Eclipse project

The WebSphere MQ Explorer is built as a set of *plug-ins* for the Eclipse platform, which is part of the Eclipse project. The Eclipse platform is a universal tools platform that provides the core functionality to implement integrated development environments (IDEs), administration interfaces, and other applications.

Each one of these applications can exist within the same Eclipse *workbench*, which provides a consistent look and feel for all applications. Each application can customize this look and feel for their application by providing a *perspective*.

The WebSphere MQ Explorer provides a perspective called *WebSphere MQ Explorer* so that the look and feel of the workbench, when it is started as the WebSphere MQ Explorer or when the WebSphere MQ Explorer perspective is manually selected, is tailored for convenient WebSphere MQ administration.

Every application in the Eclipse platform is constructed as a set of plug-ins that build on the functionality of existing plug-ins in the Eclipse platform. The plug-ins forming an application can themselves make functionality available for applications to build on as plug-ins to that application. Each area in an application where another application can extend functionality as a plug-in is called an *extension point*.

WebSphere MQ provides a number of extension points to allow flexible growth in the functionality provided by the WebSphere MQ Explorer through future WebSphere MQ plug-ins or plug-ins developed by a third party.

Note: By default, the WebSphere MQ Explorer starts in a stand-alone mode, which does not provide access to the full Eclipse workbench. To make the full workbench available, select **Window** → **Preferences**. Then, select **in an Eclipse Workbench** from the available options. You must restart the WebSphere MQ Explorer for this to take effect.

5.2.2 WebSphere MQ Explorer Healthcheck plug-in

An example of such a plug-in that adds additional problem determination aids to the WebSphere MQ Explorer through the extension points provided is the WebSphere MQ Explorer Healthcheck plug-in.

The WebSphere MQ Explorer Healthcheck plug-in is provided in SupportPac MH01. See the following Web page for more information:

http://www.ibm.com/support/docview.wss?rs=171&uid=swg24010096

5.2.3 WebSphere MQ control commands

WebSphere MQ for UNIX platforms and WebSphere MQ for Windows provide a set of commands to perform operations against WebSphere MQ queue managers and the WebSphere MQ product. These commands are executed from the command line interface provided by the individual operating system. Unless otherwise stated, the path to these commands is added to the operating system command search path by the WebSphere MQ installation process.

5.2.4 WebSphere MQ for iSeries control language commands

IBM OS/400® control language (CL) commands are provided by WebSphere MQ for iSeries to perform operations against WebSphere MQ queue managers and the WebSphere MQ product. Access the main interface to the CL commands provided by WebSphere MQ by using the CL WRKMQM command.

5.2.5 WebSphere MQ for z/OS commands

WebSphere MQ for z/OS provides a set of commands that can be issued against a queue manager subsystem from a z/OS console, or equivalent, such as the System Display and Search Facility (SDSF).

We introduce the queue manager subsystem in WebSphere MQ for z/OS in 5.3.4, “Queue manager structure and creation” on page 102.

5.2.6 WebSphere MQ Script (MQSC) commands

Performing configuration using the WebSphere MQ Explorer can have drawbacks in a production environment. Changes made to a queue manager are not logged, and consistently tracking changes made using a graphical user interface (GUI) can be a difficult task.

Using a scripting interface to execute configuration commands against a queue manager allows a business to implement change management procedures to track and log the executed commands. By documenting scripted commands used to create and configure a queue manager, small modifications to those commands allow duplicates of that queue manager to be created, for example, to scale the system to another machine.

You can create scripts to perform common administration commands, and the output from these scripts can be processed in order to generate a pass/fail result with associated diagnostic information.

WebSphere MQ provides the WebSphere MQ Script (MQSC) scripting interface to a queue manager to provide this functionality. In combination with the

WebSphere MQ control commands described in 5.2.3, “WebSphere MQ control commands” on page 93, all actions against a queue manager can be scripted using a third-party scripting language such as Perl or a Korn shell.

Executing MQSC commands

MQSC commands are executed against a queue manager as follows:

- WebSphere MQ for Windows and UNIX:
  The **runmqsc** WebSphere MQ control program provides an interface to execute MQSC commands against a queue manager. This program accepts commands on the standard input of the command line interface from which it is executed. To start an interactive MQSC session against a queue manager, use the following syntax:

  ```
  runmqsc Queue_Manager_Name
  ```

  If a set of MQSC commands has been saved in a file, the contents of this file can be passed into the **runmqsc** command on standard input using the following syntax:

  ```
  runmqsc Queue_Manager_Name < filename
  ```

  The **runmqsc** command can also be used to execute MQSC commands against a remote queue manager. Refer to Chapter 6, “Administering remote WebSphere MQ objects,” in *WebSphere MQ System Administration Guide*, SC34-6584.

- WebSphere MQ for iSeries:
  MQSC commands can be executed interactively by using the RUNMQSC CL command.

  MQSC commands can also be executed from a script contained within a source physical file using the STRMQMMQSC CL command.

  Note: WebSphere MQ for iSeries also provides CL commands that can be used to interactively perform the functions of MQSC commands using a panel-based interface. To access these WebSphere MQ CL commands, use the WRKMQM CL command.

- WebSphere MQ for z/OS:
  MQSC commands are executed against the subsystem for a particular queue manager.

  Note: On WebSphere MQ for z/OS, a set of operations and control panels are provided that can be used to interactively perform the functions of MQSC commands. These are accessible from Time Sharing Option (TSO) and Interactive System Productivity Facility (ISPF).

MQSC syntax

The syntax of MQSC is quite simple. The general format of a command is:

```
COMMAND OBJTYPE('Object_Name') ATTR1(VALUE) ATTR2('value') ATTR3
```

Where *OBJTYPE* is the object type, *COMMAND* is one of a set of command keywords valid for that object type, and *ATTR1*, *ATTR2*, and *ATTR3* are attribute names valid for that object type.

Some command keyword and object type combinations, such as ALTER QMGR, do not require an object name. Some attributes do not require a value. Many commands and object types have a shorthand version, for example, ALT for ALTER.

For some command keywords, a type and a subtype must both be specified. An example is:

```
DEFINE CHANNEL('my.channel') CHLTYPE(RCVR)
```

Each combination of a command keyword and an object type takes a specific set of attributes. Some of these attributes are required. In order to specify an empty value for an attribute, use ATTR( ), with a space between the parentheses.

Some attributes take multiple values. Where this is the case, these values are separated by commas. An example is:

```
ALTER NAMELIST('my.namelist') NAMES(NAME1,'name2')
```

If a valid combination of command keyword and object type are specified, but the attributes specified are not correct for that combination, or required attributes are missing, the command is rejected and a summary of the usage of that combination is displayed.

Note: Object names and attribute values, that are not enclosed in single quotation marks are automatically converted to uppercase. Therefore, if lowercase object names or attribute values are required, you *must* use single quotation marks.

Attribute values that contain special characters, such as a parenthesis, must be contained in single quotation marks.

MQSC is not case-sensitive for the keywords, such as COMMAND, OBJTYPE, ATTR1, ATTR2, and ATTR3 in the previous example of the general format.

Some commonly used command keywords include:

- DEFINE or DEF:
  Create a new object of a particular type and name with the attributes values specified. The REPLACE attribute can be used in a DEFINE command to cause an existing object of the same name and type to be replaced. The LIKE attribute can be used to specify the name of another object of the same type, the attribute values of which are used for any attributes not specified in the command.
- ALTER or ALT:
  Alter an existing object of a particular type and name, changing the attributes specified to the values given.
- DELETE:
  Delete an existing object of a particular type and name.
- DISPLAY or DIS:
  Display the specified attributes of existing objects with a particular type and name. The special attribute name ALL can be used to display all attributes for each object. If no attributes are specified, a default set of attributes are displayed for each object.

  An asterisk (*) wildcard character can be used at the *end* of the object name and type values. This cause the command to display the attributes of all objects, with names or object types starting with the value specified before the asterisk. For example, the following command displays all attributes of queues with names starting with `example`:

  ```
  DISPLAY QUEUE('example*') ALL
  ```

  In WebSphere MQ V6.0, additional filtering of the output of DISPLAY commands is available using the WHERE keyword. In the parenthesis for the WHERE keyword, three values are specified: an attribute name, an operator, and a filter value. For each object, the specified attribute is matched against the filter value using the operator and only if this match is successful are the attributes of the object displayed. For example, the following command shows the depth and description of all queues containing more than 10 messages:

  ```
  DISPLAY QUEUE(*) DESCR CURDEPTH WHERE(CURDEPTH,GT,10)
  ```
- START:
  Cause an existing object of a particular type and name to start, for example, starting a message channel or a listener.
- STOP:
  Cause an existing object of a particular type and name to stop, for example, starting a message channel or a listener.

An MQSC command can span multiple lines by placing a plus sign (+) as the last character on the end of a line after a space character. For example:

```
DEFINE CHANNEL(TO.PAYROLL) +
       CHLTYPE(SDR) +
       CONNAME('another.machine.com(1414)') +
       XMITQ(PAYROLL)
```

A comment line can be placed in an MQSC script by placing an asterisk (*) character as the first character on that line.

For full details of the syntax of MQSC commands and details of all the commands and the descriptions of the attributes valid for those commands, see *WebSphere MQ Script (MQSC) Command Reference*, SC34-6597, available at:

http://www.ibm.com/software/integration/wmq/library/

5.2.7 Programmable command formats (PCFs)

Programmable command formats (PCFs) provide a programming interface into a queue manager. A matching PCF command exists for each MQSC command that can be used against that queue manager. Matching PCF parameters are provided for each MQSC attribute.

The PCF commands are processed by the *command server* of the queue manager. This performs the action specified in each PCF command and generates a reply message with the result of that command.

The interface into the command server is provided with a standard request/reply model. This means that it processes requests from a queue and sends replies back to the reply-to queue specified by the application that put the request. The queue from which requests are retrieved by the command server is SYSTEM.ADMIN.COMMAND.QUEUE.

The detail of generating and issuing individual PCF command messages, or processing the responses, is beyond the scope of this book. Refer to *WebSphere MQ Programmable Command Formats and Administration Interface*, SC34-6598, available at:

http://www.ibm.com/software/integration/wmq/library/

The MS0B SupportPac can simplify the process of using the PCF interface into a queue manager from a Java application. See the following Web page for details:

http://www.ibm.com/support/docview.wss?rs=171&uid=swg24000668&loc=en_US&cs=utf-8&lang=en

A queue manager created with WebSphere MQ V6.0 on Windows, UNIX, and iSeries automatically starts the command server for the queue manager when the queue manager starts. This functionality can be disabled by changing the SCMDSERV attribute to `MANUAL` on the queue manager object in MQSC, or setting Command server control to **Manual** on the Properties window for the queue manager in the WebSphere MQ Explorer.

For queue managers created on these platforms prior to WebSphere MQ V6.0, including those that have been migrated to WebSphere MQ V6.0, the command server is not started automatically. The command server is started as follows:

- WebSphere MQ for Windows and UNIX:

  ```
  strmqcsv Queue_Manager_Name
  ```

- WebSphere MQ for iSeries:

  ```
  STRMQMCSVR MQMNAME('Queue_Manager_Name')
  ```

- In WebSphere MQ for z/OS, processing of PCF commands by the command server is only available with WebSphere MQ for z/OS V6.0.

  On WebSphere MQ for z/OS, start the command server using the command:

  ```
  START CMDSERV MQSC
  ```

5.3 The queue manager

Queue managers are the core element in a WebSphere MQ message queuing infrastructure. Every application that accesses the infrastructure must connect to a queue manager in order to access that infrastructure. Applications can only retrieve messages from queues hosted by the queue manager to which they are connected.

Note: Queue sharing groups on WebSphere MQ for z/OS allow multiple queue managers to host a shared queue. Refer to 5.3.3, “Queue sharing groups on WebSphere MQ for z/OS” on page 100 for more information.

The queue manager provides an entry point for an application into the message queuing infrastructure. An application can send messages through the queue manager to which it is connected to queues hosted on other queue managers within the message queuing infrastructure.

For messages to flow between queue managers in the infrastructure, each queue manager must have a network link to the queue managers to which it can route messages. Messages might pass through multiple queue managers on route from the application that produced a message to the final destination queue for that message.

A machine with a WebSphere MQ server installation can host multiple queue managers. The number of queue managers that can be hosted by a machine simultaneously is only limited by the resources provided by that machine.

5.3.1 Queue manager naming

Each queue manager has a name. The name of a queue manager should be unique within the WebSphere MQ infrastructure to allow each queue within that infrastructure to be a unique destination. The name of the queue manager is used when performing a connection to a queue manager and when specifying the location of a queue within the infrastructure.

WebSphere MQ only enforces a unique name for a queue manager name within the queue managers hosted on the same machine.

Choosing a suitable name for a queue manager is important. The name can reflect the usage of that queue manager, the machine name, and the location.

We recommend that you consider growth in the infrastructure when naming a queue manager, such as adding queue managers hosted on the same machine or joining multiple WebSphere MQ infrastructures in the future.

After a queue manager has been created, the name cannot be changed. In order to rename a queue manager, it must be deleted and then re-created, including all of the configuration items for that queue manager.

On WebSphere MQ for z/OS, the name of a queue manager can be a maximum of four characters in length. This name can contain only uppercase alphabetic characters, numeric characters, and the following symbol characters: $ # @.

On other platforms, the name of a queue manager can be a maximum of 48 characters in length. This name can contain uppercase and lowercase alphabetic characters, numeric characters, and the following symbol characters: . / _ %. The name of a queue manager is *case-sensitive*, so QMGR1 represents a separate queue manager from Qmgr1.

Mixed case queue manager names, or queue manager names that differ only by case, are fully supported by WebSphere MQ. However, consider use of case carefully in order to reduce confusion for applications connecting to queue managers or routing messages to queue managers.

5.3.2 WebSphere MQ objects

WebSphere MQ *objects* are defined and configured within a queue manager. These objects are individual items that together make up the queue manager and its configuration. Each object has a type, a name, and a number of *attributes* that configure that object.

A large part of this chapter discusses the types of objects that can be contained within a queue manager, their definition, and configuration and the functionality provided by each type of object. Examples of objects, which we explain in greater detail throughout this chapter, include:

- The queue manager itself
- The queues hosted by a queue manager

A number of objects are automatically defined when a queue manager is created. The objects have names starting with SYSTEM to distinguish them from the objects created by an administrator of WebSphere MQ and are generally referred to as *system objects*.

There are a number of different system objects, which we discuss individually in relevant sections of this chapter. However, no system objects should be removed by an administrator. The usage made by WebSphere MQ of these objects falls into the following categories:

- Objects used internally by WebSphere MQ:
  Some objects are required for the operation of certain features of WebSphere MQ. These objects should not be altered by an administrator.
- Objects used to provide default functionality:
  A small number of system objects are provided as the default object to perform a particular function. It is generally recommended that an administrator defines their own objects, with their own naming conventions, to replace the functionality provided by these objects.
- System default objects:
  For each object type, there is a system object with a name as follows:

  ```
  SYSTEM.DEFAULT.OBJECT.TYPE
  ```

  Each new object defined of that type, by default, inherits the same attributes as the system default object for that type. Changing the properties of a system default object changes the attributes of new objects defined of that type. However, it does not change the attributes of objects already defined.

  An object can also be created based on the attributes of any another object that has already been defined of that type. This new object is said to be created *like* the existing object.

5.3.3 Queue sharing groups on WebSphere MQ for z/OS

WebSphere MQ for z/OS operates on instances of the z/OS operating system, with IBM @server zSeries® mainframe hardware. An instance of the z/OS operating system, which can be running on a logical partition (LPAR), is generally referred to in this book as a *z/OS image*.

WebSphere MQ for z/OS builds on the capabilities of the z/OS platform to provide some additional features that are not provided on other platforms. The most significant of these features is queue sharing groups (QSGs).

Multiple queue managers that are members of a QSG have access to *shared queues* contained within that QSG. Each shared queue is available to all queue managers that are members of the QSG, in a similar way to hosting that queue locally to the queue manager.

This means that one application connected to a queue manager can put a message to a shared queue, and then a second application connected to a second queue manager within the QSG can get that message from the same shared queue.

Without using the functionality of shared queues, this message would need to be transferred to a queue hosted on the second queue manager by a distributed or cluster message channel before the second application could get the message.

Another significant benefit of a QSG is that if one queue manager within a QSG fails, the other queue managers within the QSG can continue to process data from shared queues contained within that QSG.

QSGs build on the functionality provided by connecting z/OS images together in a *sysplex*. All queue managers that are members of a QSG must be hosted on z/OS images within the same sysplex.

A sysplex provides a *coupling facility (CF)*, which allows multiple z/OS images within a sysplex to share data. The WebSphere MQ for z/OS product uses the CF in combination with the facilities provided by the IBM DB2® database product.

Due to this, each queue manager within a QSG must have access to DB2. The DB2 instances that the queue managers within a QSG access must be within the same *data-sharing group*. A data-sharing group is a DB2 feature that enables multiple DB2 instances to share a repository of information.

WebSphere MQ for z/OS uses the CF and DB2 to share the definition of the queue, and the messages contained within that queue, between the queue managers that are members of the QSG. When defined on one queue manager within the QSG, all queue managers within the QSG can access a shared queue.

Each QSG has a name. The rules for a QSG name are the same as the rules previously described for a queue manager name on WebSphere MQ for z/OS.

Note: WebSphere MQ for z/OS Version 6.0 provides some functional enhancements relating to shared queues and QSGs, as summarized in the following points:

- The maximum length of a message that can be put onto a shared queue has been increased from 63 KB to 100 MB. When a message larger than 63 KB is put on a shared queue, a placeholder is stored in the CF (4 KB) and the message data is stored using DB2.
- If QSGs are being used, and the administration structure fails, the queue managers that are active in the QSG no longer terminate. Instead, work is suspended, the structure is automatically reallocated and rebuilt, and then the work continues.

5.3.4 Queue manager structure and creation

The details of the operation and configuration of a WebSphere MQ queue manager are specific to each individual WebSphere MQ platform.

Providing the full details of the operation of the queue manager is beyond the scope of this book. However, we provide an overview in this section for a number of WebSphere MQ Version 6.0 platforms. We highlight some important considerations for each platform discussed.

WebSphere MQ for Windows

In WebSphere MQ for Windows, a queue manager runs as a set of processes hosted by the operating system.

This book assumes that the default installation directory has been specified when installing WebSphere MQ for Windows V6.0. If a different directory has been specified, substitute the custom directory for the following directory in all examples: C:\Program Files\IBM\WebSphere MQ.

Each queue manager owns and maintains a number of files on the file system:

- Queue manager data directory:
  The queue manager data directory contains the definitions of objects, message data, and other queue manager data. The default location of this directory is:

  C:\Program Files\IBM\WebSphere MQ\Qmgrs\Queue_Manager_Name
- Queue manager log files:
  Files containing the log for a queue manager. We discuss logging in 5.3.13, “Logging” on page 116. The default location for these files is:

  C:\Program Files\IBM\WebSphere MQ\log\Queue_Manager_Name

The *Queue_Manager_Name* value in these paths might not identically match the name of the queue manager. For details of how a directory name in the path is generated from the queue manager name, see *WebSphere MQ System Administration Guide*, SC34-6584. The most regular difference is that symbol characters in a queue manager name are transformed in the directory name. The symbol . is changed to ! and / is changed to &.

The core configuration information for a queue manager is stored in the Windows registry. This information includes information about how logging is performed for a queue manager and configuration related to communication protocols. Changes made to this configuration information might not be detected by a running queue manager until it is stopped and restarted.

The configuration for a queue manager in the Windows registry can be altered using the WebSphere MQ Explorer. To access the configuration information of a queue manager, right-click the icon for the queue manager in the navigator and select **Properties**.

The information contained in the Windows registry for a queue manager can also be altered using the `amqmdain reg` WebSphere MQ control command.

Note: Editing any information contained in the Windows registry directly is an advanced task. We do *not* recommend this when performing the WebSphere MQ configuration.

For full information regarding the WebSphere MQ configuration information contained in the Windows registry, see Part 4, “Configuring WebSphere MQ,” in *WebSphere MQ System Administration Guide*, SC34-6584.

A WebSphere MQ for Windows queue manager can be created in the following ways:

- Using the WebSphere MQ Explorer
  You can configure WebSphere MQ queue managers using the Create Queue Manager wizard in the WebSphere MQ Explorer. Access this wizard by right-clicking the **Queue Managers** folder and selecting **New** → **Queue Manager**.
- Using the `crtmqm` WebSphere MQ control command.
  The `crtmqm` WebSphere MQ control command is documented in Part 6, “WebSphere MQ control commands,” in *WebSphere MQ System Administration Guide*, SC34-6584.

Note: In order to create a queue manager, the user performing the procedure must be a member of the mqm group. This group is created automatically during the installation of the WebSphere MQ product.

You can specify parameters when creating a queue manager that configure the initial values for important configuration information held in the Windows registry. Some of the configuration relating to logging cannot be changed after the queue manager is created. See 5.3.13, “Logging” on page 116 for more information.

The parameters specified when creating a queue manager have default values. These defaults are stored, along with other WebSphere MQ configuration information that is not specific to an individual queue manager, in the Windows registry.

The WebSphere MQ configuration can be altered using the WebSphere MQ Explorer. To access the WebSphere MQ configuration, right-click the **WebSphere MQ** icon in the navigator and select **Properties**.

The WebSphere MQ configuration can also be altered using the `amqmdain reg` WebSphere MQ control command.

WebSphere MQ for UNIX platforms

In WebSphere MQ for UNIX platforms, a queue manager runs as a set of processes hosted by the operating system.

Each queue manager owns and maintains a number of files on the file system:

- Queue manager data directory:
  The queue manager data directory contains the definitions of objects, message data, and other queue manager data. The default location of this directory is:

  /var/mqm/qmgrs/*Queue_Manager_Name*

- Queue manager log files:
  Files containing the log for a queue manager. We discuss logging in 5.3.13, “Logging” on page 116. The default location for these files is:

  /var/mqm/log/*Queue_Manager_Name*

  Note: For performance reasons, we recommend that you mount the /var/mqm/qmgrs and /var/mqm/log file systems on different physical file systems.

The *Queue_Manager_Name* value in these paths might not identically match the name of the queue manager. For details of how a directory name in the path is generated from the queue manager name, see the “Understanding WebSphere MQ file names” section in *WebSphere MQ System Administration Guide*, SC34-6584. The most regular difference is that symbol characters in a queue manager name are transformed in the directory name. The symbol . is changed to ! and / is changed to &.

The core configuration information for a queue manager is stored in the following file on the file system, based on the default queue manager data directory: /var/mqm/qmgrs/*Queue_Manager_Name*/qm.ini.

This contains information about how logging is performed for the queue manager and configuration related to communication protocols. Changes made to this configuration information might not be detected by a running queue manager until it is stopped and restarted.

On all UNIX platforms, this file can be edited directly using a text editor such as vi or emacs.

On UNIX platforms where it is supported, the configuration information contained in this file can be changed using the WebSphere MQ Explorer. To access the configuration of a queue manager, right-click the icon for the queue manager in the navigator view and select **Properties**.

A queue manager is created in WebSphere MQ for UNIX platforms using the `crtmqm` WebSphere MQ control command. Part 6, “WebSphere MQ control commands,” in *WebSphere MQ System Administration Guide*, SC34-6584, documents this command.

On UNIX platforms where it is supported, you can create a queue manager using the Create Queue Manager wizard in the WebSphere MQ Explorer. Access this wizard by right-clicking the **Queue Managers** folder in the navigator view and selecting **New** → **Queue Manager**.

Note: In order to create a queue manager, the user performing the procedure must be a member of the mqm group. This group is created automatically during the installation of the WebSphere MQ product.

You can specify parameters when creating a queue manager to configure the initial values for important configuration information held in the qm.ini file. Some of the configuration relating to logging cannot be changed after the queue manager is created. See 5.3.13, “Logging” on page 116 for more information.

The parameters specified when creating a queue manager have default values. These defaults are stored, along with other WebSphere MQ configuration information that is not specific to an individual queue manager, in the /var/mqm/mqs.ini file.

On all UNIX platforms, this file can be edited directly using a text editor such as vi or emacs.

On UNIX platforms where it is supported, the WebSphere MQ configuration can also be altered using the WebSphere MQ Explorer. To access this configuration information for a queue manager, right-click the icon for the queue manager in the navigator view and select **Properties**.

Note: If any queue managers are running on a machine, or any applications running on that machine make connections to a queue manager, take care when editing the mqs.ini file.

In these circumstances, it must not be edited by making a duplicate copy and then renaming this copy to overwrite the existing mqs.ini file. If these steps need to be performed, all queue managers running on that machine and any applications running on that machine that make connections to a queue manager must be stopped before overwriting the mqs.ini file.

WebSphere MQ for iSeries

In WebSphere MQ for iSeries, a queue manager runs as a number of batch jobs. By default, these batch jobs run within the QMQM subsystem that is created when WebSphere MQ for iSeries is installed. For details of how WebSphere MQ issues these batch jobs, see *WebSphere MQ for iSeries V6.0 System Administration Guide*, SC34-6586.

Note: This book makes no further reference to iSeries batch jobs. Where the term *process* is used, it generally also refers to an iSeries batch job.

Ensure that the QMQM subsystem is running before attempting to issue any WebSphere MQ CL commands. Use the following command to start the QMQM subsystem:

```
STRSBS QMQM/QMQM
```

Each queue manager owns and manages a set of resources used in the operation of that queue manager:

- The queue manager library:
  Each queue manager has an associated library. This library contains the journals that form the log for the queue manager. The name of the library is based on the name of the queue manager.
- Queue manager data directory:
  The queue manager data directory on the integrated file system (IFS) contains the definitions of objects, message data, and other queue manager data. The default location of this directory is:

  /QIBM/UserData/mqm/qmgrs/*Queue_Manager_Name*

The name of the library for a queue manager and *Queue_Manager_Name* in the above IFS path might not identically match the name of the queue manager. For details of how the library and IFS directory names are generated from the queue manager name, see *WebSphere MQ for iSeries V6.0 System Administration Guide*, SC34-6586.

The core configuration information for a queue manager is stored in the following file on the file system, based on the default queue manager data directory:

/QIBM/UserData/mqm/*Queue_Manager_Name*/qm.ini

This information includes information about how logging is performed for the queue manager and configuration related to communication protocols. Changes made to this configuration information might not be detected by a running queue manager until it is stopped and restarted.

This file can be edited directly using the EDTF CL editor.

A queue manager is created in WebSphere MQ for iSeries using the CRTMQM WebSphere MQ for iSeries CL command. Refer to *WebSphere MQ for iSeries V6.0 System Administration Guide*, SC34-6586, for more information.

Note: In order to create a queue manager, the user performing the procedure must be a member of the QMQMADM group. This group is created automatically during the installation of the WebSphere MQ product.

The parameters of this command configure the initial values for important configuration information held in the qm.ini file. Some of the configuration relating to logging cannot be changed after the queue manager is created. See 5.3.13, “Logging” on page 116 for more information.

WebSphere MQ configuration that is not specific to an individual queue manager is stored in the /QIBM/UserData/mqm/mqs.ini file in IFS.

This file can be edited directly using the EDTF CL editor.

WebSphere MQ for z/OS

For information about the operation of queue managers in WebSphere MQ for z/OS, refer to *WebSphere MQ for z/OS V6.0 Concepts and Planning Guide*, GC34-6582.

5.3.5 The default queue manager

Within the queue managers hosted on one machine, a queue manager can be configured to be the default queue manager.

If an application connects to WebSphere MQ without specifying a queue manager name, and the application is hosted on the same machine as the queue manager, the application connects to the default queue manager.

Some WebSphere MQ control commands also use the default queue manager if no queue manager is specified.

WebSphere MQ for Windows, iSeries, and UNIX

In WebSphere MQ for Windows, iSeries, and UNIX, the default queue manager is specified when creating the queue manager as one of the options during the creation process.

Specifying this option causes the name of the queue manager to be recorded in the *Default queue manager* parameter in the WebSphere MQ configuration. This can be changed at a later time by altering this parameter in the WebSphere MQ configuration. We discuss altering the WebSphere MQ configuration on Windows, UNIX, and iSeries in 5.3.4, “Queue manager structure and creation” on page 102.

WebSphere MQ for z/OS

The method for specifying the default queue manager is dependent on the environment from which the application is connecting. For details, see the “Writing a WebSphere MQ application” section in *WebSphere MQ Application Programming Guide*, SC34-6595.

5.3.6 The queue manager object

Configuration information that can be altered for a queue manager while that queue manager is running is contained within the queue manager object. This is administered in the same way as the other objects of the queue manager.

Using the WebSphere MQ Explorer, attributes of the queue manager object are contained in the same Properties window as the configuration attributes for the queue manager.

5.3.7 Starting and ending a queue manager

The method of starting or ending a queue manager is specific to each WebSphere MQ platform. This section outlines the steps required on each platform.

Note: WebSphere MQ V6.0 made significant changes to the names and structure of the processes that implement a queue manager. This applies to all platforms with the exception of WebSphere MQ for z/OS. The details of these changes is beyond the scope of this book. However, this can affect existing scripts that perform the termination or cleanup of a queue manager or that detect whether a queue manager is active on a machine. For more information about the processes that are part of a WebSphere MQ V6.0 queue manager, refer to the following publications:

- Windows and UNIX:
  WebSphere MQ System Administration Guide, SC34-6584, Appendix D, “Stopping and removing queue managers manually”
- iSeries:
  WebSphere MQ for iSeries V6.0 System Administration Guide, SC34-6586, “Work management” section

WebSphere MQ for Windows, UNIX, and iSeries platforms

In order to start and end queue managers, the user performing the procedure must be a member of the mqm group. This group is created automatically during the installation of the WebSphere MQ product.

To start a queue manager, use one of the following methods:

- Using the WebSphere MQ Explorer:
  Right-click the icon for a queue manager in the navigator view and select **Start**.

Note: The WebSphere MQ Explorer on Windows can be used to configure a queue manager to start automatically with the machine. This is performed by right-clicking the icon for the queue manager in the navigator view and selecting **Properties**. Then, change the Startup value from Manual to **Automatic**.

- Using the `strmqm` WebSphere MQ control command:
  The `strmqm` command is available on WebSphere MQ for Windows and UNIX. The `strmqm` command is documented in Part 6, "WebSphere MQ control commands," in *WebSphere MQ System Administration Guide*, SC34-6584.

  Note: When using the `strmqm` WebSphere MQ control command on Windows, the queue manager is started under the user identifier of the current user. This means that the queue manager ends when the current user logs out of the machine. Consider using the `amqmdain qmgr start` command instead of the `strmqm` command.

- Using the `amqmdain qmgr start` WebSphere MQ control command:
  The `amqmdain qmgr start` WebSphere MQ control command is only available on WebSphere MQ for Windows. A queue manager that is started using `amqmdain qmgr start` command remains running after the user executing the command has logged out of the machine. This command is documented in Part 6, "WebSphere MQ control commands," in *WebSphere MQ System Administration Guide*, SC34-6584.
- Using the STRMQM CL command:
  The STRMQM CL command is available on WebSphere MQ for iSeries. The STRMQM CL command is documented in *WebSphere MQ for iSeries V6.0 System Administration Guide*, SC34-6586.

When ending a queue manager, it is important to consider that applications might still have active connections to that queue manager. Due to this, WebSphere MQ has three methods of ending a queue manager.

Use these three methods in the following order of preference. If one method fails to end the queue manager within a required time limit, move to the next. The next command can be issued while a previous, less severe, command is still in progress.

1. Quiesced shutdown:
   This is the default method for ending a queue manager. The queue manager waits until all applications have disconnected normally from the queue manager before ending. Applications can continue to perform work against the queue manager until they disconnect. Applications can choose to be

informed when a queue manager is quiescing when performing actions against the queue manger so that they can detect that the queue manager is quiescing and choose to disconnect.

2. Immediate shutdown:
   All current actions being performed against the queue manager are allowed to complete successfully before the queue manager ends. However, any new actions issued against the queue manager fail.
3. Preemptive shutdown:
   This immediately ends the queue manager. Only use this method when options 1 and 2 have failed, because it can have unpredictable consequences for connected applications.

If these three methods fail to end a queue manager, consult the following documentation:

- The “Stopping a queue manager manually” section in *WebSphere MQ System Administration Guide*, SC34-6584
- The “Quiescing WebSphere MQ for iSeries” section in *WebSphere MQ for iSeries V6.0 System Administration Guide*, SC34-6586

All three methods for ending a queue manager are available using the same procedure. To end a queue manager, use one of the following methods:

- Using the WebSphere MQ Explorer:
  Right-click the icon for a queue manager in the navigator view and select **Stop**.
- Using the **endmqm** WebSphere MQ control command:
  The **endmqm** command is available on WebSphere MQ for Windows and UNIX. The **endmqm** command is documented in Part 6, “WebSphere MQ control commands,” in *WebSphere MQ System Administration Guide*, SC34-6584.

  Note: A queue manager which was started with **amqmdain qmgr start**, can be ended using **endmqm**.

- Using the **amqmdain qmgr end** WebSphere MQ control command:
  The **amqmdain qmgr end** WebSphere MQ control command is only available on WebSphere MQ for Windows. This command is documented in Part 6, “WebSphere MQ control commands,” in *WebSphere MQ System Administration Guide*, SC34-6584.
- Using the ENDMQM CL command:
  The ENDMQM CL command is available on WebSphere MQ for iSeries. The ENDMQM CL command is documented in *WebSphere MQ for iSeries V6.0 System Administration Guide*, SC34-6586.

WebSphere MQ for z/OS

The z/OS subsystem associated with each WebSphere MQ queue manager is started during the initial program load (IPL) time.

You can issue the START QMGR command against the queue manager subsystem to start the queue manager.

The z/OS Automatic Restart Manager (ARM) can be used to automatically restart a queue manager if it fails. This is documented in the “Using the z/OS Automatic Restart Manager (ARM)” section in *WebSphere MQ for z/OS V6.0 System Administration Guide*, SC34-6585.

When ending a queue manager, it is important to consider that applications might still have active connections to that queue manager. Due to this, WebSphere MQ for z/OS has multiple methods of ending a queue manager:

- The STOP QMGR MODE(QUIESCE) command:
  This is the default method for ending a queue manager. The queue manager waits until all applications have disconnected normally from the queue manager before ending. Applications can continue to perform work against the queue manager until they disconnect. Applications can choose to be informed when a queue manager is quiescing when performing actions against the queue manger so that they can detect that the queue manager is quiescing and choose to disconnect. The queue manager is deregistered with ARM so that it is not automatically restarted.
- The STOP QMGR MODE(FORCE) command:
  This forcefully ends the queue manager. Use this method when a quiesced shutdown has failed to complete in a required period of time, or if there are no active connections to the queue manager. The queue manager is deregistered with ARM so that it is not automatically restarted.
- The STOP QMGR MODE(RESTART) command:
  This ends the queue manager in the same way as a STOP QMGR MODE(FORCE) command. However, the queue manager is not deregistered with ARM. Therefore, if ARM has been configured to automatically restart the queue manager, the queue manager restarts.

5.3.8 Providing network access to a queue manager

Queue managers and clients within the infrastructure need to establish communication with a queue manager over a network. To do this, they use an underlying communications protocol. The following protocols can be used for WebSphere MQ communication:

- Transmission Control Protocol/Internet Protocol (TCP/IP)
- SNA LU 6.2 (Windows and z/OS only)

- NetBIOS (Windows only)
- SPX (Windows only)

This book only discusses the TCP/IP protocol. For details of the other protocols, see *WebSphere MQ System Administration Guide*, SC34-6584.

In order for a communication link to be established over TCP/IP, the queue manager must be *listening* for that connection to arrive on a particular *port*.

The *IP address* or *host name* of the machine, combined with a port number on which a queue manager has an active listener, provides a *connection name*. This connection name provides an identifier for the queue manager within the TCP/IP network that other queue managers, or clients, can use to establish communication.

Any port number that is not being listened on by another WebSphere MQ queue manager or any other software running on the machine can be used for a queue manager.

The *well-known port number* for WebSphere MQ is 1414. If a single queue manager is hosted by a machine, it is common for this queue manager to listen on TCP/IP port number 1414. If no port number is specified in a connection name, WebSphere MQ assumes that the machine at a given host name or IP address has a queue manager listening on port 1414.

5.3.9 WebSphere MQ listener

On all WebSphere MQ platforms, with the exception of WebSphere MQ for z/OS, listening on TCP/IP is performed by the WebSphere MQ listener process.

This process listens for connections to arrive on a port and then creates a message channel agent (MCA) to process that connection, whether it is a distributed message channel, cluster message channel, or client connection. We discuss message channels and MCAs in 7.1.2, "Message channel agents (MCAs)" on page 157.

The MCA created by the WebSphere MQ listener does not run within its own process on the system. Instead, the MCA is created within a pool of processes by the listener process. The number of processes within this pool is managed automatically by WebSphere MQ based on the number of MCAs active for the queue manager.

This approach is generally called *channel pooling*. Using channel pooling means that each MCA requires less resources than if it were running within its own process. A queue manager can have thousands of connections active at any time depending on the design of the system and the loads on the queue manager.

Note: Prior to Version 5.3, WebSphere MQ did not provide a channel pooling mechanism on UNIX platforms. The operating system inetd listener process was used instead. WebSphere MQ Version 5.3 and WebSphere MQ Version 6.0 still support the use of the inetd operating system listener. However, this does not benefit from the channel pooling features provided by the WebSphere MQ listener, because each MCA runs within its own process.

Listeners are created using the DEFINE LISTENER MQSC command and started using the START LISTENER MQSC command.

In the WebSphere MQ Explorer, listeners can be created and started automatically when creating a queue. Alternatively, create listeners by right-clicking the **Listeners** folder under the queue manager in the navigator view and selecting **New** → **Listener**.

You can start listeners using the WebSphere MQ Explorer by selecting the **Listeners** folder under the queue manager in the navigator view. In the Listeners content page that opens, right-click the listener object and select **Start**.

WebSphere MQ for Windows also supports LU 6.2, NetBIOS, and SPX listeners.

A listener can be configured to automatically start with the queue manager. We recommend that all queue managers have a listener defined that is automatically started with the queue manager, rather than manually starting the listener.

Note: Prior to WebSphere MQ Version 6.0, the WebSphere MQ listener was not a queue manager object. For reference, the following methods for starting a listener are available in WebSphere MQ, V5.3:

- In WebSphere MQ for UNIX platforms V5.3, you must manually run the listener from a command shell using the **`runmqlsr`** WebSphere MQ control command.
- In WebSphere MQ for iSeries V5.3, use the STRMQMLSR CL command to start a listener.
- In WebSphere MQ for Windows V5.3, a listener was started, and you can automatically start a listener with a queue manager using the **`amqmdain crtlsr`** WebSphere MQ control command or the WebSphere MQ Services snap-in.

5.3.10 WebSphere MQ for z/OS channel initiator

On WebSphere MQ for z/OS, listening on TCP/IP is performed within the WebSphere MQ for z/OS channel initiator.

The WebSphere MQ channel initiator, also known as the *mover*, runs in the address space of the queue manager. It hosts all message channel agents (MCAs) for a queue manager, whether they host a distributed message channel, cluster message channel, or client connection. We discuss message channels and MCAs in 7.1.2, "Message channel agents (MCAs)" on page 157.

Start the channel initiator by issuing the START CHINIT command against the subsystem of the queue manager.

Multiple TCP/IP listeners can be started within the channel initiator. Each listener listens on a particular TCP/IP port. A listener is started using the START LISTENER command against the subsystem of the queue manager.

WebSphere MQ for z/OS also supports LU 6.2 listeners.

5.3.11 The dead letter queue

WebSphere MQ provides assured delivery, and because of this, if a message cannot be delivered to its destination queue, or to a transmission queue on route to a destination, a consistent action is taken.

This involves placing the message on the *dead letter queue* of the last queue manager that the message attempted to pass through on route to its destination.

When a queue manager is created, a dead letter queue is not automatically created by WebSphere MQ. A queue must be created, and the queue manager must be configured to use that dead letter queue.

Reasons why a message might not be deliverable include: A queue of the name specified does not exist on the queue manager, the queue manager does not have knowledge of the next queue manager on route to the destination, or a queue already contains the maximum allowed number of messages specified for that queue.

Note: If no dead letter queue has been configured on a queue manager, and a message cannot be delivered to that queue manager from another queue manager, the transfer of all messages over the message channel that connects those two queue managers stops. This message channel is only able to start if configuration is performed on the target queue manager to specify a dead letter queue or to allow the message to be delivered successfully by defining a destination queue. Alternatively, the individual message that cannot be delivered can be manually removed from the queue, but this requires an application to identify and retrieve that message.

Because of this, configure a dead letter queue for *all* queue managers within the message queuing infrastructure.

We discuss dead letter queues and how they are configured in 7.4.11, "Message delivery failures" on page 175.

5.3.12 The command server

WebSphere MQ enables administration to be performed remotely on queue managers. To facilitate this, a *command server* can run within a queue manager. This command server executes commands sent to that queue manager. We describe these commands in 5.2.7, "Programmable command formats (PCFs)" on page 97.

5.3.13 Logging

Logging is a fundamental function performed internally by a queue manager. A queue manager's log is a record of actions performed by the queue manager in the order that they are performed.

All actions performed on persistent messages are logged as l*og records* within this log, as well as configuration changes to the objects of a queue manager and other internal actions.

The method a queue manager uses to write to the log assures that the actions that are logged are not completed until they have been written to reliable storage in the log.

The log data is separate from the queue manager data. The queue manager data contains only the current status of all objects and all messages stored on queues for that queue manager. The methods used by the queue manager to write to the queue manager data can involve buffering data in memory or writing the data to reliable storage using optimized facilities provided by the operating system.

Because of this, it is possible that if the queue manager ends abruptly, for example, because the machine fails abruptly, the queue manager data is in an inconsistent state.

In these circumstances, the queue manager uses the log to rebuild the correct current state of objects during the restart process. This is performed by replaying log records since the last point of consistency between the log and the queue manager data.

A queue manager regularly makes the log data and queue manager data consistent with each other. This process happens during a *checkpoint*. Checkpoints occur automatically while a queue manager is running and when a queue manager ends.

If a queue manager is ended normally, the checkpoint means that a minimal number of log records need to be replayed, and the queue manager starts optimally. If a queue manager ends abruptly, more log records might need to be replayed to start the queue manager.

If an item within the queue manager data becomes corrupted in some way, for example, the message data for a queue, the object related to that item in the queue manager, such as a queue, is marked as *damaged*. If an object is marked as damaged, applications cannot access that object. If the object is a queue, the messages on that queue are unavailable.

5.3.14 Media recovery

It might be possible to recover that object from the queue manager logs. This is called *media recovery*. On WebSphere MQ for Windows and UNIX, the queue manager must have been configured to use *linear logging*, rather than the default *circular logging*, in order for media recovery to be available. On WebSphere MQ for iSeries, the logging can be considered as always being linear logging.

Considerations for logging and media recovery are different on WebSphere MQ for z/OS and are not addressed in this book. Refer to *WebSphere MQ for z/OS V6.0 Concepts and Planning Guide*, GC34-6582.

The following points summarize circular and linear logging:

- Circular logging:
  The queue manager manages the size of the log automatically, without any administration being required. However, the queue manager only ensures that enough information is held to maintain the integrity of business-critical data, as held in persistent messages, and to perform a restart of the queue manager. Media recovery is not available, because the logs do not contain enough information about each object in the queue manager.
- Linear logging:
  The queue manager maintains a continuous log from the point of creation of the queue manager nd does not perform any management of the size of the log. Therefore, the log contains all the information required to rebuild the objects within the log. However, administration is required to archive or delete logs that are no longer required or the available storage resources for the log can eventually be filled.

  A queue manager informs an administrator of the oldest log records required to restart the queue manager. It also informs an administrator of the oldest log records required to perform media recovery of the objects within that queue manager.

  Note: WebSphere MQ V6.0 enables an administrator to determine the oldest log records required to perform a media recovery of an individual queue.

  An administrator can remove any log records that are older than the oldest log record required for restart without affecting the operation of the queue manager. However, an administrator can also choose to keep older log records for a queue manager, possibly in a compressed form on backup media, in order to facilitate the media recovery of objects.

  A *media image* of individual objects within a queue manager, or all of the objects within a queue manager, can be recorded in the log by an administrator. Recording a media image writes all log records required to perform media recovery of that object. Therefore, this can reduce the size of the log required to perform media recovery. Without recording media images, performing media recovery might require log records as old as the original creation of an object being recovered and involve replaying a large number of log records.

We recommend a more detailed understanding of logging for a queue manager that hosts business-critical services. This knowledge can help when choosing the logging mechanism for a queue manager, planning the administration of linear logs, and for understanding how units of work affect logging. Refer to:

- Windows and UNIX:
  WebSphere MQ System Administration Guide, SC34-6584, the “Recovery and problem determination” section
- iSeries:
  WebSphere MQ for iSeries V6.0 System Administration Guide, SC34-6586, the “Backup, recovery and restart” section

5.3.15 Error logs

In WebSphere MQ for Windows, UNIX, and iSeries, important events that occur for a queue manager are written to the queue manager error logs with time stamps to show when those events occurred. These provide similar details for a queue manager to the system logs provided for an operating system. Therefore, they need to be regularly checked by the WebSphere MQ administrator for a machine. Examples of the types of information contained in the queue manager error logs includes:

- Information about the startup and shutdown of that queue manager.
- Information about the connections established to and from that queue manager, including distributed message channels, cluster message channels, and client connections. This includes information about failures.
- Security violations, where an attempt is made by an application to access an object for which it is not authorized to access.
- Unexpected events that occur for the queue manager.

The queue manager error logs are contained in a set of human-readable files, which can be opened in a text viewer. For each queue manager, a set of files are held of a fixed size, 256 KB by default. Each time a file is filled, the queue manager moves onto a new file, up to a maximum of three. The names of these files are as follows:

- AMQERR01.LOG
- AMQERR02.LOG
- AMQERR03.LOG

Note: In WebSphere MQ V6.0, the size of each error log can be configured using the `ErrorLogSize` parameter in the queue manager configuration. WebSphere MQ V6.0 also allows the frequency of logging for very common events, such as establishing communication, to be limited. See the “Configuring WebSphere MQ” section in *WebSphere MQ System Administration Guide*, SC34-6584.

Some events that occur on a machine with WebSphere MQ installed cannot be linked to an individual queue manager. These include information about failures for a client application attempting to connect to a queue manager. These events are logged in a set of error logs called the WebSphere MQ system error logs, in the same format as queue manager error logs.

In a small number of cases, WebSphere MQ might associate an event with a queue manager, but be unable to log the event for that particular queue manager. In these cases, the event is either logged in another set of error logs, called the system queue manager error logs, or a first-failure support technology (FFST) report is generated. For more information, see 12.1.6, “First-failure support technology (FFST)” on page 325.

The error logs are located in the following locations:

- Windows:
  - Queue manager error logs:

    C:\Program Files\IBM\WebSphere MQ\Qmgrs\Queue_Manager_Name\errors
  - WebSphere MQ system error logs:

    C:\Program Files\IBM\WebSphere MQ\errors
  - WebSphere MQ system error logs for a client only installation:

    c:\Program Files\IBM\WebSphere MQ Client\errors
  - System queue manager error logs:

    C:\Program Files\IBM\WebSphere MQ\Qmgrs\@SYSTEM\errors
- UNIX:
  - Queue manager error logs:

    /var/mqm/qmgrs/*Queue_Manager_Name*/errors
  - WebSphere MQ system error logs:

    /var/mqm/errors

- System queue manager error logs:

  /var/mqm/qmgrs/@SYSTEM/errors

- iSeries:
  - Queue manager error logs:

    /QIBM/UserData/mqm/*Queue_Manager_Name*/errors
  - WebSphere MQ system error logs:

    /QIBM/UserData/mqm/errors
  - System queue manager error logs:

    /QIBM/UserData/mqm/&SYSTEM/errors

5.3.16 64-bit hardware

64-bit hardware provides addressing of significantly more memory resources to be available to an individual application than on a 32-bit system. In order for this extra memory resource to be available to applications, the operating system must also support 64-bit addressing.

WebSphere MQ queue managers on UNIX platforms prior to V6.0 did not exploit the additional memory addressing provided by these platforms.

Applications that connected to queue managers from these platforms were only able to exploit the 64-bit addressing provided by these 64-bit platforms by using 64-bit client products provided in SupportPacs. This reduced performance by using a network connection.

WebSphere MQ V6.0 provides 64-bit queue managers on a number of UNIX platforms. These queue managers still accept connections from 32-bit applications using bindings or client connection methods, but can also accept connections from 64-bit applications using bindings connections.

This allows applications providing services to exploit the 64-bit addressing capabilities of the operating system and hardware, while using bindings connections into WebSphere MQ Version 6.0 queue managers.

The queue manager also exploits the 64-bit addressing capabilities of these platforms by allowing the capacity of an individual queue manager to scale beyond the limits imposed by 32-bit addressing.

Note: The 64-bit WebSphere MQ V6.0 queue managers are not necessarily better performing than 32-bit WebSphere MQ V5.3 queue managers on the same hardware. However, in some situations, the performance of WebSphere MQ V5.3 was limited by 32-bit addressing, so in these circumstances, performance improvements can be observed.

On platforms with 64-bit addressing, some internal structures used by WebSphere MQ require more memory resources. Therefore, the ultimate capacity of some 64-bit WebSphere MQ V6.0 queue managers might be lower than a 32-bit WebSphere MQ V5.3 queue manager on the same hardware. However, a 64-bit WebSphere MQ V6.0 queue manager is likely to scale to a larger capacity on a more powerful server that provides a significant quantity of memory resources. This is because some of this memory resource simply cannot be used by a 32-bit queue manager.

See the following Web page for details about the 64-bit platforms currently supported by WebSphere MQ:

http://www.ibm.com/software/integration/websphere/mqplatforms/supported.html

6

Technical introduction to message queuing

This chapter provides details about the core functionality provided by WebSphere MQ for message queuing. It provides information about the objects that can be created within a queue manager and their use. We describe WebSphere MQ messages and how to send and receive those messages using the message queue interface (MQI).

Some of the details of the MQI do not directly apply to object-oriented, or standardized application programming interfaces (APIs), such as the Java Message Service (JMS). However, these APIs build on the core functionality provided by the MQI. Because of this, gaining an understanding of the WebSphere MQ objects, messages, and the MQI remains useful when accessing that infrastructure through standardized APIs.

We discuss the following topics:

- Message queue interface
- Queues
- Triggering

6.1 Message queue interface

The message queue interface (MQI) is a procedural interface into messaging with WebSphere MQ. Therefore, it is only used directly from procedural programming languages, such as C. Many applications developed to access a WebSphere MQ message queuing infrastructure use object-oriented languages, such as Java or C++. Applications written in both procedural and object-oriented languages can also benefit from using standardized interfaces into WebSphere MQ messaging, such as the Java Message Server (JMS) or the Extended Message Service (XMS).

However, these APIs all use the MQI in their underlying implementations, either by calling into modules written in C or by sending MQI commands over a client connection to the queue manager. Therefore, an understanding of the MQI itself can be very useful when writing applications using these APIs and in understanding the flow of messages through the WebSphere MQ infrastructure.

The MQI contains 13 functions, as well as structures and data types. These provide access to the functionality provided by a queue manager for sending and receiving message using a point-to-point model, defining the structure of WebSphere MQ messages, and allowing the attributes of core object types to be queried and modified.

6.1.1 WebSphere MQ message descriptor (MQMD)

A WebSphere MQ message descriptor (MQMD) is associated with every message that resides on a queue within a queue manager. It is a structure containing a number of fields that provide information about the message.

The MQMD for a message is passed into the queue manager by an application separately from the message body, when it puts a message. It is returned by the queue manager separately from the message body when an application gets a message from a queue.

The following list provides a summary of commonly used fields contained in the MQMD:

- Message type (`MsgType`):
  WebSphere MQ messages can be marked as the following types:
  - A *datagram* that does not need a reply, but can request reports.
  - A *request* that requires a reply.
  - A *reply* generated in response to a request.
  - A *report* generated due to processing of a message.

- Report and feedback:
  The report field is used to request report messages that are generated based on the processing of a message by the queue manager or a target application. For example, an application might request a report from the destination queue manager that a message was delivered successfully, or it might request a report from the target application that the message was processed successfully. When a report message is generated, a code is placed in the feedback field by the application, or queue manager, generating the report. This code signifies the reason for the report being generated.
- Reply-to queue (`ReplyToQ`):
  An application putting a request message, or specifying report options, specifies a reply-to queue name in this field. Applications generating a reply to a request, or a report in response to a datagram, put the reply or report to this queue name.
- Reply-to queue manager (`ReplyToQMgr`):
  This specifies the name of the queue manager where the reply-to queue is hosted. WebSphere MQ automatically puts the name of the queue manager to which the message is initially put in this field. An application only needs to modify this field if it requires reports or replies to be sent to a queue on a different queue manager.
- Message identifier (`MsgId`):
  This is an identifier for the message. This can be specified by an application or generated by the queue manager. The message identifier is returned by the queue manager to an application putting a message, allowing for its use in further processing.

  A message identifier generated by a queue manager is unique within the whole WebSphere MQ infrastructure. However, this depends on *all queue managers* within the infrastructure having names that are unique within the first 12 characters.
- Correlation identifier (`CorrelId`):
  This field can be used by applications to provide a link between a message and another message or an application. The most common use is to correlate a reply with a request when using request/reply messaging. An application processing a request copies the message identifier of the request into the correlation identifier of the reply. If replies for multiple requests, or multiple request applications, arrive on the same queue, an application can get the correct reply by only getting messages with the correct correlation identifier.
- Persistence:
  This field defines whether a message is persistent or nonpersistent, as discussed in 3.4.1, “Persistent and nonpersistent messages” on page 41. If not specified explicitly by an application putting a message, this field inherits the default persistence of the queue to which the message is put.

- Priority:
  Messages can be marked with a priority from 0 to 9. WebSphere MQ can be configured so that messages with higher priorities are returned to applications before messages with lower priorities.
- Coded character set identifier, CCSID (`CodedCharSetId`):
  A CCSID identifies a character set that maps particular characters to particular bytes or sets of bytes. For data that represents textual characters, called string data, WebSphere MQ can perform a conversion between the character sets so that the same characters are represented by the same data in the message. We discuss this in 4.3.2, "Data conversion" on page 56. This field defaults to the CCSID of the queue manager to which the message was put, which is generally the CCSID of the machine.
- Encoding:
  The order of the bytes within data that represents a number, called *numeric* data, varies between platforms. WebSphere MQ uses the encoding field to specify how numeric data represents a number. As with the CCSID, conversion can be performed on numeric data by WebSphere MQ. The encoding field defaults to the encoding of the machine where the message was generated.
- Format:
  The format field defines the type of data held within the message. The format field can specify a type of data for the whole message, such as string data, a numeric data type, or binary data. It can also specify a WebSphere MQ structure type, which might be only the first of a number of structures contained in a *chain* of structures within the message. Each of these structures contains various types of data within each structure. The format field defaults to binary data, on which WebSphere MQ performs no conversion.
- Put time (`PutTime`) and put date (`PutDate`):
  The queue manager stores a time and date in these fields, which represent when the message arrived on the queue where it is currently stored.
- Expiry:
  Messages can be given a lifetime, specified in this field in tenths of a second. If the message remains in the WebSphere MQ message queuing infrastructure for more than this time, it expires and becomes unavailable to applications and becomes eligible for deletion by a queue manager. Expired messages t are only deleted when an attempt is made to retrieve or view them by an application. WebSphere MQ V6.0 has an *expiry task* that regularly views all messages on all queues, causing expired messages to be deleted.

The MQMD contains information about the application that put the message to a queue and the user identifier under which that application was running.

The MQMD contains information relating to grouped, or segmented, messages. We discuss these messages in 4.6.7, “Segmentation of messages” on page 71.

6.1.2 Completion codes and reason codes

Every function call provided by the MQI returns a completion code (`CompCode`) and a reason code (`Reason`):

- Completion code:
  The completion code can have the following three values:
  - `MQCC_OK`: The function call completed successfully.
  - `MQCC_WARNING`: The function call partially completed. Check the reason code for details.
  - `MQCC_FAILED`: The function call failed. Check the reason code for details.
- Reason code:
  There are multiple possible reason codes that can be returned from each function call; some represent failures and some represent a partial completion of the command. See the “Reason” section of the Fields description for the individual function call being issued in *WebSphere MQ Application Programming Reference*, SC34-6596.

  The reason code is an integer value. To get the symbolic name from the numeric value in hex or decimal, or to get the numeric value from the symbolic name, use the **`mqrc`** WebSphere MQ command. The following example commands give the same information for the `MQRC_NO_MSG_AVAILABLE` return code:

  ```
  mqrc MQRC_NO_MSG_AVAILABLE
  mqrc 2033
  mqrc 0x7F1
  mqrc 0x000007f1
  ```

  Refer to 12.1.2, “Reason codes” on page 323 for more information.

6.1.3 MQCONN and MQCONNX

Each application, or thread within that application, must make a connection to a queue manager in order to send messages through the message queuing infrastructure. This can be a local queue manager hosted on the same machine as the application, or a remote queue manager hosted on another machine.

Whether an application connects to a local queue manager using *bindings*, or to a remote queue manager using a *client connection*, is not chosen programmatically in the MQI. For C applications using the MQI directly, this choice is made by linking the application either with the bindings or client libraries. When using other APIs, the method of choosing bindings or client connections to a queue manager for an application depends on the API and is discussed in the WebSphere MQ documentation for that API.

The MQI provides two functions that can be used to connect to a queue manager: MQCONN and MQCONNX. These functions differ only in that additional options can be passed to an MQCONNX call in a connection options structure (MQCNO).

Using either call, the information required to perform a connection is the name of the queue manager to which an application needs to connect. If a name is not specified, and the application uses bindings connections, the default queue manager is assumed, as described in 5.3.5, “The default queue manager” on page 108.

An application can only connect to a queue manager if it is authorized to do so. This authorization is based on an identity context. For applications using bindings, this identity context is the user identifier provided by the operating system under which the application is running. For client connections, the identity context can be the user identifier under which the application is running, or the queue manager might have been configured to provide the client with a particular user identifier that exists on the remote machine where the queue manager is hosted.

When successfully connected, the application is provided with a connection handle (`Hconn`) that must be passed to all future MQI calls made by that application.

6.1.4 MQOPEN and MQCLOSE

The MQOPEN MQI function is used to access a particular object within a connected queue manager. A call to MQOPEN must be performed individually for each object, such as a queue, that the application needs to access. Every MQOPEN call should be partnered with a corresponding MQCLOSE call.

An application passes an object descriptor (MQOD) into the MQOPEN call. This describes the object the application wants to open, providing a name for the object. For queues, the name of a queue manager can also be provided. This allows the application to target a particular message to a particular queue manager in the system known to the local queue manager. See 6.2.1, “Queue name resolution” on page 134 for more information.

It also passes in options to specify the actions to be performed by the application on that object.

Based on the information provided in the object descriptor and the identity context of the application, the queue manager determines whether the application is authorized to perform the requested action on the specified object. If it is authorized, the application is provided with an object handle (`Hobj`) that it must pass to future MQI calls it makes against that object.

Not all types of objects can be accessed from the MQI. Most commonly, the MQI is used to open queue objects to put and get messages. We discuss the types of queue objects hosted by a queue manager and how queues on remote queue manager can be accessed in 6.2, “Queues” on page 133.

The following possible actions can be requested in an MQOPEN call:

- Output: Put messages onto a queue using MQPUT calls. This is only available for queue objects.
- Browse: Retrieve messages from a queue using MQGET calls, leaving them available for other applications. This is only available for queue objects that resolve to queues hosted on the queue manager to which the application is connected.
- Input: Retrieve messages from a queue using MQGET calls, making them unavailable to other applications after they are retrieved. Input can be exclusive so that only one application can have the queue open for input at any given time. This is only available for queue objects that resolve to queues hosted on the queue manager to which the application is connected.
- Inquire: Get attribute values from the object using MQINQ calls.
- Set: Set attribute values for the object using MQSET calls.

6.1.5 MQPUT

The MQPUT MQI function is used to put messages onto queues opened for output.

Calls to MQPUT are returned after the message has been placed on a queue on the queue manager to which the application is connected. However, the queue object opened in the MQOPEN call might have represented a queue destination hosted on another, remote queue manager. In this case, the queue to which the message is put is a *transmission queue* on the local queue manager.

Message delivery from this transmission queue to the target queue manager, or another queue manager on route to the destination, is performed by message channels. We discuss message channels in 7.4, “Distributed message channels” on page 167.

A message descriptor structure (MQMD) is passed into the MQPUT call. During the MQPUT, call the queue manager generates information within this structure and returns it to the application within the same structure.

Note: If the same MQMD structure is used for multiple MQPUT calls, the fields generated by the queue manager must be reset between these calls. Specifically, the message identifier should be cleared, or all messages put with that MQMD will have the same message identifier.

A put message options structure (MQPMO) is passed into the MQPUT call to specify how the MQPUT call is performed, and some information is returned to the application within this structure.

A common usage of the MQPMO is to specify that the MQPUT should be performed under syncpoint. This means that the action of putting that message is contained within the current unit of work, and the message is not available for other applications to get or to be transferred to the target queue manager until an MQCMIT call is made.

6.1.6 MQPUT1

The MQPUT1 MQI function performs an MQOPEN, followed by an MQPUT, followed by an MQCLOSE. Enough information is provided to the call by an application to perform all three operations.

The MQPUT1 call is convenient, and efficient, when an application is putting a single message to a queue. This call is often used when putting a reply or a report message.

6.1.7 MQGET

The MQGET MQI function is used to get messages from queues opened for input or browse.

A message descriptor structure (MQMD) is passed into the MQGET call. The queue manager returns the message descriptor of a message that is successfully retrieved within this structure.

A get message options structure (MQGMO) is passed into the MQGET call to specify how the MQGET call is performed, and some information is returned to the application within this structure.

The MQGMO specifies whether the MQGET should *browse* messages, leaving them available for other applications, or *destructively get* messages, removing them from the queue.

An application might not be interested in all messages on a queue. For example, the application might only be interested in reply or report messages relating to datagrams or requests made by that application. To facilitate this, an application can specify that it is only interested in messages with a particular correlation identifier by placing that correlation identifier in the MQMD passed to the MQGET call. The application can control whether any matching is performed against the correlation identifier supplied in the MQMD using *match options* in the MQGMO. This is called *get by correlation identifier*.

An application can also *get by message identifier* using the same mechanism. However, this can be less efficient than the get by correlation identifier for queues containing many messages and is not generally used under normal operation. One use for get by message identifier is for administration tasks, such as browsing or destructively getting a particular message from a queue.

The MQGMO specifies whether the application should *wait* for matching messages to arrive on the queue, retrieving a matching message if it arrives within the specified period. An application can choose not to wait for message to arrive, to wait indefinitely for messages to arrive on a queue, or to wait for a wait interval specified in milliseconds. Multiple applications can wait for messages to arrive on a single queue. The order in which messages are retrieved from a queue when multiple applications are waiting for messages is not defined.

An application provides a buffer to the MQGET call and specifies the size of this buffer. If a message is too large to fit within this buffer, the application can use the MQGMO to specify whether the queue manager should return a partial message to the application, or fail the MQGET call.

The MQGMO specifies whether the MQGET should be performed under syncpoint. This means that the action of getting that message is contained within the current unit of work. The message remains on the queue, although unavailable for other applications to get, until an MQCMIT call is made.

Note: We recommend performing MQGET calls under syncpoint and using a subsequent MQCMIT call for applications getting persistent messages containing business-critical data over a client connection. This protects the message from loss due to communication failures between the application and the queue manager, which can occur while the queue manager is sending the message across the client connection to the application.

6.1.8 MQBEGIN

The MQBEGIN MQI function is only used when a queue manager has been configured to coordinate global units of work involving database products. When an application calls MQBEGIN, a new unit of work is started, which includes database updates to database products correctly configured to participate in WebSphere MQ coordinated units of work and MQPUT and MQGET calls made under syncpoint.

If an external transaction manager has been configured to coordinate units of work involving WebSphere MQ, a call must be made to that transaction manager by the application in order to begin a unit of work.

We discuss global units of work in 4.5.5, “Global units of work” on page 63.

6.1.9 MQCMIT and MQBACK

The MQCMIT MQI function is used to commit the current unit of work. For local units of work, those that only contain WebSphere MQ operations, this contains all MQPUT and MQGET calls made under syncpoint since the application connected or last performed an MQCMIT or MQBACK. For global units of work coordinated by WebSphere MQ, those including database operations and WebSphere MQ operations, this unit of work contains all MQPUT and MQGET calls made under syncpoint since the application last performed an MQBEGIN, MQCMIT, or MQBACK call.

The MQBACK MQI function is used to back out the current unit of work. The operations contained in the unit of work are the same as described for MQCMIT. Backing out a unit of work causes all operations to be undone, or *rolled back*. Messages destructively got within the unit of work are returned to their queue, and messages put within the unit of work are removed from their queue and never become available to other applications.

6.1.10 MQINQ and MQSET

The MQINQ MQI function is used to query the values of specified attributes from objects opened for inquire.

A set of *selectors* is passed into the MQINQ call. This set contains the names of each attribute for which values are requested.

A set of buffers are also provided to contain the values returned by the call. These buffers are split between buffers to store character string-based attributes, such as a description, and buffers to contain numeric integer-based attributes, such as the depth of a queue.

The MQSET MQI call is used to set the values of specified attributes of objects opened for set.

Selectors and buffers are passed to the MQSET call; the buffers specify the new values for the attributes when the call is issued.

6.1.11 MQDISC

An MQDISC should be performed corresponding to every MQCONN or MQCONNX call made by an application. If the application fails to perform an MQDISC call before exiting, or terminates abruptly, WebSphere MQ cleans up the connection when it detects that the application has terminated.

An MQCMIT call is attempted by the queue manager during an MQDISC call when WebSphere MQ is coordinating local or global units of work. If this MQCMIT call fails, the application is notified in the result of the MQDISC call. If an application does not perform an MQDISC call before exiting, the queue manager performs an MQBACK call when it detects that the application has terminated.

6.2 Queues

The term *queue* is used frequently in WebSphere MQ terminology. However, in different contexts, the term has different meanings, as follows:

- A queue object:
  Queue objects are objects defined within a queue manager and can be specified in the object name when performing an MQOPEN call on that queue manager. They are defined and administered using MQSC or the WebSphere MQ Explorer.

  There are a number of types of queue objects that can be defined. These can represent an actual queue that can contain messages, or a method for finding the destination of another queue within the WebSphere MQ infrastructure.

  All types of queue objects can be displayed in MQSC using the general type of `QUEUE`. Each individual queue object type is described in this section.
- A queue that can contain messages:
  Only one type of queue object actually represents a queue within the queue manager that can hold messages. This is a *local queue* object.

  Transmission queues, which are the intermediate queues between queue managers, are special cases of local queues.

  Local queues can be defined explicitly within a queue manager.

Local queues can also be dynamically defined from *model queue* objects, using an MQOPEN call against the model queue name. Local queues that have been defined dynamically are generally called *dynamic queues*. They are removed during an MQCLOSE call, and in the case of *temporary dynamic queues,* they can be removed automatically by the queue manager when applications disconnect.

- A cluster instance of a queue, known to the local queue manager: These are generally referred to as *cluster queues*. A cluster queue is not an actual queue object on the local queue manager. It is a representation, local to the queue manager, of an instance of a queue object that exists somewhere within a queue manager cluster. The actual queue object might be hosted on the local queue manager, or it might be hosted on another queue manager within the cluster. Multiple cluster queues can exist within a queue manager with the same name. We discuss queue manager clusters and cluster queues in more detail in Chapter 8, “Queue manager clusters” on page 181.

Note: All of the queue objects described in the rest of this section have specific uses within a queue manager cluster. These uses include when interconnecting queue manager clusters or interconnecting a queue manager cluster with queue manager infrastructures that do not use queue manager clusters.

However, when building a new WebSphere MQ infrastructure, using queue manager clusters can remove many requirements for these object types and thus simplify administration. See Chapter 8, “Queue manager clusters” on page 181 for more information about queue manager clusters.

6.2.1 Queue name resolution

When an application sends a message, the destination can be a queue on the same queue manager to which the application is connected. However, it can also be a destination on another queue manager within the WebSphere MQ infrastructure. The queue manager to which the application is connected must have knowledge of how to route messages to that remote queue manager. It must have a queue designated to temporarily store messages travelling to that destination, called a transmission queue.

There can be multiple intermediate queue managers between the queue manager to which an application is connected and the final destination. Each queue manager on route makes an independent decision regarding the next destination for a message on route to its final destination. This is based on the queue manager’s own knowledge of the infrastructure. This knowledge is contained in the WebSphere MQ objects defined on that queue manager and can be supplemented with knowledge received from other queue managers in a queue manager cluster.

The process of resolving destination information, as provided by an application, into the next destination for messages on route to their final destination is called *queue name resolution*. Queue name resolution occurs within a queue manager each time a message is received from an application or another queue manager.

Every time an MQOPEN call is performed to open a queue before putting messages, queue name resolution is performed by the queue manager. This takes the information provided in the `MQOD` structure on the MQOPEN call, the *object name*, and *object queue manager name*, and attempts to resolve this information into a valid destination queue name and queue manager name for the message.

After queue name resolution has been performed, an application can put messages to the resolved destination using MQPUT calls. This results in one of the following actions being performed by the queue manager:

- If queue name resolution has identified that the destination queue is local to the queue manager, the message are put directly into that queue.
- If queue name resolution has identified that the destination is on another queue manager, known to the current queue manager, the message is put on a *transmission queue* to be sent to that queue manager. This step requires that information is added to the message so that queue name resolution can be performed again when the message reaches the remote queue manager. This information is called a *transmission queue header*. See 7.4.1, "Message transmission" on page 167 for details about what happens to this message after it has been placed on the transmission queue.

Figure 6-1 on page 136 provides a graphical overview of queue name resolution. This chapter provides a description of each type of queue object, represented by the icons within the queue name resolution box in Figure 6-1 on page 136. These are the same icons used to represent each object type in the WebSphere MQ Explorer.

Different types of objects affect queue name resolution in a different way. This is illustrated with similar figures for each type of object in the following sections. These figures show a circumstance in which the destination of a message is affected by defining an object of that type on the queue manager to which an application is connected.

Note: We discuss how queue name resolution is affected by being a member of a queue manager cluster in 8.1.7, "Sharing queue objects within clusters" on page 189.

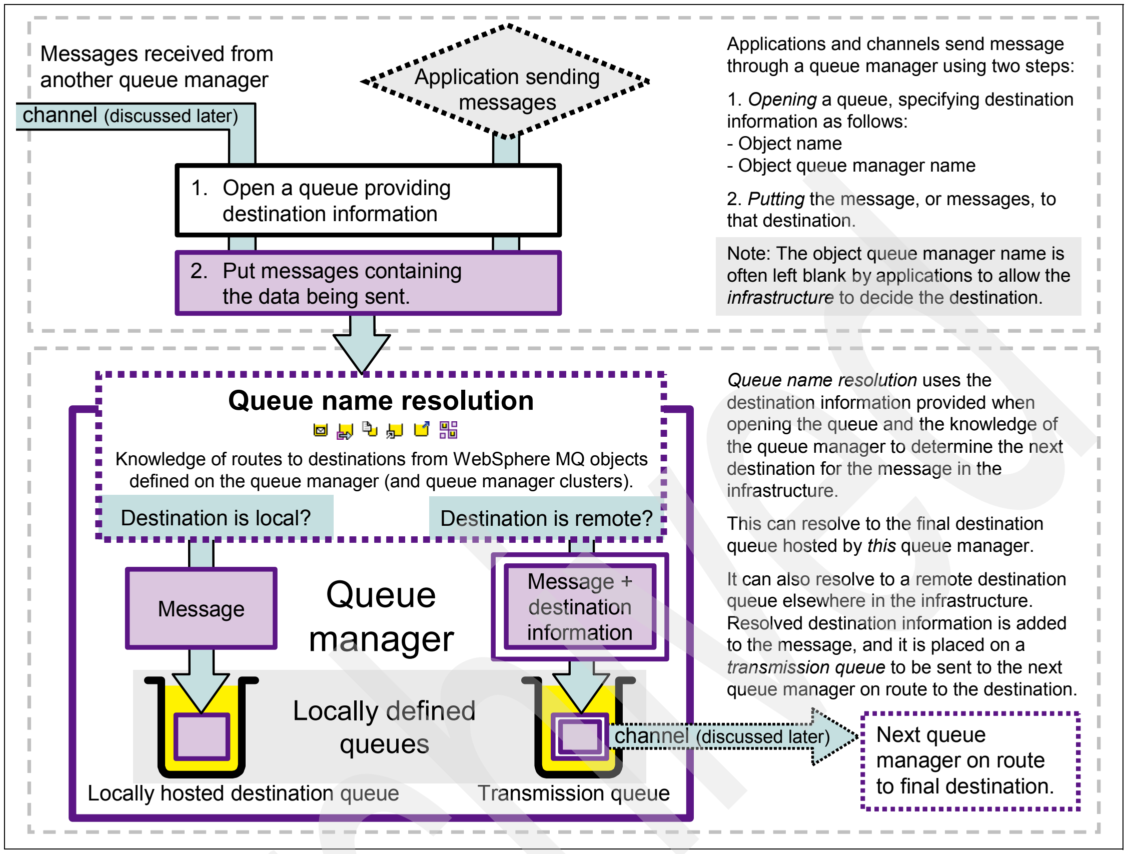

Messages received from another queue manager
channel (discussed later)
Application sending messages
1. Open a queue providing destination information
2. Put messages containing the data being sent.
Applications and channels send message through a queue manager using two steps:
1. Opening a queue, specifying destination information as follows:
- Object name
- Object queue manager name
2. Putting the message, or messages, to that destination.
Note: The object queue manager name is often left blank by applications to allow the infrastructure to decide the destination.
Queue name resolution
Knowledge of routes to destinations from WebSphere MQ objects defined on the queue manager (and queue manager clusters).
Destination is local?
Destination is remote?
Message
Queue manager
Message + destination information
Locally defined queues
channel (discussed later)
Next queue manager on route to final destination.
Locally hosted destination queue
Transmission queue
Queue name resolution uses the destination information provided when opening the queue and the knowledge of the queue manager to determine the next destination for the message in the infrastructure.
This can resolve to the final destination queue hosted by this queue manager.
It can also resolve to a remote destination queue elsewhere in the infrastructure. Resolved destination information is added to the message, and it is placed on a transmission queue to be sent to the next queue manager on route to the destination.

Figure 6-1 Queue name resolution

6.2.2 Local queue objects and transmission queues

Local queue objects are the only type of queue object that represent a real queue that holds messages. Local queue objects can be manually defined as described in this section or created dynamically as described in 6.2.4, “Model queue objects and dynamic creation of local queues” on page 141.

Because local queue objects are the only type that hold messages, they have a wide variety of uses. This book generally refers to local queue objects as local queues, because they represent a real queue.

A simple example of using a local queue is where multiple applications communicate asynchronously on the same machine. Applications put messages to and get messages from that same local queue.

All other queue object types can be considered as methods for creating local queues, resolving the name and location of local queues, or routing messages between local queues.

Figure 6-2 shows an application putting a message to a local queue. In this case, the queue name resolution simply resolves the *object name* specified to the local queue using the local queue object defined.

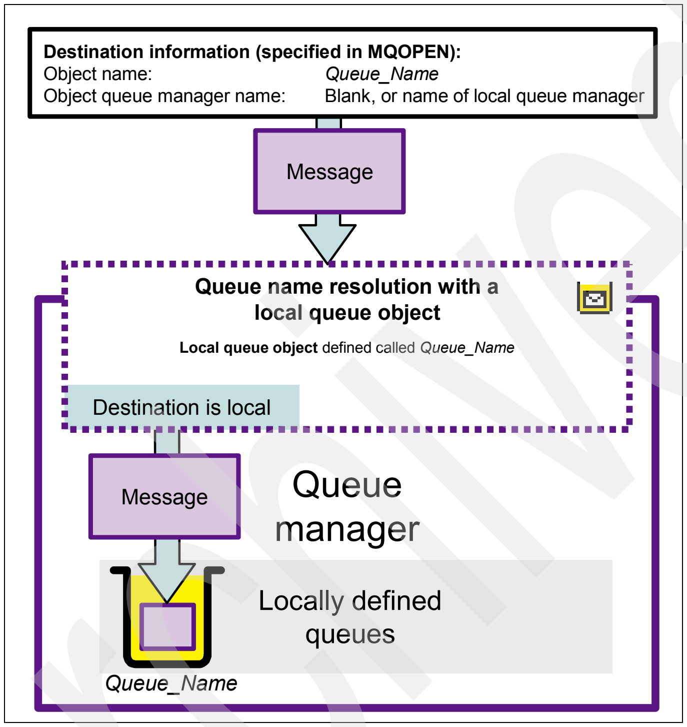

Destination information (specified in MQOPEN):
Object name: Queue_Name
Object queue manager name: Blank, or name of local queue manager
Message
Queue name resolution with a local queue object
Local queue object defined called Queue_Name
Destination is local
Message
Queue manager
Locally defined queues
Queue_Name

Figure 6-2 Queue name resolution with a local queue object

Transmission queues

The `USAGE` attribute of a local queue object can be used to designate a local queue as a transmission queue. To do this, specify the usage attribute of the local queue as transmission (`XMITQ`).

If a local queue is designated as a transmission queue, applications should not attempt to put messages to this queue directly.

The transmission of messages from a transmission queue to a remote queue manager is performed by a message channel. We discuss message channels in 7.4.1, “Message transmission” on page 167.

Defining a transmission queue provides a queue manager with knowledge of how to route messages to a single destination queue manager. Any messages sent with an *object queue manager name* the same as the name of the transmission queue are placed on that transmission queue. For this reason, the name of the transmission queue and the name of the remote queue manager should generally match. However, the remote queue objects described in 6.2.5, “Remote queue objects” on page 143 allow a route to be explicitly defined where the queue manager name does not match the transmission queue name.

Note: Reply and report messages are sent using the `ReplyToQMgr` field in the message descriptor of the original request or datagram message. If the queue manager does not have knowledge of how to route messages to the reply-to queue manager, the reply cannot be sent.

Defining a transmission queue with the *same name* as the reply-to queue manager provides a queue manager with this knowledge.

Figure 6-3 on page 139 shows an application sending a message and specifying an *object queue manager name*. The message is placed on the transmission queue, because the *object queue manager name* and the name of the transmission queue match.

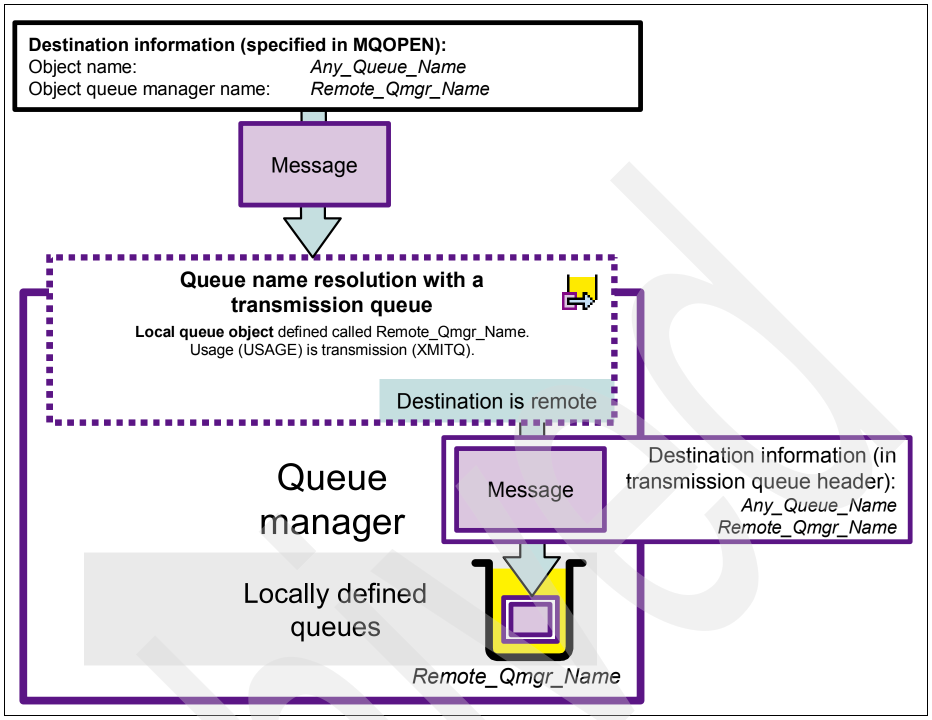

Destination information (specified in MQOPEN):
Object name: Any_Queue_Name
Object queue manager name: Remote_Qmgr_Name
Message
Queue name resolution with a transmission queue
Local queue object defined called Remote_Qmgr_Name.
Usage (USAGE) is transmission (XMITQ).
Destination is remote
Queue manager
Message
Destination information (in transmission queue header):
Any_Queue_Name
Remote_Qmgr_Name
Locally defined queues
Remote_Qmgr_Name

Figure 6-3 Queue name resolution with a transmission queue

Manually defining local queue objects

The *definition type* (DEFTYPE) attribute of a local queue distinguishes between local queues that have been manually defined and those that have been dynamically created from model queue objects. For manually defined local queues, the definition type (DEFTYPE) attribute is set to predefined (PREDEFINED).

Note: The definition type (DEFTYPE) attribute does not distinguish between those local queues created automatically by WebSphere MQ when a queue manager is created and those defined by an administrator.

You can manually define local queues using one of the following methods:

- Using the DEFINE QLOCAL MQSC command.
- Using the WebSphere MQ Explorer:
  a. Right-click the **Queues** folder under a particular queue manager in the navigator view of the WebSphere MQ Explorer.
  b. Select **New** → **Local Queue**.

c. Follow the instructions provided in the New Local Queue wizard.

6.2.3 Alias queue objects

An alias queue object is a representation of another *target* queue object, which has a different name. An alias queue can be accessed in the same way as the target queue object of which it is an alias. References to it are redirected to the target queue object specified as part of the alias queue definition.

The *target queue* (`TARGQ`), or *base queue*, attribute of an alias queue object specifies the name of the target queue object.

The target queue object specified must be one of the following types:

- A local queue defined on the same queue manager as the alias queue object.
- A remote queue object defined on the same queue manager as the alias queue object.
- An instance of a queue object shared within a queue manager cluster of which the queue manager is a member. We discuss queue manager clusters in Chapter 8, "Queue manager clusters" on page 181.

Note: WebSphere MQ does not enforce that the specified target queue when defining or altering an alias queue is the name of a valid queue object. Attempts to perform an **`MQOPEN`** call for that alias queue, or routing of messages through that alias queue, fail if the target queue is not valid.

Figure 6-4 on page 141 shows an application putting a message to a local queue through an alias queue object. The *object name* requested by the application is resolved into the target local queue object, as specified in the alias queue definition.

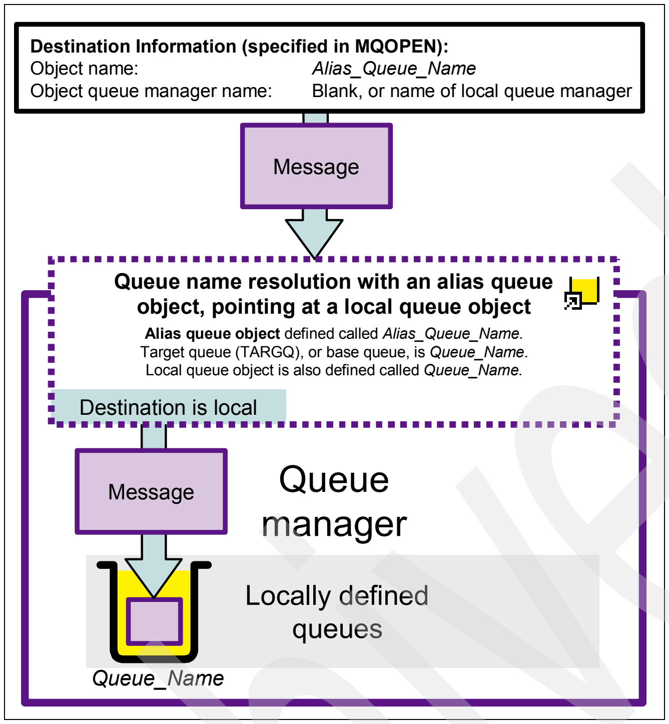

Destination Information (specified in MQOPEN):
Object name: Alias_Queue_Name
Object queue manager name: Blank, or name of local queue manager
Message
Queue name resolution with an alias queue object, pointing at a local queue object
Alias queue object defined called Alias_Queue_Name.
Target queue (TARGQ), or base queue, is Queue_Name.
Local queue object is also defined called Queue_Name.
Destination is local
Message
Queue manager
Locally defined queues
Queue_Name

Figure 6-4 Queue name resolution with an alias queue object, pointing at a local queue

Use one of the following methods to define alias queue objects:

- Using the DEFINE QALIAS MQSC command.
- Using the WebSphere MQ Explorer:
  a. Right-click the **Queues** folder under a particular queue manager in the navigator view of the WebSphere MQ Explorer.
  b. Select **New** → **Alias Queue**.
  c. Follow the instructions provided in the New Alias Queue wizard.

6.2.4 Model queue objects and dynamic creation of local queues

Model queue objects provide the attributes of a local queue object that can be created dynamically by an application. Dynamically created queues are instances of local queues and can hold messages. This book generally refers to a dynamically created instance of a local queue as a *dynamic queue*.

Dynamic queues can be used to give an application a temporary identity within the WebSphere MQ message queuing infrastructure. The most common use is in request/reply messaging to provide an individual reply-to queue for an application. We discuss this in 4.6.14, “Reply-to queue considerations” on page 75.

Simply by specifying the name of a model queue object in the *object name* of an `MQOD` passed to an MQOPEN call, an application can dynamically create a local queue. The name of the local queue that is created is *not related* to the name of the model queue object.

The name of a dynamic queue can be chosen by the application that performs the MQOPEN call using the dynamic queue name (`DynamicQName`) field in the `MQOD`.

A wildcard can be placed at the *end* of the dynamic queue name field in an `MQOD`. This causes the queue manager to generate the rest of the dynamic queue name. This name is unique within the queue manager. To assure uniqueness within the 48 characters available for a queue object name, a wildcard cannot be placed after character 33 of the dynamic queue name field.

The default value of the dynamic queue name field in an `MQOD` is as follows:

- WebSphere MQ for z/OS:

  `CSQ.*`

- WebSphere MQ for all other platforms:

  `AMQ.*`

After it is created, the dynamic queue can be accessed in the same way as a manually defined local queue. Applications must send messages to the queue using its dynamically created name, not the name of the model queue object.

There are two distinct types of dynamic queue. The definition type (`DEFTYPE`) attribute of a model queue object specifies the type of the dynamic queue that is created by opening that model queue object. The type of a dynamic queue defines how it can be used and when it is deleted by the queue manager. The available definition types are as follows:

- Temporary dynamic (`TEMPDYN`):
  The dynamic queue created, called a t*emporary dynamic local queue*, can only contain nonpersistent messages. This is because a queue manager can automatically delete a temporary dynamic local queue. When this occurs, any messages contained on the queue are lost. A temporary dynamic local queue is deleted by the queue manager in the following circumstances:

  – When the application performs an MQCLOSE specifying the object handle (`Hobj`) returned from the MQOPEN call that created the queue.

- When the application disconnects from a queue manager by performing an MQDISC call.
- After the queue manager has detected termination of the application if the application ends without performing an MQDISC call.
- If the queue manager is restarted.

- Permanent dynamic (`PERMDYN`):
  The dynamic queue created, called a *permanent dynamic local queue*, can contain persistent or nonpersistent messages. Permanent dynamic local queues are not deleted automatically by a queue manager. Permanent dynamic local queues can be manually deleted as follows:
  - An application, not necessarily the application that performed the MQOPEN call that originally created the queue, performs an MQCLOSE call specifying the *delete* (`MQCO_DELETE`) option. This call is only successfully completed if the queue is empty.
  - An application, not necessarily the application that performed the MQOPEN call that originally created the queue, performs an MQCLOSE call specifying the *delete-purge* (`MQCO_DELETE_PURGE`) option. This succeeds if the queue contains messages. However, it fails if any messages on the queue have been put or got within a unit of work, and that unit of work has not yet been committed or backed out; these are uncommitted messages.
  - By deleting the queue object using an administration interface, such as the WebSphere MQ Explorer or MQSC.

Use one of the following methods to define model queue objects:

- Using the DEFINE QMODEL MQSC command.
- Using the WebSphere MQ Explorer:
  a. Right-click the **Queues** folder under a particular queue manager in the navigator view of the WebSphere MQ Explorer.
  b. Select **New** → **Model Queue**.
  c. Follow the instructions provided in the New Model Queue wizard.

6.2.5 Remote queue objects

Remote queue objects are used to define routes to other queue managers within the WebSphere MQ message queuing infrastructure. This involves mapping queue manager names to transmission queues, and mapping queue names to different queue names on remote queue managers.

A remote queue object can be used in three ways:

- As a local definition of a remote queue:
  This enables an application to open a remote queue object to put messages without specifying an explicit remote queue manager name in the same way as though it was a local queue. This can be used to place a queue on a remote queue manager without making any changes to an application that was originally written to access a local queue. To create a local definition of a remote queue, create a remote queue object with attributes as follows:

  – The name of the remote queue object:
    The name of the queue to which applications connected to the queue manager put messages.

  – Remote name (RNAME):
    The name of the queue on the destination queue manager. It can be the same as the name of the remote queue object. Queue name resolution is performed on the remote machine using this queue name.

  – Remote queue manager name (RQMNAME):
    The name of the destination queue manager, not necessarily the next queue manager on route. Queue name resolution is performed on the remote machine using this queue manager name.

  – Transmission queue (XMITQ):
    This attribute specifies the name of the transmission queue that takes the message to the next queue manager on route to the destination. If the remote queue manager name is the same as the transmission queue name, this attribute can be left blank.

  Figure 6-5 on page 145 shows an application sending a message through a local definition of a remote queue. The *object name* specified by the application is the name of the remote queue object. Queue name resolution places this message on the transmission queue specified in the remote queue object definition. A transmission queue header is added to the message, which contains the *remote name* and r*emote queue manager name* attributes from the remote queue object definition. When the message reaches its destination, the information in the transmission queue header is used to perform queue name resolution on that remote queue manager.

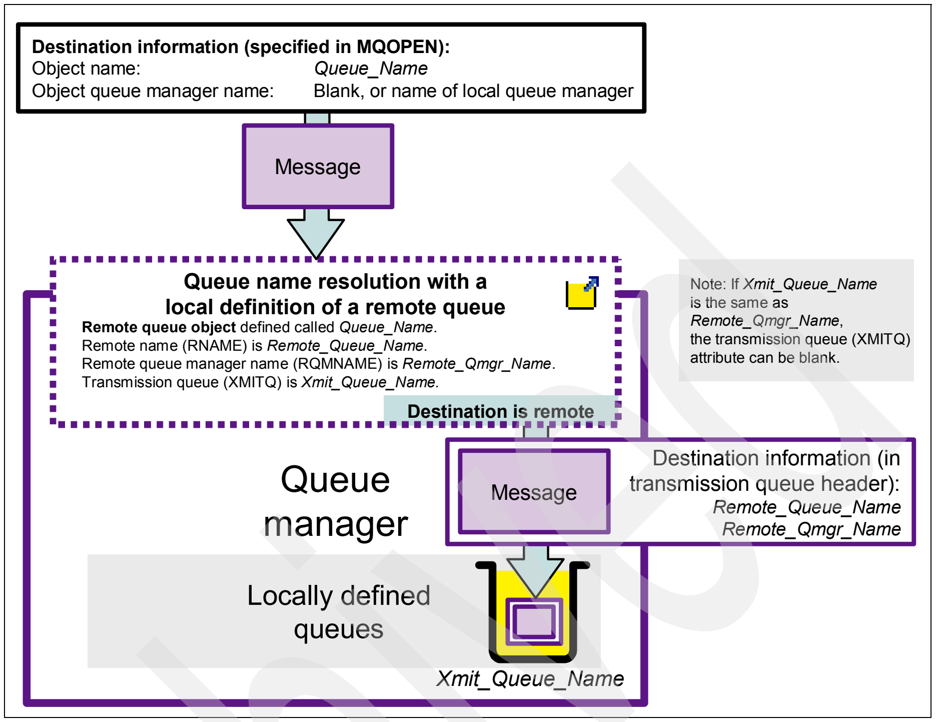
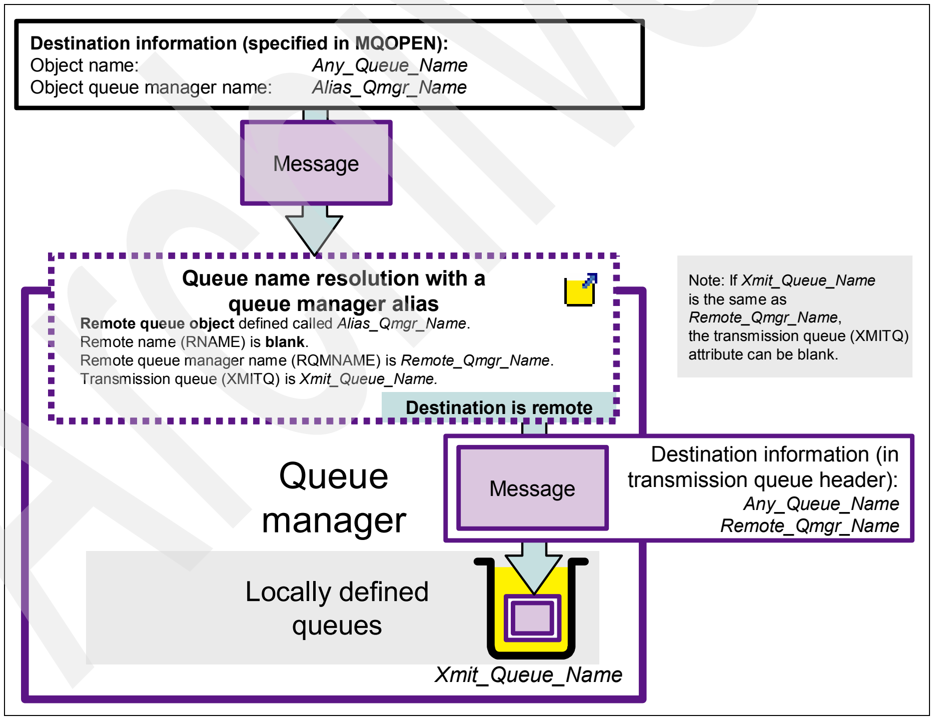

Destination information (specified in MQOPEN):
Object name: Queue_Name
Object queue manager name: Blank, or name of local queue manager
Message
Queue name resolution with a local definition of a remote queue
Remote queue object defined called Queue_Name.
Remote name (RNAME) is Remote_Queue_Name.
Remote queue manager name (RQMNAME) is Remote_Qmgr_Name.
Transmission queue (XMITQ) is Xmit_Queue_Name.
Destination is remote
Note: If Xmit_Queue_Name is the same as Remote_Qmgr_Name, the transmission queue (XMITQ) attribute can be blank.
Queue manager
Message
Destination information (in transmission queue header):
Remote_Queue_Name
Remote_Qmgr_Name
Locally defined queues
Xmit_Queue_Name

Figure 6-5 Queue name resolution with a local definition of a remote queue

- As a queue manager alias:
  These are used to provide a queue manager with knowledge of a route to a destination queue manager. This includes when the name of the destination queue manager is not the same as the transmission queue used to route messages to that queue manager. Queue manager aliases can also be used to map one queue manager name to another. This includes mapping to a blank queue manager name to designate the local queue manager, or that workload balancing needs to be performed when using queue manager clusters.
  - The name of the remote queue object:
    This is the name of the queue manager that is being aliased. Queue name resolution uses this object, where the *object queue manager name* specified matches the name of the remote queue object.
  - Remote name (`RNAME`):
    This must be blank for a queue manager alias.

- Remote queue manager name (`RQMNAME`):
  This is the name of the destination queue manager, not necessarily the next queue manager on route. After a message has been routed to the next queue manager on route using the specified transmission queue, this queue manager name is used for queue name resolution. It can be left blank to designate that queue name resolution needs to be performed as if no queue manager name had been specified.
- Transmission queue (`XMITQ`):
  This attribute specifies the name of the transmission queue that takes the message to the next queue manager on route to the destination. If the remote queue manager name is the same as the transmission queue name, this attribute can be left blank. It is also left blank when defining an alias for the local queue manager.

Figure 6-6 shows an application sending a message through a queue manager alias. The *object queue manager name* specified by the application is mapped to a different queue manager name, and the message is placed on a transmission queue to be delivered to that queue manager.



Destination information (specified in MQOPEN):
Object name: Any_Queue_Name
Object queue manager name: Alias_Qmgr_Name
Message
Queue name resolution with a queue manager alias
Remote queue object defined called Alias_Qmgr_Name.
Remote name (RNAME) is blank.
Remote queue manager name (RQMNAME) is Remote_Qmgr_Name.
Transmission queue (XMITQ) is Xmit_Queue_Name.
Destination is remote
Note: If Xmit_Queue_Name is the same as Remote_Qmgr_Name, the transmission queue (XMITQ) attribute can be blank.
Queue manager
Message
Destination information (in transmission queue header):
Any_Queue_Name
Remote_Qmgr_Name
Locally defined queues
Xmit_Queue_Name

Figure 6-6 Queue name resolution with a queue manager alias

- As a reply-to queue alias:
  These can be used to customize the routes that replies take back to a queue manager through the infrastructure. These affect the reply-to queue manager name in the *message descriptor* when an application puts a message, rather than affecting queue name resolution when opening a queue. The use of reply-to queue aliases is beyond the scope of this book. Refer to *WebSphere MQ Intercommunication*, SC34-6587 for more information.

Use one of the following methods to define remote queue objects:

- Using the DEFINE QREMOTE MQSC command.
- Using the WebSphere MQ Explorer:
  a. Right-click the **Queues** folder under a particular queue manager in the navigator view of the WebSphere MQ Explorer.
  b. Select **New** → **Remote Queue**.
  c. Follow the instructions provided in the New Remote Queue wizard.

6.2.6 Default attributes and authority checks

During the process of opening a queue, WebSphere MQ performs authority checks to determine whether an entity is authorized to perform the requested actions. We discuss this in 11.2, "Granting access to queue manager resources" on page 303.

WebSphere MQ also sets up some defaults for messages that are subsequently sent by the application opening the queue. The application can choose to override these defaults.

These authority checks and defaults are based on the attributes and permissions of *one* WebSphere MQ object. This can be considered as the first WebSphere MQ object accessed while resolving the queue name specified by the application.

When sending messages, the object used depends on the combination of *object name* and *object queue manager name* specified by the application sending the message. Table 6-1 on page 148 summarizes the possible combinations. This includes those relating to queue manager clusters. We discuss queue manager clusters in Chapter 8, "Queue manager clusters" on page 181.

Table 6-1 WebSphere MQ object used for authority checks and defaults

Object name	Object queue manager name	WebSphere MQ object used for authority checks and defaults
The name of a local, alias, model, or remote queue object defined on the local queue manager	Blank, or the name of the local queue manager	The WebSphere MQ object with the specified name.
Anything	The name of a remote queue manager that can be resolved to a transmission queue	The transmission queue.
The name of a queue shared within a queue manager cluster of which the queue manager is a member	Blank	Authority checks are performed based on the `SYSTEM.CLUSTER.TRANSMIT.QUEUE`. The defaults are taken from the definition of the queue object on the *remote* queue manager. The defaults can be seen by displaying the attributes of the cluster queue record, for example, using DISPLAY QCLUSTER in MQSC.

Default persistence

A commonly used queue attribute is the default persistence (`DEFPSIST`) attribute. When messages are put, an application can choose whether a message is persistent or not as an option to the MQPUT call. Alternatively, the application can allow the default persistence attribute of the queue object to define whether messages are persistent or not.

For example, a queue manager has a local queue called `local.psist` defined with `DEFPSIST(YES)`. This same queue manager has an alias queue defined called `alias.nonpsist` with `DEFPSIST(NO)` and `TARGQ('local.psist')`. A message put after opening `local.psist` is persistent. A message put having opened `alias.nonpsist` is nonpersistent. However, both are placed on the same queue.

Note: This is an example, rather than a recommended approach. WebSphere MQ is most efficient when all messages on a queue are either persistent or nonpersistent.

6.2.7 Queue status and online monitoring for queues

Local queues are the most important resource hosted by a queue manager, because they contain all of the messages that reside within that queue manager.

When monitoring the operation of the applications accessing a queue manager, or when diagnosing problems with those applications, an administrator is likely to require knowledge of how applications are accessing particular queues. Information about the flow of messages through particular queues can also be useful to identify and resolve problems or monitor performance when provisioning resources.

For this purpose, WebSphere MQ provides monitoring information about the status of each local queue hosted by a queue manager. A summary of this information, which has been significantly enhanced in WebSphere MQ V6.0, is as follows:

- The number of messages on the queue.
- The number of applications with the queue open for input or browse.
- The last time and date an application got a message from the queue.
- The number of applications with the queue open for output.
- The last time and date an application put a message to the queue.
- The number of uncommitted messages on the queue. These are messages that have been put to or got from a queue within a unit of work that is not yet complete.
- Information about each application connection to the queue manager that has the queue open. This includes the following information:
  - The name of the application that created that connection
  - The process identifier of the application
  - The identity context under which the application is running
  - The actions for which the application opened the queue, such as browse, input, output, or inquire
  - If the connection is remote to the queue manager using a client connection, the information is provided about that connection
  - Whether the application is actively performing work against the queue manager

- Information regarding the performance of the applications using the queue. This includes the following information:
  - Short-term and longer-term estimated averages of the time each message spends on the queue before being retrieved. These are provided in microseconds, because a message might spend a very short amount of time on a queue.
  - The largest amount of time that any message currently on the queue has spent on that queue. This is provided in seconds. For queues where messages flow through this queue quickly, a large value can represent a problem with the processing of one of the messages on the queue.

> **Note:** This performance-related queue monitoring information is new for WebSphere MQ V6.0. This information is called *online monitoring* information. It can be enabled for all queues on a queue manager using the `MONQ` attribute on the queue manager object, or for specific local queues using the `MONQ` attribute on the queue. See *Monitoring WebSphere MQ*, SC34-6593, for more information. Alternatively, press F1 with the Online monitoring section of the properties page for a queue manager highlighted in the WebSphere MQ Explorer.

Use one of the following methods to view queue status information:

- Using the DISPLAY QSTATUS MQSC command.
- Using the WebSphere MQ Explorer. This is generally the most convenient way to view queue status information.
  a. Click the **Queues** folder under a particular queue manager in the navigator view of the WebSphere MQ Explorer.
  b. Right-click the local queue of interest in the table shown on the Queues content page.
  c. Select **Status**.

6.3 Triggering

Messages arriving on queues represent events within the system. In most cases, processing of that message is required.

WebSphere MQ provides *triggering* to allow this processing to automatically be *initiated* by a queue manager. This processing can represent starting a channel to move messages from a transmission queue to a remote queue manager, or starting an application instance to perform an action on a message or a batch of messages.

Depending on the use of a queue, a message arriving on that queue might, or might not, represent an event for which processing must be performed. The queue can be configured to generate *trigger events* that match the usage of that queue.

6.3.1 Generation of trigger events

A queue can be configured in the following ways to generate trigger events:

- Every:
  A trigger message is generated for every message that arrives on the queue. To enable this form of triggering on a local queue, specify the following attributes:
  - Trigger control to on: `TRIGGER`
  - Trigger type to every: `TRIGTYPE(EVERY)`
- First:
  After a queue becomes empty, the first message that arrives on that queue generates a trigger event. A trigger event is usually generated if any application has the queue open for input to retrieve messages. However, if a message arrives after a specified trigger interval, and no application has the queue open for input, another trigger event occurs. To enable this form of triggering on a local queue, specify the following attributes:
  - Trigger control to on: `TRIGGER`
  - Trigger type to first: `TRIGTYPE(FIRST)`
  - Trigger interval, which is a queue manager-wide attribute on the queue manager object, to the required number of milliseconds: `TRIGINT(`*`5000`*`)`
- Depth:
  A message arriving on a queue, which brings the total number of messages on a queue above a certain threshold, generates a trigger event. A trigger event is not generated if any application has the queue open for input to retrieve messages. This form of triggering is configured by specifying the following attributes on a local queue:
  - Trigger control to on: `TRIGGER`
  - Trigger type to first: `TRIGTYPE(DEPTH)`
  - Trigger depth to the threshold required: `TRIGDPTH(`*`10`*`)`

Note: There are a number of conditions that must be met for a trigger event to occur. If a trigger event is not generated when expected, we recommend that you check each condition in turn to diagnose the problem. The "Starting WebSphere MQ applications using triggers" section in *WebSphere MQ Application Programming Guide*, SC34-6595, provides the full list of conditions.

6.3.2 Initiation queues and trigger messages

When a trigger event occurs, a message is placed on an *initiation queue*. This message is called a *trigger message*. The name of this queue onto which the trigger message is place is specified in the initiation queue (`INITQ`) attribute of a local queue object.

Any local queue can be designated as an initiation queue. No special configuration needs to be performed on a local queue in order for it to be designated as an initiation queue, but it should not be configured as a transmission queue.

Each trigger message contains information about the action that needs to be performed in response to the trigger event. The message contains the following information:

- A queue name:
  This is the name of the queue from which the event was generated.
- Details of an application to execute:
  Details of how to execute an application are platform specific. WebSphere MQ provides the `PROCESS` object type. This has attributes that provide enough information to execute a particular application in the operating system. If the `PROCESS` attribute of the queue that generated the event contains the name of a valid `PROCESS` object, the trigger message contains all attributes from that object.
- Trigger data:
  This is custom data from the trigger data (`TRIGDATA`) attribute of the queue that generated the event.

6.3.3 Trigger monitors

An application that waits for messages to arrive on a local queue designated as an initiation queue is referred to as a trigger monitor.

Trigger monitors provided by WebSphere MQ

WebSphere MQ provides trigger monitors for each platform. These are provided to perform the core functionality of executing applications based on the `PROCESS` definition each time a trigger event is generated. The details of the trigger event are passed to the application that is invoked.

Refer to the “Writing WebSphere MQ applications” section in *WebSphere MQ Application Programming Guide*, SC34-6595, for further details.

The WebSphere MQ distributed channel initiator

The WebSphere MQ distributed channel initiator is a WebSphere MQ process started automatically with the queue manager by default. It is a special trigger monitor that starts message channels from the names of channels specified in the trigger data attribute of transmission queues configured for triggering. See 7.4.13, "Channel initiation" on page 178 for details.

Note: Do not confuse the channel initiator described in this section with the WebSphere MQ for z/OS channel initiator described in 5.3.10, "WebSphere MQ for z/OS channel initiator" on page 115.

WebSphere MQ for z/OS and WebSphere MQ for iSeries provide channel listener programs that provide the functionality for starting channels.

7

Queue manager intercommunication and client connections in WebSphere MQ

This chapter describes how intercommunication to and from a queue manager occurs, including applications connecting remotely as clients to a queue manager communicating between queue managers over message channels.

We discuss the following topics:

- Channels
- Starting and stopping channels
- Client channels
- Distributed message channels
- Channel auto-definition

7.1 Channels

All network communication in WebSphere MQ is performed across a *channel*. As with the term queue, the term channel is used regularly within WebSphere MQ terminology and can have different meanings in different contexts. We summarize these meanings here:

- An established network communication link between two queue managers, or between a client application and a queue manager:
  After a WebSphere MQ network communication link has been established through which messages can flow or MQI commands can be issued, it is referred to as a *channel*.

  A channel between two queue managers over which messages can flow is referred to as a *message channel*.

  A channel between a client application and a queue manager over which MQI calls are issued is referred to as a *client channel*, or *MQI channel*.
- A channel object:
  Channel objects are objects defined within a queue manager. Each channel object has a name and a *channel type*.

  The attributes of a channel object define how communication is performed. For example, these attributes can specify whether Secure Sockets Layer (SSL) authentication is required when establishing a channel.

  Some channel object types are used to define how message channels can be established for that queue manager. Other channel object types are used to define how message channels can be established to other queue managers in the infrastructure.

  Some channel object types are used to join a queue manager to a queue manager cluster. After the channel objects required to join a queue manager cluster have been defined, message channels are established automatically to and from the other queue managers in the queue manager cluster.

  Other channel object types are used to define how applications can connect directly to a queue manager over a network.
- A message channel agent (MCA):
  Every channel in WebSphere MQ is a network link between two message channel agents (MCAs).

  Each connection established or attempted to or from a queue manager is hosted by a message channel agent.

  A connection performed by an application connecting to a queue manager is also performed by an MCA, even though it is not performed from within a queue manager.

7.1.1 Introduction to client channels

When using a client channel to connect to a queue manager, the MCA is invoked to communicate with the remote queue manager whenever a WebSphere MQ function call is issued by the application. This MCA is provided by the client API being used to connect to the queue manager.

This client API can be part of the core WebSphere MQ client product. However, it can also be the Java Message Service (JMS) client that is provided with the WebSphere Application Server product, or another client product such as provided in the Extended Message Service (XMS).

A client channel passes MQI commands to a queue manager. However, the interface into the client MCA can be through an object-oriented interface, such as the WebSphere MQ C++ or Java APIs, or a standardized interface, such as the JMS API or the XMS API. Function calls within these APIs that send or receive messages through the WebSphere MQ infrastructure perform the MQI commands within the MCA.

7.1.2 Message channel agents (MCAs)

An MCA establishes a channel with a partner MCA hosted by a queue manager through the listener provided by that queue manager.

The name and attributes of both MCAs are usually taken from channel objects defined on a queue manager. However, applications using client channels can manually specify the name and attributes of their MCA.

Channel auto-definition can occur on a queue manager to allow a queue manager to receive a connection and start an MCA where no channel object of the correct name exists on the queue manager. A channel object of the correct name and type is automatically defined on that queue manager and remains after the channel has disconnected. We discuss channel auto-definition in 7.5, “Channel auto-definition” on page 179.

In order to establish a channel, the two MCAs *negotiate*, or *bind*. The following list describes of some steps performed between the two MCAs during this negotiation stage:

- The MCA that decides whether to accept a connection must ensure that a channel object is known to the queue manager of a correct name and type to accept a connection from the remote MCA. Channel auto-definition can be used to create this object, as described in 7.5, “Channel auto-definition” on page 179. The attributes of this object affect the negotiation.

- Channel objects can be disabled. If either channel object is disabled, the channel does not start. See 7.2.1, “Understanding channel status” on page 160 for a description of how a channel is disabled.
- An SSL handshake might be performed depending on the configuration of the two MCAs. The MCA accepting the connection or both of the MCAs provide a certificate for validation by the partner during this handshake. An MCA can be configured to only accept a connection from an entity with a certain distinguished name within its validated certificate.
- An identity context is established for the partner MCA. This is most significant for client channels, where this identity context is used to authenticate each MQI call. This can be the user identifier the client application (or partner MCA) is running under, or forced to a particular user identifier by the queue manager based on a channel attribute.
- Some attributes of the channel must be negotiated between the two MCAs in order to find a value acceptable to both MCAs. The batch size is one such attribute, as described in 7.4.2, “Batches” on page 170. These attributes are usually numeric and generally take the lower value of the two MCAs.

Figure 7-1 summarizes how two MCAs establish a channel.

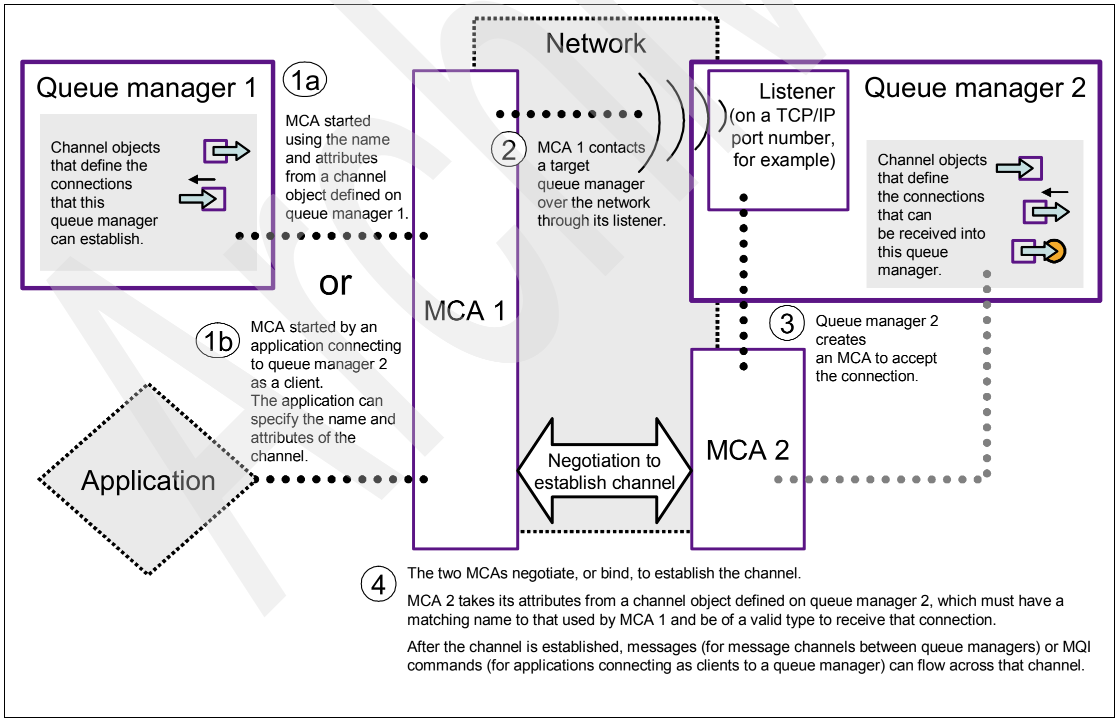

Network
Queue manager 1
Channel objects that define the connections that this queue manager can establish.
1a
MCA started using the name and attributes from a channel object defined on queue manager 1.
or
1b
MCA started by an application connecting to queue manager 2 as a client. The application can specify the name and attributes of the channel.
Application
MCA 1
2
MCA 1 contacts a target queue manager over the network through its listener.
Listener (on a TCP/IP port number, for example)
Queue manager 2
Channel objects that define the connections that can be received into this queue manager.
3
Queue manager 2 creates an MCA to accept the connection.
MCA 2
Negotiation to establish channel
4
The two MCAs negotiate, or bind, to establish the channel.
MCA 2 takes its attributes from a channel object defined on queue manager 2, which must have a matching name to that used by MCA 1 and be of a valid type to receive that connection.
After the channel is established, messages (for message channels between queue managers) or MQI commands (for applications connecting as clients to a queue manager) can flow across that channel.

Figure 7-1 Establishing a channel between queue managers or from an application to a queue manager

7.2 Starting and stopping channels

Starting a channel represents initiating an MCA to connect to an MCA on a remote queue manager and establish a channel.

Stopping a channel represents ending communication between two MCAs that have established a channel.

Client channels are started when an application connects to a queue manager and remain active until that application disconnects.

The START CHANNEL and STOP CHANNEL MQSC commands can be used to start and stop message channels between queue managers. They are also used to enable and disable channel objects to control whether channels can be started using those channel objects.

The same functionality is provided by the WebSphere MQ Explorer by selecting the **Channels** folder under a queue manager in the navigator view, right-clicking a channel object, and selecting **Start** or **Stop**.

Issuing a start command against a message channel object, which is of a type that establishes a channel from a queue manager to another queue manager, causes that channel to become active and begin transferring messages between the transmission queue of one queue manager and queues on the other queue manager.

However, message channels can also be started automatically when messages arrive on a transmission queue using a *channel initiator*. Message channels in a queue manager cluster are started automatically by a queue manager by a channel initiator when required.

Issuing a stop command against a channel object has two functions:

- To end all channels associated with that channel object, allowing them to be restarted by a client application, channel negotiator, or queue manager cluster when next required.

  In MQSC, this action is performed using the `MODE(INACTIVE)` attribute on the STOP CHANNEL command. In the WebSphere MQ Explorer, this action is performed by selecting **Inactive** from the New state drop-down list on the Stop Channel window.

- To disable that channel object so that no channels can be established that use that channel object until the channel is enabled and manually started again using a start command.

  In MQSC, this action is performed using the `MODE(STOPPED)` attribute on the STOP CHANNEL command. In the WebSphere MQ Explorer, this action is performed by selecting **Stopped** from the New state drop-down list on the Stop Channel window.

7.2.1 Understanding channel status

A queue manager holds status records associated with the channel objects known to that queue manager. These channel objects are the ones manually defined on that queue manager, defined automatically using channel auto-definition, or known through a queue manager cluster.

For channel types that can accept connections from applications or other queue managers, there can be multiple status records associated with a single channel object. This is because a queue manager can accept multiple connections using that channel object.

Status records are accessed with the DISPLAY CHSTATUS MQSC command. The most significant attribute of a status record is the `STATUS` attribute, which represents the overall state of that channel.

Note: In the WebSphere MQ Explorer, all status records corresponding to a distributed message channel, or client channel, object can be displayed. To do this, select the **Channels** folder under the queue manager in the navigator view, right-click a particular channel, and select **Status** → **Current status**.

If no channel status records exist for a channel object, the channel associated with that channel object is referred to as inactive, or in the `INACTIVE` state.

The `STATUS` attribute of a status record can have the following values:

- `RUNNING`: A running channel is one where the MCAs have successfully negotiated, and messages or MQI commands can flow across that channel.
- `STOPPED`: The channel object associated with the status record is disabled. This means that the channel that would be established using the channel object is not able to start until the channel is enabled. This status can be entered by manually stopping the channel using the STOP CHANNEL MQSC command or the WebSphere MQ Explorer. A channel object can be enabled using the START CHANNEL MQSC command or using the WebSphere MQ Explorer. For channel objects that establish channels, this action causes the channel to start. A `STOPPED` status record persists across a queue manager restart.

- `RETRYING`: For channels that establish connections, an attempt to start the channel has failed. The channel automatically retries the connection at regular intervals. This behavior is defined using the short retry interval (`SHORTTMR`), short retry count (`SHORTRTY`), long retry interval (`LONGTMR`), and long retry (`LONGRTY`) attributes of the channel object. After the configured number of retry attempts have been performed, the channel enters `STOPPED` status and must be manually restarted. A `RETRYING` status record persists across a queue manager restart.
- `STOPPING`, `STARTING`, `BINDING`, `REQUESTING`, `INITIALIZING`: These are intermediate states channels can enter while being established or ending.
- `PAUSED`: This state relates to message channels retrying the action of delivering a message. We describe this in 7.4.11, “Message delivery failures” on page 175.

7.2.2 Channel names

Channel names can contain a maximum of 20 characters. These can be uppercase characters, lowercase characters, numeric characters, or the . / _ % symbol characters.

This length is significantly shorter than the 48-character maximum length of a queue manager name on platforms except WebSphere MQ for z/OS. A channel name should reflect its use in a consistent way to aid administration.

If you keep all queue manager names within the system 17 characters or shorter, you can use the following naming scheme for all channels within the system:

`TO.`*`Destination_Qmgr`*

This naming convention can work for both distributed and cluster message channels. It is flexible from the perspective that multiple queue managers can connect to a queue manager using the same channel name, but each channel from a particular queue manager only connects to one destination queue manager.

7.3 Client channels

Client channels enable applications that are not running on the same machine as a queue manager to connect to that queue manager and perform the same operations as though they were connected to that queue manager locally.

Client channels can also be used to connect an application that is running on the same machine as a queue manager in order to benefit from the stability provided by client channels. Connecting to a queue manager using a client channel provides the greatest isolation between the application and the queue manager; so if an application fails and corrupts the resources to which it has access, it is least likely to affect the operation of the queue manager.

7.3.1 Operation of client channels

When an application connects to a queue manager using the connect mechanism provided by the API used to develop the application, it invokes an MCA provided by that client API. This MCA is called a *client MCA* in this book.

This MCA establishes communication with the queue manager over a client channel through the listener provided by that queue manager. It issues an MQCONN or MQCONNX MQI call over that channel to create a connection.

All subsequent message queuing calls made by the application that use that queue manager call into the MCA. The MCA issues MQI calls to the queue manager across the client connection.

At the queue manager side of the connection, a *server connection* MCA is started by the queue manager. This performs the MQI calls requested by the client MCA using a local connection to the queue manager.

The attributes of this server connection MCA can be based on a server connection channel object defined on that queue manager, or it can be defined automatically using channel auto-definition, as described in 7.5, “Channel auto-definition” on page 179.

7.3.2 Server connection channel objects

A server connection channel object defines the name of a channel that a client can use to connect to a queue manager and the attributes of the MCA that hosts that connection.

Use one of the following methods to define the server channel objects:

- Using the DEFINE CHANNEL CHLTYPE(SVRCONN) MQSC command.
- Using the WebSphere MQ Explorer:
  a. Right-click the **Channels** folder under a particular queue manager in the navigator view of the WebSphere MQ Explorer.
  b. Select **New** → **Server-connection Channel**.

One important consideration for server connection channel objects is the identity context that the server connection MCA assumes when performing MQI commands.

7.3.3 Security considerations

By default, this identity context is the user identifier under which the application is running on the remote machine where it is running. However, there are complications relating to the passing of user identifiers between different machines with different operating systems. Some examples of this include:

- The listener for a queue manager listens on a TCP/IP port on the machine that hosts the queue manager. It might be difficult to restrict access to this port to only the machines in the network or in interconnected networks that are designated to connect to the queue manager. It might be possible for one of these applications to run under the mqm administrator user identifier on the local machine, and thus connect to the remote machine with the access privileges of that user identifier.
- The number of applications that connect to a queue manager might be very large and run under different user identifiers on different machines. Each one of these user identifiers needs to be defined and maintained on the machine hosting the queue manager in order for these applications to connect successfully to the queue manager.
- WebSphere MQ for Windows supports longer user identifiers than WebSphere MQ for UNIX. When an application running on Windows connects using a client connection to a queue manager hosted on UNIX, generally, the first 12 characters of the user identifier are passed in lowercase as the identity context for that application. This user identifier needs to be defined on the UNIX machine in order for the connection to succeed. Common user identifiers on Windows, such as Administrator, are longer than 12 characters.
- Some client MCAs, such as the WebSphere MQ V5.3 Java or JMS client MCAs, do not pass a user identifier to the server connection MCA.

Due to these considerations, we generally recommend that all applications that connect using a particular channel name are given the same identity context. This is performed using the MCA user (`MCAUSER`) attribute on the server connection channel object.

Note: When using MQSC on UNIX, it is important to enclose the `MCAUSER` attribute value in single quotation marks. This is because user identifiers are case-sensitive on the UNIX platforms. An example definition is:

```
DEFINE CHANNEL(PAYROLL.CLIENT) CHLTYPE(SVRCONN) MCAUSER('mqclient')
```

In order to provide greater assurance that only correctly authorized applications can connect to a queue manager, consider the use of technologies such as firewalls. These allow only certain machines to connect to the ports of a machine hosting a queue manager.

Another alternative is to protect a server connection channel using SSL. SSL can ensure that a connecting application has a valid certificate signed by a trusted certificate authority. A server connection channel can be configured to only accept connections from particular distinguished names as contained within the certificate provided by the connecting application.

7.3.4 Configuring a client MCA to connect to a queue manager

There are a number of ways in which an application can specify the attributes of the client MCA.

For example, attributes that a client MCA requires are the connection name it uses to connect to a queue manage and the name of the channel it establishes.

Some methods used by an application to specify these attributes are specific to the individual API used to access the WebSphere MQ message queuing infrastructure.

These attributes are often referred to as simply *connection settings*.

For example, a client using the MQI interface directly from the C programming language can use the `MQCNO` structure passed to a MQCONNX call to specify the connection settings of the client MCA.

C and C++ applications connecting to a queue manager can use the `MQSERVER` environment variable to specify simple connection settings. The syntax of this environment variable is as follows:

```
CHANNEL.NAME/TCP/hostnameoripaddress(port)
```

Applications using the WebSphere MQ base Java API, or the WebSphere MQ .NET API, can specify connection settings in the properties of the `MQEnvironment` class.

Applications using the JMS API can specify connection settings as parameters to a `MQConnectionFactory` object created using the JMS administration interface.

7.3.5 Client connection channel objects

WebSphere MQ provides client connection channel objects as another method of specifying connection settings, including the more advanced attributes such as the configuration of SSL.

Use one of the following methods to define the client connection channel objects:

- Using the DEFINE CHANNEL CHLTYPE(CLNTCONN) MQSC command.
- Using the WebSphere MQ Explorer:
  a. Right-click the **Client Connections** folder under a particular queue manager in the navigator view of the WebSphere MQ Explorer.
  b. Select **New** → **Client-connection Channel**.

Each client connection channel object specifies a queue manager name (`QMNAME`) attribute. This is used by the client MCA when finding a suitable client connection channel object to use to connect to a queue manager. The application specifies the name of a queue manager to which it needs to connect, and this queue manager name can be prefixed with an asterisk (*) character.

The client MCA can attempt to connect using multiple client connection objects defined on a queue manager. This way, an application can connect to a secondary, or backup, queue manager when the primary queue manager is not available.

How an application specifies a queue manager name depends on the API being used, for example, the `QMgrName` parameter of the MQCONN or MQCONNX call when using the MQI, or the `QMANAGER` property of a `MQConnectionFactory` object created in the JMS administration tool for the JMS API.

Table 7-1 provides an overview of how the queue manager name specified by an application relates to the client connection channels that are used by the client MCA to connect to a queue manager.

Table 7-1 Summary of how a client MCA chooses client connection channel objects

Queue manager name given by application	**Description of how matching client connection channel objects are found**
A blank queue manager name	Client connection channel objects with a blank queue manager name attribute match. If a queue manager can be contacted using the channel name and attributes of one of these client connection channel objects, the connection succeeds regardless of the actual name of the queue manager.
`*Queue_Manager_Name` The asterisk must be the first character of the queue manager name given by the application.	Only client connection channel objects with a queue manager name of *`Queue_Manager_Name`* match. If a queue manager can be contacted using the channel name and attributes of one of these client connection channel objects, the connection succeeds regardless of the actual name of the queue manager. This mechanism is useful for providing backup queue managers if a single queue of a particular name is unavailable for connection.

Queue manager name given by application	Description of how matching client connection channel objects are found
`Queue_Manager_Name`	Only client connection channel objects with a queue manager name of `Queue_Manager_Name` match. If a queue manager can be contacted using the channel name and attributes of one of these client connection channel objects, the connection only succeeds if the actual name of the queue manager is `Queue_Manager_Name`.

7.3.6 Client channel definition table (CCDT)

Each client connection channel object that is defined adds an entry to a file called the client channel definition table (CCDT) individual to that queue manager.

The CCDT is not human readable, but it can be read by multiple different client MCAs provided by client APIs.

This file is stored in the following location:

- WebSphere MQ for Windows:

  C:\Program Files\IBM\WebSphere MQ\Qmgrs *Queue_Manager_Name*\@ipcc\AMQCLCHL.TAB
- WebSphere MQ for UNIX:

  /var/mqm/qmgrs/*Queue_Manager_Name*/@ipcc/AMQCLCHL.TAB
- WebSphere MQ for iSeries:

  /QIBM/UserData/mqm/qmgrs/*Queue_Manager_Name*/&ipcc

After a set of client connection channel objects are defined on a queue manager, not necessarily the queue manager to which an application connects, the file can be copied to another location for use by a client API.

For example, it can be placed on a directory that is shared over a network to all machines that run applications that connect to the queue manager.

Before an application can connect, in the way described in Table 7-1 on page 165, the client API must be configured to know the location of the CCDT.

Not all client APIs currently support use of a CCDT. Examples that do support the CCDT include:

- WebSphere MQ C and C++ client APIs:
  The location of the CCDT is configured using the `MQCHLLIB` and `MQCHLTAB` environment variables. `MQCHLLIB` is set to the directory that contains the CCDT. `MQCHLTAB` is set to the name of the CCDT file.

An example of a Windows client on the same machine as the queue manager on which the CCDT was created is as follows:

```
MQCHLLIB=C:\Program Files\IBM\WebSphere MQ\Qmgrs\Queue_Manager_Name\@ipcc
MQCHLTAB=AMQCLCHL.TAB
```

Note: If the `MQCHLLIB` and `MQCHLTAB` environment variables are not set, the default location to search for a CCDT is:

- Windows:

  C:\Program Files\IBM\WebSphere MQ\AMQCLCHL.TAB
- UNIX:

  /var/mqm/AMQCLCHL.TAB

- WebSphere MQ V6.0 base Java client API:
  A URL object is passed as the second parameter to the constructor of a `MQQueueManager` object.
- WebSphere MQ V6.0 JMS client API:
  A URL is specified in the `CCDTURL` property of a `MQConnectionFactory` created in the JMS administration tool.

7.4 Distributed message channels

A distributed message channel is one that takes messages from a particular transmission queue on one queue manager and delivers those messages to queues on a specific remote queue manager.

A distributed message channel must be manually configured for all transmission queues defined on a queue manager. As discussed in 6.2.1, “Queue name resolution” on page 134, a queue manager places a message on a transmission queue as the result of queue name resolution. During this process, it adds enough information to that message in a transmission queue header for queue name resolution to occur when that message reaches the destination queue manager.

Note: The administrative impact of defining distributed message channels can be reduced by using a queue manager cluster.

7.4.1 Message transmission

Each distributed message channel consists of two MCAs, thus two channel objects, one on each queue manager. Each MCA assumes one of the following roles based on the type of message channel object defined on the queue manager:

- Sending MCA:
  The sending MCA opens a particular transmission queue, specified in the attributes of the channel object, for exclusive input. This means that no two channels can be configured to flow messages from the same transmission queue. It gets messages from that transmission queue and sends them to the partner MCA.

  Note: This is not the case for cluster message channels, because all cluster message channels use the same transmission queue. We discuss queue manager clusters in Chapter 8, "Queue manager clusters" on page 181.

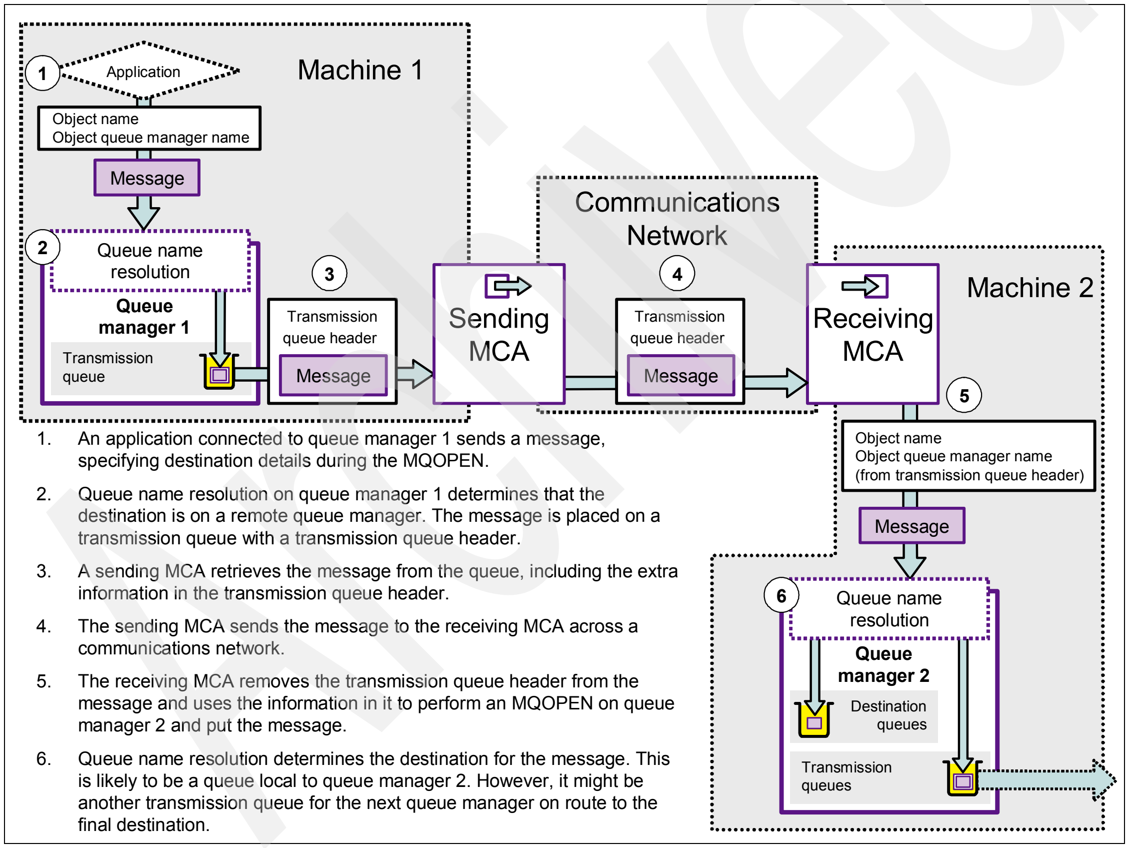
- Receiving MCA:
  The receiving MCA receives messages from the sending MCA. For each message, it removes the transmission queue header from the message and reads its contents. It opens the queue specified in the transmission queue header for that message and then puts the message to the queue. These open and put actions are performed using standard MQI calls. This means that queue name resolution happens on the queue manager in the same way as though an application had connected to that queue manager and put the message using the details in the transmission queue header. If a valid destination cannot be determined for the message during the queue name resolution on that queue manager, the message cannot be delivered. We discuss this in 7.4.11, "Message delivery failures" on page 175.

The transmission queue header is generated during queue name resolution on the queue manager where the sending MCA is running and used to put the message to a queue on the queue manager where the receiving MCA is running. The header contains the following information:

- Remote queue name:
  The name of the destination queue for the message, which was resolved by the queue manager during queue name resolution. When the MCA at the remote queue manager attempts to put the message to a queue on the remote queue manager, it specifies this queue name in `MQOD` when opening the queue.
- Remote queue manager name:
  The name of the destination queue manager that hosts the destination queue. This is also resolved during queue name resolution and might not match the name of the queue manager to which the message is delivered, for example, if the next queue manager to receive the message is not the final destination for the message.

- The original message descriptor of the message:
  In order to add the transmission queue header to a message, the message becomes altered. The message queue header is added to the beginning of the message body, and the message descriptor is altered to describe the transmission queue header, rather than the data contained in the message. However, when the message arrives on the destination queue, the message must reflect the original message correctly, including the message descriptor. The original message descriptor of the message is stored in the transmission queue header and used when the MCA puts the message to the remote queue manager with the transmission queue header removed.

Figure 7-2 shows the intercommunication over a channel between two queue managers.



1. An application connected to queue manager 1 sends a message, specifying destination details during the MQOPEN.
2. Queue name resolution on queue manager 1 determines that the destination is on a remote queue manager. The message is placed on a transmission queue with a transmission queue header.
3. A sending MCA retrieves the message from the queue, including the extra information in the transmission queue header.
4. The sending MCA sends the message to the receiving MCA across a communications network.
5. The receiving MCA removes the transmission queue header from the message and uses the information in it to perform an MQOPEN on queue manager 2 and put the message.
6. Queue name resolution determines the destination for the message. This is likely to be a queue local to queue manager 2. However, it might be another transmission queue for the next queue manager on route to the final destination.

Figure 7-2 Communication between queue managers with sending and receiving MCAs

7.4.2 Batches

To provide exactly once assured delivery of messages, both the sending and receiving MCA must be able to ensure that a message is not lost or sent twice and that it is received successfully by the receiving MCA.

To ensure this, a sending MCA can use a unit of work when getting messages from a queue, and a receiving MCA can use a unit of work when putting messages.

Note: By default, message channels do not use a unit of work for transferring nonpersistent messages. This means that communication failures can result in the loss of nonpersistent messages. This default behavior can be changed for a channel by specifying a nonpersistent message speed (NPMSPEED) of normal, instead of fast, on the channel object for the sending or receiving MCA.

These two separate units of work must be coordinated between the two MCAs, to ensure that if a communications failure occurs, and messages are not lost or delivered twice. To do this, the two MCAs must communicate over the network to either both commit their unit of work, or both back out their unit of work if a communication failure occurs.

Additional network communication is involved in this process, so it is not usually performed for each message that is transferred across the channel. Individual messages being transferred are grouped together in a *batch*, and all of those messages are either committed to their target queues or backed out onto the transmission queue as a group.

The maximum number of messages that are contained within a batch is called the *batch size*. This is configured using the batch size attribute (BATCHSZ) on the channel object for the sending and receiving MCA. The batch size used is negotiated to the lower of the two attributes.

In some cases, there are insufficient messages available on the transmission queue to fill a batch. In this case, the sending MCA waits for a short time for messages to arrive on the transmission queue before committing the messages in that batch that have already been sent. The number of milliseconds a sending MCA waits before committing a batch is configured with the batch interval attribute (BATCHINT). This is specified on the channel object for the sending MCA.

7.4.3 Indoubt channels and sequence numbers

If a network communication failure is encountered while a channel is performing confirmation of a batch between the two MCAs, a channel can become indoubt.

This is because a confirmation request was sent, but no reply was received. The MCA that sent the confirmation request does not know if the partner MCA received that request and the communication failed during the reply or whether the communication failed during the send.

The unit of work for one channel must remain in a prepared state, ready to be either committed or backed out, depending on the action that was performed by the partner MCA. Both MCAs maintain a sequence number related to the number of messages successfully transferred over the channel in order to automatically resolve the indoubt state when the channel restarts.

An indoubt state on a channel is represented by an `INDOUBT` attribute of `YES` on the channel status record for the channel.

Note: Refer to *WebSphere MQ Intercommunication*, SC34-6587, for more information about indoubt channels and the manual steps you can perform if an indoubt state is encountered and not automatically resolved.

7.4.4 Disconnection intervals

It might not be efficient to leave a communication channel open indefinitely. WebSphere MQ allows a message channel to be automatically ended if no messages are sent across that channel in a time interval.

This time interval is specified in seconds using the disconnect interval (`DISCINT`) attribute on the channel object for the sending MCA.

A disconnect interval of zero can be used to specify that a channel should remain open indefinitely.

7.4.5 Connection names

In order for a channel to be established between a sending and receiving MCA, one MCA must contact the partner queue manager using the listener of that queue manager.

A channel can be started from either side of a connection depending on the type of channel object that defines the MCAs on each side of the channel. We discuss this in 7.4.10, “Valid distributed message channel object pairs” on page 174.

The connection name (`CONNAME`) attribute of a channel object specifies the TCP/IP host name or IP address and port where a listener is running for the partner queue manager.

We discuss connection names and listeners in 5.3.8, “Providing network access to a queue manager” on page 112. A connection name is specified as follows:

```
hostname.or.ipaddress(port)
```

If no port is specified, the well-known TCP/IP port number of WebSphere MQ, 1414, is used.

Note: When defining channels using MQSC, the connection name must be contained in single quotation marks, for example:

```
DEFINE CHANNEL(TO.EXAMPLE.PAYROLL) CHLTYPE(SDR) +
       XMITQ(EXAMPLE.PAYROLL) +
       CONNAME('payroll.example.com(9001)')
```

7.4.6 Receiver channel objects

Receiver channel objects are defined on a queue manager to define the attributes of a receiving MCA to which other queue managers can send messages.

A receiver channel object cannot be used to initiate a channel.

Use one of the following methods to define receiver channel objects:

- Using the DEFINE CHANNEL CHLTYPE(RCVR) MQSC command.
- Using the WebSphere MQ Explorer:
  a. Right-click the **Channels** folder under a particular queue manager in the navigator view of the WebSphere MQ Explorer.
  b. Select **New** → **Receiver Channel**.

7.4.7 Requester channel objects

Requester channel objects are defined on a queue manager to define the attributes of a receiving MCA to which other queue managers can send messages.

A requester channel object can be used to initiate a channel.

Use one of the following methods to define requester channel objects:

- Using the DEFINE CHANNEL CHLTYPE(RQSTR) MQSC command.
- Using the WebSphere MQ Explorer:
  a. Right-click the **Channels** folder under a particular queue manager in the navigator view of the WebSphere MQ Explorer.

   b. Select **New** → **Requester Channel**.

The connection name (`CONNAME`) attribute is required.

7.4.8 Sender channel objects

Sender channel objects are defined on a queue manager to define the attributes of a sending MCA that can send messages to other queue managers from a specified transmission queue.

Only one sender or server channel MCA can be active at any time for the same transmission queue.

A sender channel object can be used to initiate a channel.

Use one of the following methods to define sender channel objects:

- Using the DEFINE CHANNEL CHLTYPE(SDR) MQSC command.
- Using the WebSphere MQ Explorer:
   a. Right-click the **Channels** folder under a particular queue manager in the navigator view of the WebSphere MQ Explorer.
   b. Select **New** → **Sender Channel**.

The transmission queue (`XMITQ`) and connection name (`CONNAME`) attributes are required.

7.4.9 Server channel objects

Server channel objects are defined on a queue manager to define the attributes of a sending MCA that can send messages to other queue managers from a specified transmission queue.

Only one sender or server channel MCA can be active at any time for the same transmission queue.

A server channel object can only be used to initiate a channel if a connection name is specified in its definition. If a connection name is specified, the server channel object is said to be *fully qualified*.

Use one of the following methods to define server channel objects:

- Using the DEFINE CHANNEL CHLTYPE(SVR) MQSC command.
- Using the WebSphere MQ Explorer:
   a. Right-click the **Channels** folder under a particular queue manager in the navigator view of the WebSphere MQ Explorer.

b. Select **New** → **Server Channel**.

The transmission queue (`XMITQ`) attribute is required.

Note: Do not confuse server channel objects with server connection channel objects, discussed in 7.3.2, “Server connection channel objects” on page 162.

7.4.10 Valid distributed message channel object pairs

The MCAs created from the previously discussed channel objects can only be partnered together to form a channel in certain combinations. In this section, we describe the valid combinations and their uses.

Sender-receiver channels

This form of channel can only be initiated from the sender side.

Multiple sender channel objects, defined on different queue managers, can be used to connect to the same receiver channel object on a queue manager.

Commonly, a single receiver channel object is defined on a queue manager, and all queue managers in the infrastructure has sender channels defined with the same name as that receiver channel to communicate with that queue manager.

Requester-server channels

This form of channel can be initiated from the requester side, or optionally also from the server side if that server is fully qualified with a connection name.

This form of channel does not ensure that a requester initiating a channel is hosted at a particular connection name. This allows multiple requesters to be defined with the same name on different queue managers and for each to request messages from a single transmission queue on the same remote queue manager. However, only one channel to a requester can be active getting messages from the transmission queue at any time.

Requester-sender channels

This form of channel is similar to a requester-server channel with a fully qualified server. However, after the channel has been initiated by the requester, the channel is disconnected and reinitiated by the sender channel using the connection name in the sender channel object.

This enables the sender channel to ensure that it is partnered with a requester channel hosted on a particular queue manager.

Server-receiver channels

This form of channel is functionally equivalent to a sender-receiver pair. The channel is initiated from the server side, so the server must be fully qualified with a connection name.

7.4.11 Message delivery failures

If the receiving MCA cannot deliver a message to a queue, the MCA takes predictable action for that message.

A delivery of a message might fail for the following reasons:

- Queue name resolution failed while attempting to open the queue. The queue name and queue manager name used when opening the queue to put the message are contained in the transmission queue header, as described in 7.4.1, “Message transmission” on page 167. We discuss queue name resolution and how it is affected by defining each type of queue object in 6.2.1, “Queue name resolution” on page 134.
- The resolved queue, which can be a local queue on the queue manager or a transmission queue representing the next destination on route to the final destination, already contains the maximum allowed number of messages.
- The resolved queue has been put disabled.

The steps taken by the MCA are controlled by the attributes of the channel object for that MCA and the dead letter queue (`DEADQ`) attribute of the queue manager object. Here, we summarize these steps:

1. Because some delivery failures can be transitory in nature, such as a queue containing the maximum allowed number of messages, the action of putting the message can be retried by the receiving MCA. The number of times an MCA attempts to retry putting the message is configured using the message retry (`MRRTY`) attribute, specified on the channel object that defines the receiving MCA. The delay between these retries is configured using the message retry timer (`MRTMR`) attribute, specified on the channel object that defines the receiving MCA.
2. If the dead letter queue (`DEADQ`) attribute on the queue manager object has been specified, the receiving MCA attempts to open a queue of that name. If this open is successful, a dead letter header (`MQDLH`) is added to the message, and the message is put to the dead letter queue.

   The dead letter header contains information about the failure. This includes the queue name and queue manager name from the transmission queue header. It also contains the *reason code* returned from the failed attempt to open the queue to put the message. We discuss reason codes in 6.1.2, “Completion codes and reason codes” on page 127.

3. If no dead letter queue is defined for the queue manager, or the dead letter queue does not exist, the action depends on whether units of work are being used to transfer messages across the channel. As discussed in 7.4.2, “Batches” on page 170, units of work are used when transferring persistent messages, or nonpersistent messages when a nonpersistent message speed of *normal* has been specified on the channel.

   If units of work are not being used in delivering the messages, the nonpersistent message is discarded.

   If units of work are being used in delivering the message, the message is backed out onto the transmission queue at the sending side. The channel then enters `RETRYING` status, as described in 7.2.1, “Understanding channel status” on page 160. The channel attempts to deliver the message again each time it attempts to restart. If the number of retries configured for the channel is exceeded, the channel enters `STOPPED` status and needs to be manually restarted. No messages can flow across the channel until the message can be delivered, a dead letter queue is defined, or the message is manually removed from the transmission queue.

> **Note:** We recommend that you define a dead letter queue for all queue managers to avoid the actions described in step 3.
>
> A queue called the SYSTEM.DEAD.LETTER.QUEUE is provided when each queue manager is created. The queue manager can be configured to use this queue as the dead letter queue. However, this is not generally advised, because it is useful to hide all queues with the `SYSTEM.` prefix in the WebSphere MQ Explorer.
>
> Ensure that the maximum message length attribute on the dead letter queue is large enough to ensure that no messages are truncated when they are put to this queue.

7.4.12 Dead letter queue handling

After a dead letter queue has been defined for a queue manager, consideration should be taken for messages being delivered to that queue. Take actions to ensure that messages do not remain on the dead letter queue indefinitely, unprocessed by the applications in the system.

Performing actions on messages when they arrive on a dead letter queue is generally referred to as dead letter queue handling.

Approaches to dead letter queue handling include:

- Regular manual inspection of the dead letter queue:
  The format of the dead letter header, attached to messages when they are placed on the dead letter queue, is not easily human readable. Because of this, the WebSphere MQ Explorer provides the ability to display the individual fields within the dead letter header in a human readable form. To access this functionality, perform the following steps:

  a. Select the **Queues** folder under a queue manager in the navigator view.
  b. Right-click the dead letter queue in the Queues content view.
  c. Select **Browse Messages**.
  d. Right-click a message displayed in the table.
  e. Select **Properties**.
  f. Select the **Dead-letter header** section.

- Triggering the same action every time the dead letter queue contains messages:
  An example of this type of action is to execute a simple application that sends an e-mail to an administrator so that they can manually inspect the dead letter queue. See 6.3, "Triggering" on page 150 for details about triggering.

  Examples of the types of triggering that might be used are:

  – `FIRST` triggering with a long trigger interval specified for the queue manager. This causes the application to be executed regularly when there are messages on the dead letter queue.
  – `EVERY` triggering. This causes the application to be executed each time a message arrives on the dead letter queue.

- Using the WebSphere MQ dead letter queue handler:
  The WebSphere MQ dead letter queue handler can automatically perform actions on the messages that arrive on the dead letter queue based on a set of rules configured when the handler is started. For information about the WebSphere MQ dead letter queue handler, see *WebSphere MQ Intercommunication*, SC34-6587.

- Using a custom developed dead letter queue handler:
  If the rules provided by the WebSphere MQ dead letter queue handler are not sufficient for an individual requirement, a custom application can be written that processes messages as they arrive on the dead letter queue.

7.4.13 Channel initiation

Message channels are not automatically restarted by WebSphere MQ when they become inactive due to reaching their disconnect interval or after a queue manager is restarted.

The number of channels on a queue manager can be large, and manually starting channels using MQSC or the WebSphere MQ Explorer might not be an efficient mechanism in an infrastructure of interconnected queue managers.

WebSphere MQ for Windows and UNIX provide the channel initiator process to automatically start channels when messages arrive on transmission queues.

> **Note:** Do not confuse the channel initiator described in this section with the WebSphere MQ for z/OS channel initiator described in 5.3.10, “WebSphere MQ for z/OS channel initiator” on page 115.
>
> WebSphere MQ for z/OS and WebSphere MQ for iSeries provide channel listener programs that provide functionality for starting channels.

The channel initiator is a trigger monitor application that reads trigger messages from an initiation queue and starts the channel specified in the trigger data contained in each trigger message.

A WebSphere MQ channel initiator can be started using the **`runmqchi`** WebSphere MQ control command. However, a default channel initiator is provided by WebSphere MQ and started automatically with a queue manager.

This default channel negotiator can be disabled by changing the start channel initiator (`SCHINIT`) parameter on the queue manager object to `MANUAL`.

The default channel initiator monitors the failing initiation queue:

```
SYSTEM.CHANNEL.INITQ
```

Configuring a transmission queue to automatically start a channel that processes messages from that queue is performed by configuring attributes on the queue as follows:

- Triggering to on: `TRIGGER`
- Trigger type to first: `TRIGTYPE(FIRST)`
- Trigger data to the name of the channel: `TRIGDATA('`*`TO.remote.qmgr`*`')`
- Initiation queue to: `SYSTEM.CHANNEL.INITQ`: `INITQ(SYSTEM.CHANNEL.INITQ)`

7.5 Channel auto-definition

A queue manager can be configured to automatically create channel objects in response to MCA connection requests.

Channel auto-definition is enabled for a queue manager by configuring the channel auto-definition (`CHAD`) attribute of the queue manager object to `ENABLED`.

7.5.1 Channel auto-definition for client channels

If an application attempts to connect to a queue manager using a client channel, and no corresponding server connection channel object exists of the correct name on the queue manager, one is created automatically.

This object exists after channel auto-definition has occurred as though it had been manually created. The attributes of this server connection channel object are based on the following server connection channel object, which is created automatically with the queue manager:

```
SYSTEM.AUTO.SVRCONN
```

7.5.2 Channel auto-definition for distributed message channels

If a remote queue manager attempts to establish a connection using a sender channel object, or a fully qualified server channel object, and no corresponding receiver or requester channel exists of the correct name on the queue manager, a receiver channel object is created automatically.

This object exists after channel auto-definition has occurred as though it had been manually created. The attributes of this receiver channel object are based on the following receiver channel object, which is created automatically with the queue manager:

```
SYSTEM.AUTO.RCVR
```

8

Queue manager clusters

This chapter introduces queue manager clusters and shows how they can reduce administration of WebSphere MQ infrastructures and provide additional features that can increase service availability and provide workload balancing across multiple instances of a service hosted within a cluster.

We discuss the following topics:

- Overview of clustering concepts
- Viewing cluster repository information
- Actions on queue managers in a cluster
- Workload balancing

8.1 Overview of clustering concepts

All queue managers within a queue manager cluster have knowledge of the resources hosted within that queue manager cluster without requiring explicit local definitions for those resources. New resources, such as queue managers and the queues they host, can be added dynamically to the queue manager cluster and automatically made available for use by all queue managers within the queue manager cluster.

This allows additional hardware resources to be added to or removed from the queue manager cluster to respond to changing loads placed on the WebSphere MQ message queuing infrastructure.

A queue manager can be a member of multiple queue manager clusters, allowing the segregation of the components of a WebSphere MQ infrastructure based on the services provided or organizing into groups. These queue managers can provide a bridge between queue manager clusters, as well as bridging between a queue manager cluster and an existing WebSphere MQ infrastructure based on distributed message channels.

Note: When implementing a new WebSphere MQ infrastructure, we recommend that you use a queue manager cluster as the basis of this infrastructure.

Existing WebSphere MQ infrastructures can be extended to benefit from the additional workload balancing and high availability features provided by a queue manager cluster by bridging the existing infrastructure to an extended infrastructure based on a queue manager cluster.

The concepts of a queue manager cluster, as described in this section, are sometimes considered initially complicated to understand. However, they resolve to an infrastructure of queue managers that is much simpler to administer than one using distributed message channels, as is shown in the practical examples in 10.4, “Create a queue manager cluster” on page 291.

This infrastructure can scale to contain thousands of queue managers.

Note: Within the rest of this chapter, the term *cluster* is used to mean a queue manager cluster. Do not confuse this with high availability clusters, which are not a WebSphere MQ-specific concept, as discussed in 3.6, “High availability” on page 44.

8.1.1 Full and partial repository queue managers

Every queue manager holds a *repository* of information about the clusters of which it is a member.

Each cluster of which a queue manager is a member can be configured to maintain a *partial* or *full* repository of information for that cluster. We summarize partial and full repositories as follows:

- Full repository:
  A full repository contains knowledge of every resource shared in the cluster. This includes all queue managers in the cluster and all queue objects shared by queue managers in the cluster. There are normally exactly two queue managers configured as a full repository for each cluster in the infrastructure.

  A queue manager that hosts a full repository for a cluster is called a *full repository queue manager* within that cluster.

- Partial repository:
  A partial repository maintains knowledge of full repository queue managers within the cluster. It only maintains knowledge of queue objects and partial repository queue managers that are required by that queue manager. If an application connected to that queue manager attempts to open a queue name that is not already known to the queue manager, the full repository queue managers of all clusters of which the queue manager is a member are queried when attempting to find that resource.

  A queue manager that hosts a partial repository for a cluster is called a *partial repository queue manager* within that cluster.

In order to join a cluster, a queue manager must have knowledge of a full repository for that cluster. For this reason, at least two full repository queue managers must be defined within each cluster. These two queue managers initially establish the cluster by communicating with each other.

It is not generally recommended to have more than two full repositories in a cluster. Because full repositories maintain a complete knowledge of all resources within the queue manager cluster, they must regularly exchange information. Increasing the number of full repositories increases the loads placed on the infrastructure from sharing this information.

8.1.2 Cluster names

Every cluster has a name. Cluster names can contain a maximum of 48 characters. These can be uppercase characters, lowercase characters, numeric characters, and the . / _ % symbol characters.

Using short cluster names can be useful, because it allows the cluster name to be included in all cluster channel object definitions.

8.1.3 Configuring a queue manager as a full repository

Note: We recommended that you perform this step prior to joining the queue manager to the cluster using the cluster sender channel objects and cluster receiver channel objects described in this section.

The repository (`REPOS`) and repository namelist (`REPOSNL`) attributes of the queue manager object specify the clusters in which a queue manager hosts a full repository of information.

The repository (`REPOS`) attribute specifies a single cluster name.

The repository namelist (`REPOSNL`) attribute specifies multiple cluster names. Specify the name of a namelist object defined on the queue manager in this attribute. Define the namelist object to contain a list of cluster names.

Note: Maintaining a full repository can cause significant processing within a queue manager, because a cluster might contain many queue managers or many queues might be hosted by the queue managers within the cluster. For this reason, we do not recommend that the full repository queue managers within a cluster host the services provided by that cluster.

We recommend that other queue managers on the same machine are defined if that machine is designated to host services within the cluster. In a large cluster, it might be appropriate to dedicate hardware to hosting the full repository queue managers for a cluster.

We also recommend that the two full repository queue managers for a cluster are not hosted on the same machine. This eliminates a single point of failure in the cluster, because partial repository queue managers are still able to gain information about the cluster if one machine fails or becomes unavailable.

8.1.4 Cluster message channels

All information within a cluster is transferred across *cluster message channels*. This includes when an application sends a message to a queue hosted within the queue manager cluster and when a queue manager requests information about resources hosted in the cluster from a full repository.

Cluster message channels are very closely related to distributed message channels. They transfer messages from a transmission queue on one queue manager to a queue hosted on a particular queue manager within the cluster. However, an important difference is that all messages sent across cluster channels are sent from the *same* transmission queue. The name of this transmission queue, which is automatically defined on every queue manager when it is created, is as follows:

```
SYSTEM.CLUSTER.TRANSMIT.QUEUE
```

The queue manager automatically places any messages that are targeted for queue managers within a cluster on this queue.

WebSphere MQ automatically creates and starts these cluster message channels whenever they are required. This means that after a queue manager has joined a cluster, further manual administration of channels is not required.

Cluster message channels use sending and receiving message channel agents (MCAs) and have the same capabilities as distributed message channels. For example, they have disconnection intervals, perform batching, can use SSL authentication, and have channel status records. See 7.4, "Distributed message channels" on page 167 for information about distributed message channels and how message transmission occurs between sending and receiving MCAs.

The initial step of a queue manager joining a cluster must be performed manually. When joining a cluster, a queue manager specifies the attributes that all queue managers within that cluster should use when establishing communication with that queue manager.

For this purpose, WebSphere MQ provides cluster channel objects. Cluster channel objects can be defined on a queue manager to initiate the process of joining that queue manager to a cluster.

8.1.5 Cluster receiver channels

A cluster receiver channel object defines the attributes that to be used by all queue managers within a cluster, or multiple clusters, when defining cluster sender channels to that queue manager.

The cluster receiver channel object is *published* in the cluster to inform the full repository queue managers in the cluster about the queue manager on which it is defined. This means that it is sent to one full repository within the cluster, which distributes it to the other full repository queue managers within the cluster. These full repositories, in turn, publish this definition to other partial repository queue managers in the cluster when they request resources hosted on that queue manager.

A cluster receiver channel object defines the name and attributes used by *both* the sending and receiving MCAs when queue managers in the cluster establish a channel to this queue manager.

For example, a disconnection interval is specified on the cluster receiver channel object for cluster message channels. This disconnection interval is used by all sending MCAs that establish a connection to the queue manager.

This can simplify administration, because certain attributes, such as SSL configuration, must match in the sending and receiving MCAs. In a cluster, these attributes only need to be configured on the queue manager joining the cluster, rather than at the queue managers on both sides of the connection.

Use one of the following methods to manually define cluster receiver channel objects:

- Using the DEFINE CHANNEL CHLTYPE(CLUSRCVR) MQSC command.
- Using the WebSphere MQ Explorer:
  a. Right-click the **Channels** folder under a particular queue manager in the navigator view of the WebSphere MQ Explorer.
  b. Select **New** → **Cluster-receiver Channel**.

The connection name (`CONNAME`) attribute of a cluster receiver channel object is required. This specifies the host name or IP address and port where a listener is running for the *queue manager on which it is defined*. This is the connection name that other queue managers in the cluster use to establish communication with the queue manager.

The cluster (`CLUSTER`) and cluster namelist (`CLUSNL`) attributes of a cluster receiver channel object specify the clusters to which the channel definition applies.

The cluster (`CLUSTER`) attribute specifies a single cluster name.

The cluster namelist (`CLUSNL`) specifies multiple clusters names. Specify the name of a namelist object defined on the queue manager in this attribute. Define the namelist object to contain a list of cluster names.

Removing a queue manager from a cluster is performed by a queue manager when the cluster attribute or cluster namelist attribute of the receiver channel, which joined that queue manager to the cluster, is changed to no longer specify a cluster, or when the cluster name is removed from the names attribute of the namelist specified in the cluster namelist attribute of the cluster receiver channel that joined that queue manager to a cluster.

Note: A queue manager can publish a single cluster receiver channel object in multiple clusters using the cluster namelist (`CLUSNL`), allowing the channels established to that queue manager in multiple clusters to have the same name. Channel objects of this type regularly use the following naming convention:

`TO.`*`QmgrName`*

Alternatively, a queue manager can publish separate cluster receiver channel objects in each cluster using the cluster (`CLUSTER`) attribute of each cluster receiver channel object. Channel objects of this type regularly use the following naming convention:

`TO.ClusName.QmgrName`

This second naming convention reduces the number of characters available for the queue manager name in the 20-character channel name.

8.1.6 Cluster sender channels

The term *cluster sender channel* is often used to describe different things:

- A manually defined cluster sender channel object, or `CLUSSDR`. This is used when joining a cluster to contact a full repository in the cluster.
- An automatically defined cluster sender, `CLUSSDRA`, or *auto explicit cluster sender*. These are established automatically by a queue manager based on the cluster receiver channel definitions published by queue managers in the cluster to communicate with the queue managers in the cluster.
- An automatically defined cluster sender that replaces a manually defined cluster sender, `CLUSSDRB`, or *auto cluster sender*. These are established automatically by a queue manager based on the cluster receiver channel definition published by a full repository queue manager to which a cluster sender channel object was manually defined when joining the cluster.

A manually, or *explicitly*, defined cluster sender channel object has a single purpose. This is to initially establish communication with a single full repository when joining a cluster.

The name of a manually defined cluster sender channel object *must* match the name of the cluster receiver channel object defined on the full repository queue manager that was published by that full repository queue manager when it joined the cluster.

Equally, some attributes of the manually defined cluster sender channel, such as the SSL configuration, must be valid in order for the connection to succeed.

Use one of the following methods to manually define cluster sender channel objects:

- Using the DEFINE CHANNEL CHLTYPE(CLUSSDR) MQSC command.
- Using the WebSphere MQ Explorer:
  a. Right-click the **Channels** folder under a particular queue manager in the navigator view of the WebSphere MQ Explorer.
  b. Select **New** → **Cluster-sender Channel**.

The connection name (`CONNAME`) attribute of a cluster sender channel object is required. This specifies the host name or IP address and port where a listener is running for a *full repository queue manager* within the cluster. It does not matter which full repository is specified. However, the name of the channel must match the name of the appropriate cluster receiver channel object defined on that full repository.

The cluster (`CLUSTER`) and cluster namelist (`CLUSNL`) attributes of a cluster sender channel object specify the clusters for which the channel definition can be used to contact a full repository.

The cluster (`CLUSTER`) attribute specifies a single cluster name.

The cluster namelist (`CLUSNL`) specifies multiple cluster names. Specify the name of a namelist object defined on the queue manager in this attribute. Define the namelist object to contain a list of cluster names.

When a cluster sender channel object is defined, any cluster receiver channel objects that apply to the same clusters are published to the full repository over that channel. This causes the automated process of establishing the queue manager's membership in those clusters to begin.

For each cluster, this process results in the cluster receiver channel definition being held by all full repositories in that cluster. The definition of any queue objects shared by the queue manager joining the cluster are also held by all full repositories. The queue manager joining the cluster gains knowledge of all full repositories in the cluster.

After this process is complete, the explicitly defined cluster sender channel is no longer used. The queue manager subsequently uses the cluster receiver channel definitions shared by each queue manager in the cluster. These allow it to establish channels to all full and partial repository queue managers in the cluster. Equally, those queue managers can establish channels back to the queue manager using its cluster receiver channel definition shared in the cluster.

Note: If more than two full repository queue managers are defined in a cluster, cluster sender channels must be explicitly defined from each full repository to each other full repository. Full repositories only gain knowledge of other full repositories to which they have an explicit cluster sender defined.

Partial respository queue managers only require a single explicit cluster sender channel definition, regardless of the number of full repository queue managers in the cluster. There is no benefit of defining more than one explicit cluster sender channel on a partial repository queue manager.

8.1.7 Sharing queue objects within clusters

Full repository and partial repository queue managers within a cluster can share queue objects within that cluster.

This allows these queue objects to be known automatically by all queue managers within the cluster. For example, an application connected to one queue manager can put a message to a local queue hosted on a second queue manager that has been shared within the cluster without specifying the name of that remote queue manager.

The following types of queue objects can be shared:

- Local queue objects:
  These are the type of objects most regularly shared within a cluster. This allows an application connected to any queue manager in a cluster to put a message to a queue shared by another queue manager in the cluster by specifying only the queue name when opening the queue.

Note: Dynamically created local queues cannot be shared in a cluster by sharing the model queue definition from which they are created.

However, if an application puts a message to *any* queue name with a specific queue manager name that is a member of the cluster, that message is delivered to that queue manager. This means that an application can send a reply to a dynamically created reply queue using the reply-to queue name and reply-to queue manager name specified in the message descriptor of the request message. The reply-to queue manager name is provided automatically by the queue manager to which the application that created the request message was connected.

As such, applications can perform request/reply messaging to a service hosted on a queue shared in a cluster in the same way as though that queue was hosted on the queue manager to which the application is connected.

- Alias queue objects:
  An alias queue shared in a cluster enables a queue manager to share a queue object hosted on that queue manager under a different name within the cluster. The name of the queue object local to the queue manager is specified in the target queue (`TARGQ`/base queue) attribute, and the name of the alias queue object is the name under which that queue is available in the cluster.
- Remote queue objects:
  Sharing different kinds of remote queue objects, described in 6.2.5, "Remote queue objects" on page 143, in a cluster has different effects. Common uses include:
  - Local definition of a remote queue:
    This enables a queue that is hosted outside of the cluster on a queue manager that is not a member of a cluster to be shared within the cluster. The name of the remote queue object is the name under which the queue is shared in the cluster. The name of the destination queue on the target queue manager is specified in the remote name (`RNAME`) attribute. The name of the queue manager that hosts the queue is specified in the remote queue manager name (`RQMNAME`) attribute. The name of the transmission queue on the queue manager where the remote queue definition is defined can be specified in the transmission queue (`XMITQ`) attribute, or it defaults to the remote queue manager name.

– Queue manager alias:
  This is where the remote name (RNAME) attribute of the remote queue object is blank. This has two primary uses:

  - A queue manager can assume an alternative name within the cluster. The alternate name is the name of the remote queue object shared in the cluster; the actual queue manager name is specified in the remote queue manager name (RQMNAME) attribute.
  - The name of a queue manager outside of a cluster can be known within the cluster and a transmission queue can be identified for that queue manager on the queue manager sharing the remote queue object definition. The name under which the queue manager is to be known in the cluster is the name of the remote queue object shared in the cluster. The name of the queue manager outside of the cluster is specified in the remote queue manager name (RQMNAME) attribute. The name of the transmission queue on the queue manager within the cluster, if different to the name of the queue manager outside of the cluster, is specified in the transmission queue (XMITQ) attribute.

Note: Another useful way to use remote queue objects within a cluster is to define a queue manager alias on a queue manager with a blank remote queue manager name (RNAME) attribute without sharing the remote queue object in the cluster.

This causes the messages that arrive at that queue manager from outside of a cluster with a particular queue manager name specified to be workload balanced within the cluster.

With all of these queue object types, multiple queue managers in a cluster can share queue objects with the same name and the same type, or different types, within the same cluster. We describe this in 8.4, “Workload balancing” on page 209.

The cluster (CLUSTER) and cluster namelist (CLUSNL) attributes of these queue objects specify the clusters in which the object is shared.

The cluster (CLUSTER) attribute specifies a single cluster name.

The cluster namelist (CLUSNL) specifies the names of multiple clusters. Specify the name of a namelist object defined on the queue manager in this attribute. Define the namelist object to contain a list of cluster names.

8.1.8 Queue manager identifier (QMID)

When each queue manager is created, a queue manager identifier (QMID) for that queue manager is created based on the time the queue manager was created and the queue manager name. It is very unlikely that two queue managers have the same QMID, even if two queue managers are created with the same name.

Clusters use this uniqueness to distinguish between the queue managers within a cluster, rather than relying on the queue manager name to be unique.

It is unlikely, and definitely not recommended, that two queue managers are intentionally joined to a cluster with the same name. However, it is possible that a queue manager might be re-created for some reason, for example, after a hardware failure. This new queue manager might be joined to the cluster without the previous queue manager that had the same name successfully leaving the cluster. It is important for the cluster to be able to distinguish between queue managers in these situations.

If this situation occurs, and multiple queue managers are observed within the cluster with the same name, see the documentation for the RESET CLUSTER ACTION(FORCEREMOVE) MQSC command for information about how to resolve the situation.

Note: Only take this action after reading the appropriate documentation in *WebSphere MQ Queue Manager Clusters*, SC34-6589. It is especially important to consider this action carefully if it is performed in relation to full repository queue managers in a cluster.

8.1.9 Cluster subscriptions and publications

In this section, we provide an overview of the way information is maintained within the full and partial repositories of a cluster. This provides a basis for understanding the information provided when viewing information about a cluster from a partial or full repository.

All full repositories in a cluster communicate to ensure that they maintain the same, complete information about all of the queue managers and queue objects in the cluster.

When a partial queue manager joins a cluster, it *publishes* information about itself, including the information provided within the cluster receiver channel object, to two full repositories within the queue manager cluster.

Equally, when a partial repository queue manager shares a queue object within a cluster, it publishes information about this queue object to two full repositories within the queue manager cluster.

These publications are repeated by a partial repository regularly, every 27 days, in order for the full repositories to ensure that a queue manager remains active within a cluster and information is up to date. In addition, any changes to these objects shared in the queue manager cluster are published to the full repositories.

If a publication is not repeated within 30 days, it expires and is not provided by full repositories in response to requests for information made by partial repositories. However, the repositories maintain them for another 60 days grace period in case the queue manager was temporarily unavailable for some reason. If, after this total 90 days has passed, a queue manager has not republished its information, that queue manager is no longer considered part of the cluster by the full repository. When communication to the full repository is reestablished by the partial repository, it becomes known in the cluster again.

A partial repository only maintains information about the queue managers and queue objects in the cluster in which it has an interest:

- All full repository queue managers successfully joined to the cluster
- Queue objects hosted within the cluster of the same name as the queue objects that the queue manager shares within the cluster
- Queue objects hosted within the cluster with names to which applications that connect to the queue manager send messages
- Partial repository queue managers that host queues objects known to the queue manager

A partial repository queue manager *subscribes* to full repository queue managers in order to gain this information. In response to a subscription, it receives all relevant publications known to that full repository, or is informed that no matching information is available.

These subscriptions are generated by a partial repository when it shares a new object in the cluster. They are also generated during queue name resolution when applications connected to the queue manager attempt to open queues to put messages.

If object names or object queue manager names specified when opening queues to put message are not known to the partial repository queue manager, subscriptions are generated to full repositories for all clusters in which it is a partial repository.

Note: A queue manager might wait for up to 10 seconds to get a response from a full repository queue manager when opening a queue currently unknown to the partial repository. If this fails, the application attempting to put the message is unable to do so. Therefore, it is important that communication can always be established with at least one full repository in the cluster.

Subscriptions only have a finite lifetime of 30 days. Within those 30 days, a partial repository is automatically sent updates by full repository queue managers regarding its subscriptions, and it will renew a subscription after 27 days. However, it only does this if publications relating to that subscription have been accessed since the subscription was last renewed. Otherwise, it lets the subscription expire. However, it issues the subscription again when an application next attempts to open the queue object.

While a subscription is valid, the information contained in the partial repository relating to that subscription is also valid, because the full repository automatically provides updates as they occur. This allows the partial repository to communicate directly with the queue managers hosting queue objects, without any need to contact a full repository. Therefore, queues can be opened and messages can be put efficiently by the applications connected to that partial repository.

The repository of a queue manager is maintained by the repository manager component of the queue manager. This manages subscriptions and sends publications to the full repositories in the cluster.

This mechanism provides a balance between the number of subscriptions that must be maintained by a full repository and the knowledge contained within the partial repositories to allow applications to access queue objects efficiently.

8.2 Viewing cluster repository information

As we discussed earlier, status records for cluster message channels are held by each queue manager in the same way that status records are held for distributed message channels. Cluster message channels can also be started, stopped, enabled, and disabled using the START CHANNEL and STOP CHANNEL commands in MQSC.

However, further information about clusters can also be obtained from MQSC and the WebSphere MQ Explorer based on the current contents of the partial or full repository held by a queue manager.

8.2.1 Viewing repository information in MQSC

MQSC provides two commands to access this information:

- DISPLAY QCLUSTER:
  This command shows information about all instances of queue objects shared within clusters, as known to that queue manager. For example, issuing the following command on a full repository for a cluster shows all the information known about all queue objects shared within a queue manager cluster:

  ```
  DISPLAY QCLUSTER(*) CLUSTER('Cluster.Name') ALL
  ```

  Each queue object instance shared in the cluster is called a *cluster queue*. Note that multiple cluster queues of the same name can be listed as shared by different queue managers in the cluster to facilitate the workload balancing, as described in 8.4, "Workload balancing" on page 209. Each cluster queue record contains information about the type of queue object, which queue manager hosts it within the cluster, and whether applications can put to the cluster queue or if it has been put disabled.

  Note: The same command issued on a partial repository might not return the full list of cluster queues shared within a cluster. This is due to the considerations discussed in 8.1.9, "Cluster subscriptions and publications" on page 192.

- DISPLAY CLUSQMGR:
  This command shows information about queue managers within the cluster known to that queue manager. For example, issuing the following command on a full repository for a cluster shows all the information known about all queue managers within a queue manager cluster:

  ```
  DISPLAY CLUSQMGR(*) CLUSTER('Cluster.Name') ALL
  ```

  Each record displayed in this output is called a *cluster queue manager* (`CLUSQMGR`) record. This includes the cluster queue manager record for the local queue manager where the command is issued. The information in this record includes the name and details about the cluster message channel that relates to the queue manager. This can be a cluster receiver channel object for the local queue manager, or any of the cluster sender channel types discussed in 8.1.6, "Cluster sender channels" on page 187. The type of cluster message channel is shown in the definition type (`DEFTYPE`) attribute of a cluster queue manager record. This includes status information about cluster sender channels from that queue manager to other queue managers in the cluster.

Note: Use the DISPLAY CHSTATUS command for information about channels established to that queue manager using its cluster receiver channel definition shared in the cluster. Multiple queue managers in the cluster can have active cluster channels to a queue manager, and these share the name of the cluster receiver channel. The cluster queue manager record might show an inactive status when channels to the queue manager are running.

The cluster queue manager record also contains details about the queue manager, including its QMID and whether it is a full repository in the cluster. The record for a full repository has a queue manager type (`QMTYPE`) attribute of repository (`REPOS`) and a partial repository has a queue manager type (`QMTYPE`) attribute of normal (`NORMAL`).

Note: The same command issued on a partial repository might not return the full list of queue managers within a cluster, but it should show all full repositories in the cluster. This is due to the considerations discussed in 8.1.9, "Cluster subscriptions and publications" on page 192.

If a manually defined cluster sender channel cannot be started to a full repository when joining a cluster, a cluster queue manager record is displayed with a name of the following form:

```
SYSTEM.TEMPQMGR.hostname(port)
```

Where *`hostname(port)`* is the connection name for the channel. This is because the name of the queue manager cannot be established until a channel is successfully started to that queue manager.

8.2.2 Viewing repository information in WebSphere MQ Explorer

The WebSphere MQ Explorer provides the Queue Manager Clusters folder in the navigator view in order to view cluster repository information.

Icons for clusters are automatically displayed under this folder whenever a queue manager that is a full repository for a cluster is connected to the WebSphere MQ Explorer under the Queue Managers folder. These queue managers can be hosted on the same machine as the WebSphere MQ Explorer, or remotely connected to the WebSphere MQ Explorer.

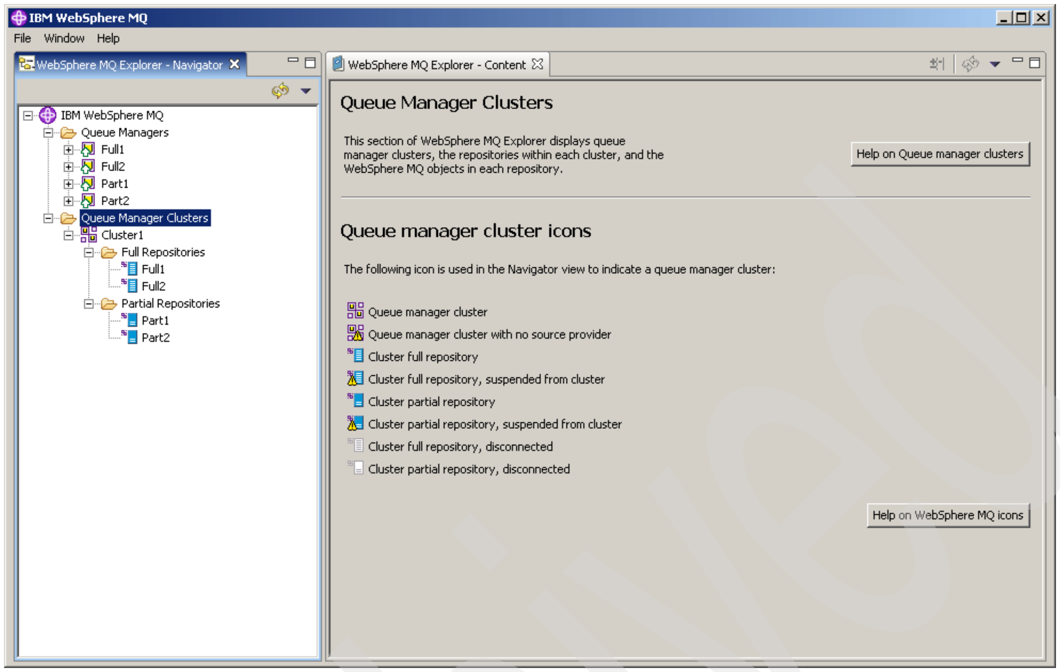
Note: The WebSphere MQ Explorer does not display information about clusters for which only partial repository queue managers are connected under the Queue Managers folder.

This is because these queue managers do not hold enough information about the cluster to reliably know about all queue managers within the cluster, as described in 8.1.9, “Cluster subscriptions and publications” on page 192.

Under each of these icons in the navigator tree, two folders are shown as follows:

- Full Repositories:
  Clicking this node displays a summary page about the queue managers that are members of the cluster and hold full repositories for the cluster. Under this node, an icon is displayed for each full repository queue manager within the cluster.
- Partial Repositories:
  Clicking this node displays a summary page about the queue managers that are members for the cluster and hold a partial repository for the cluster. Under this node, an icon is displayed for each partial repository queue manager within the cluster.

Figure 8-1 on page 198 shows the Queue Manager Clusters folder in the WebSphere MQ Explorer. The example cluster contains four queue managers, all local to the machine hosting the WebSphere MQ Explorer, of which two are full repositories and two are partial repositories.



Queue Manager Clusters

This section of WebSphere MQ Explorer displays queue manager clusters, the repositories within each cluster, and the WebSphere MQ objects in each repository.

Queue manager cluster icons

The following icon is used in the Navigator view to indicate a queue manager cluster:

- Queue manager cluster
- Queue manager cluster with no source provider
- Cluster full repository
- Cluster full repository, suspended from cluster
- Cluster partial repository
- Cluster partial repository, suspended from cluster
- Cluster full repository, disconnected
- Cluster partial repository, disconnected

Figure 8-1 The Queue Manager Clusters folder in the WebSphere MQ Explorer

If multiple full respository queue managers that host a full repository for the same cluster are connected to the WebSphere MQ Explorer, the WebSphere MQ Explorer chooses one of these queue managers to propagate the contents of the Full Repositories and Partial Repositories folders for that cluster. This queue manager is called the *cluster information source* for the cluster.

All full repositories in the queue manager holds the same information, as described in 8.1.9, “Cluster subscriptions and publications” on page 192. However, if a queue manager is disconnected from the network, or the cluster channels to that queue manager have been disabled, it is possible for the information contained within a full repository about the queue managers within a cluster to be incomplete. In these circumstances, a different queue manager, which also holds a full repository for the cluster and is connected under the Queue Managers folder, can be chosen as the cluster information source for the cluster. To do this, highlight the icon for the cluster in the navigator view, and click the **Select** button on the displayed Cluster content page.

Viewing the repository of an individual queue manager

After the cluster information source has been queried to gain information about the queue managers in the cluster, the information contained in the partial or full repository of each queue manager in the cluster can be displayed.

When a queue manager is selected under the Full Repositories or Partial Repositories folders for a particular cluster, a content page opens containing the information that queue manager holds in its repository.

The information on this page is queried *from the queue manager that is selected*, not the cluster information source.

If a remote queue manager is not displayed under the Queue Managers folder, the repository information held by that queue manager for the cluster can still be displayed. The icon for the queue manager is initially unavailable, and the tables in the display contain no data. To establish a connection to that queue manager through the cluster and propagate the repository information on this page, right-click the unavailable icon for the queue manager under the Partial Repositories or Full Repositories folder for the cluster in the navigator tree, and select **Connect To Queue Manager**.

Note: The connection to a queue manager that is not displayed under the Queue Managers folder used to display repository information is not performed directly to that queue manager. Instead, it is performed by sending and receiving messages to that queue manager through the cluster. The connection used by the WebSphere MQ Explorer is to the cluster information source selected for the cluster.

The queue manager for which repository information is to be displayed must have a running command server in order for this connection to be successful. Also, communication to that queue manager must be possible through the cluster to and from the full repository queue manager that is selected as the cluster information source for the cluster. If the target queue manager is not running, a command server is not running for that queue manager, an authorization failure occurs, or the communication through the cluster fails, the error message displayed is usually as follows:

```
Command server not responding within timeout period. (AMQ4032)
```

The information in the repository content page, displayed after successfully establishing a connection to that queue manager, is split into three tabs:

- Cluster Queues:
  This view shows information about all instances of queue objects shared within clusters as known to that queue manager. This is the same information displayed when issuing the DISPLAY QCLUSTER MQSC command against that queue manager.

  Each queue object instance shared in the cluster is called a *cluster queue*. Note that multiple cluster queues of the same name might be listed as shared by different queue managers in the cluster to facility the workload balancing described in 8.4, "Workload balancing" on page 209. Each cluster queue record contains information about the type of queue object, which queue manager hosts it within the cluster, and whether applications can put to the cluster queue or if it has been put disabled.

  The number of queue objects shared in the cluster, which that queue manager has knowledge of in its repository, is shown in the graphical display above the table.

  Each of these cluster queues has a row in the table. The table columns show the attributes of each cluster queue record. Double-clicking an individual record shows these attributes in a properties window.

  Note: It is not possible to alter the attributes of the queue object that the cluster queue record represents from this properties window. This includes where the queue manager is connected to the WebSphere MQ Explorer under the Queue Managers folder.

  In order to alter these attributes, the queue manager must be connected to the WebSphere MQ Explorer under the Queue Managers folder. Then, select the **Queues** folder under that queue manager in the navigator tree, and double-click the queue object, which is shared in the cluster, in the table.

- Cluster-sender channels:
  This view shows the same information as the DISPLAY CLUSQMGR MQSC command for all remote queue managers in the cluster. These are records about all queue managers in the cluster known to this queue manager with which this queue manager uses a cluster sender channel to communicate. We discuss the types of cluster sender channels in 8.1.6, "Cluster sender channels" on page 187.

  The table has a row for each queue manager known by this queue manager within the cluster. Each row represents a cluster sender channel that is running, or can run, to that queue manager. This is a cluster queue manager record. The cluster queue manager record contains details regarding the status of the channel and about the queue manager, including its QMID and whether it is a full repository in the cluster. The table columns show these details, which can be accessed in a properties window by double-clicking the row in the table. See 8.2.1, “Viewing repository information in MQSC” on page 195 for more details about the cluster queue manager records this tab displays.

Note: It is not possible to alter the attributes of any channel objects to which the cluster queue manager records refer from this properties window. This includes where the queue manager is connected to the WebSphere MQ Explorer under the Queue Managers folder.

These cluster queue manager records, although they show cluster sender channels, generally represent automatically defined channels based on the cluster receiver channel defined by a queue manager when it joined the cluster. See 8.1.5, “Cluster receiver channels” on page 185 for details. Cluster sender channel objects are only manually defined when joining a cluster to contact a full repository within the cluster. See 8.1.6, “Cluster sender channels” on page 187 for details.

- Cluster-receiver channels:
  This view shows the same information as the DISPLAY CLUSQMGR MQSC command for the local queue manager. This is usually a single record. It describes the information published by the queue manager when it joined the queue manager in its cluster receiver channel definition. See 8.1.5, “Cluster receiver channels” on page 185 for more information.

  The table usually contains a single row showing that this queue manager has published a single cluster receiver channel object to the cluster. This is a cluster queue manager record. The cluster queue manager record contains details regarding the status of the channel and about the queue manager, including its QMID and whether it is a full repository in the cluster. The table columns show these details, which can be accessed in a properties window by double-clicking the row in the table. See 8.2.1, “Viewing repository information in MQSC” on page 195 for more details about the cluster queue manager records this tab displays.

Note: It is not possible to alter the attributes of the channel object to which the cluster queue manager record refers from this properties window. This includes where the queue manager is connected to the WebSphere MQ Explorer under the Queue Managers folder.

We discuss the use of cluster receiver channels in joining a cluster in 8.1.5, “Cluster receiver channels” on page 185.

8.3 Actions on queue managers in a cluster

This section describes the common actions that are performed when administering or creating a queue manager cluster. This administration is not a complicated procedure, but should be followed correctly in order to ensure that incorrect information about a queue manager is not held by the other queue managers in the cluster and that communication to a queue manager in the cluster can be established.

8.3.1 Suspending and resuming a queue manager within a cluster

Suspending a queue manager in a cluster is not the same as removing it from the cluster. Suspending a queue manager in a cluster makes it unlikely that messages are delivered to the queues shared by that queue manager within the cluster. It significantly reduces the likelihood that the workload balancing algorithm on other queue managers chooses to deliver messages to queues hosted by that queue manager. We discuss workload balancing in 8.4, “Workload balancing” on page 209.

Suspending a queue manager in a cluster does not prevent messages that are targeted specifically for that queue manager from reaching that queue manager through the cluster.

After being suspended, a queue manager can be subsequently resumed.

Use one of the following methods to suspend and resume a queue manager in a cluster:

- Using the SUSPEND QMGR CLUSTER or RESUME QMGR CLUSTER MQSC commands to suspend or resume the queue manager in a single cluster.
- Using the SUSPEND QMGR CLUSNL or RESUME QMGR CLUSNL MQSC commands to suspend or resume the queue manager in multiple clusters.

- Using the WebSphere MQ Explorer:
  a. If the queue manager being suspended is a partial repository in the cluster, a full repository for the cluster must be connected under the Queue Managers folder in the navigator view.
  b. Expand the **Queue Manager Clusters** folder in the navigator view.
  c. Expand the icon in the navigator view with the same name as the cluster.
  d. Expand either the **Full Repositories** folder, or the **Partial Repositories** folder in the navigator view, depending on whether the queue manager is a full or partial repository in the cluster.
  e. Right-click the icon in the navigator view with the same name as the queue manager.
     - If the queue manager is to be suspended in the cluster, select **Suspend Cluster Membership**.
     - If the queue manager is to be resumed in the cluster, select **Resume Cluster Membership**. This option is only available if the queue manager is suspended in the cluster.

8.3.2 Resetting a queue manager's cluster membership

You can use the RESET CLUSTER MQSC command to flush all information held about a queue manager from the cluster to provide that queue manager with a clean start in the cluster without needing it to leave and rejoin the cluster.

This action is also available from the WebSphere MQ Explorer by right-clicking a queue manager icon under a particular cluster.

Note: Only take this action after reading the appropriate documentation in *WebSphere MQ Queue Manager Clusters*, SC34-6589. It is especially important to consider this action carefully if it is performed in relation to full repository queue managers in a cluster.

8.3.3 Steps to join a queue manager to a cluster

We describe the manual steps involved in adding a queue manager to an existing cluster, or creating a new cluster, in this section. For clusters that do not use namelists in the channel definitions or the repository attribute of queue managers, you can also perform these actions using wizards in the WebSphere MQ Explorer.

Using wizards in the WebSphere MQ Explorer

The WebSphere MQ Explorer wizards distinguish between the creation of a new cluster and adding a queue manager to an existing cluster. For either action, the queue managers involved must have already been created and started and must be connected under the Queue Managers folder in the WebSphere MQ Explorer.

To access the Create Cluster wizard:

1. Ensure that two full repository queue managers, which initially form the cluster, exist and are connected under the Queue Managers folder.
2. Right-click the **Queue Manager Clusters** folder in the navigator tree.
3. Select **New** → **Queue manager cluster**.

To access the Add to cluster wizard:

1. Ensure that a full repository for the cluster is connected to the WebSphere MQ Explorer under the Queue Managers folder.
2. Ensure that the queue manager that is joining the cluster is connected under the Queue Managers folder.
3. Right-click the cluster the queue manager is joining under the Queue Manager Clusters folder in the navigator tree.
4. Select **Add Queue Manager To Cluster**.

Manual steps

Use the following manual steps to add a queue manager to an existing cluster or create a new cluster:

1. Ensure that a listener is defined for the queue manager on a particular port and that the host name or IP address of the machine hosting the queue manager is known and available to all other queue managers in the cluster.

   Note: Do not use host name of `localhost` or the IP address of `127.0.0.1`, because these are only valid on the local machine.

2. Ensure that the connection name of a full repository within the cluster is known, including the host name or IP address of the machine hosting the queue manager and the port on which the listener is running. If you are performing these steps for one of the initial two full repository queue managers for the cluster, this is the connection name of the other queue manager that hosts, or can host, a full repository for the cluster.

Note: If this queue manager is going to replace an existing full repository within the cluster, we recommend that this is the connection name of the full repository that remains in the cluster.

3. If the queue manager is to host a full repository for a single cluster, set the repository (REPOS) attribute of the queue manager object to the name of the cluster.

   If the queue manager is to host full repositories for multiple clusters, define a namelist object with the names of all clusters specified in the NAMES attribute. Specify the name of this namelist object in the repository namelist (REPOSNL) attribute of the queue manager object.

4. Define a cluster receiver channel object to define how other queue managers within the cluster contact this queue manager:
   - Specify the connection name to use to contact the queue manager *joining the cluster* in the connection name (CONNAME) attribute.
   - If the cluster receiver channel object is to be used to join the queue manager to only one cluster, specify the name of the cluster in the cluster (CLUSTER) attribute.
   - Alternatively, if the same cluster receiver channel object is to be used to join the queue manager to multiple clusters, define a namelist object with the names of all clusters specified in the NAMES attribute. Specify the name of this namelist object in the cluster namelist (CLUSNL) attribute of the queue manager object.
   - Specify any additional attributes, such as the disconnection interval, batch size, or SSL configuration, that queue managers within the cluster should use when sending messages to this queue manager.

5. Define a cluster sender channel object to define a connection to an existing full repository in the cluster, or the other full repository queue manager that is defined:
   - Specify the connection name to use to contact the queue manager *hosting the full repository for the cluster* in the connection name (CONNAME) attribute.
   - If the remote full repository queue manager only hosts a full repository for one cluster that this queue manager joins, or if the full repository queue manager has different cluster receiver channel objects specified for each cluster for which it hosts a full repository, specify the name of the cluster in the cluster (CLUSTER) attribute.

– Alternatively, if the same cluster receiver channel object is defined on the full repository queue manager for multiple clusters that this queue manager is to join, define a namelist object with the names of all clusters specified in the `NAMES` attribute. Specify the name of this namelist object in the cluster namelist (`CLUSNL`) attribute of the queue manager object.

– Specify any additional attributes, such as the SSL configuration, that are required by the cluster receiver channel definition defined on the remote full repository queue manager.

Note: If the queue manager joining the cluster is a full repository queue manager, and more than one other full repository exists in the cluster, explicitly define cluster sender channel objects to *all* other full repository queue managers in the cluster.

6. The queue manager now becomes a member of the cluster and can share queue objects within the cluster by using cluster (`CLUSTER`) or cluster namelist (`CLUSNL`) attributes. Existing queue objects can be altered and new ones defined.

Note: Until the process of joining the cluster has completed successfully, entries of the following form are shown in output from DISPLAY CLUSQMGR commands or under the Clusters folder in the WebSphere MQ Explorer:

```
SYSTEM.TEMPQMGR.hostname(port)
```

You should do observe these entries if the full repository queue manager to which the explicit cluster sender is defined is running, has a listener running on the correct port, and has already joined the cluster. If you observe these, carefully check the attributes specified on the defined cluster channel objects and check that listeners are running for both queue managers on the correct ports. We provide general troubleshooting information in 12.3.3, “Troubleshooting cluster message channels” on page 331.

8.3.4 Steps for a queue manager to leave a cluster

We describe the manual steps involved in removing a queue manager from a cluster. For clusters that do not use namelists in the channel definitions or the repository attribute of queue managers, you can also perform this action using wizards in the WebSphere MQ Explorer.

Using wizards in the WebSphere MQ Explorer

Use the following steps to access the Remove Queue Manager from Cluster wizard through the WebSphere MQ Explorer:

1. Find the icon for the queue manager under the icon for the cluster in the Queue Manager Clusters folder of the navigator view. It is either in the Full Repositories or Partial Repositories folder.
2. Right-click this icon and select **Remove Queue Manager From Cluster**.

Manual steps

Use the following manual steps to remove a queue manager from a cluster:

1. If the queue manager hosts a full repository for the cluster, ensure that two full repositories remain available in the cluster.

   Note: If this is not the case, join a new queue manager to the cluster as a full repository before removing the existing full repository queue manager.

2. If the queue manager hosts a full repository for the cluster, inform all queue managers in the cluster that the queue manager no longer hosts a full repository. To do this:
   - If the queue manager only hosts a repository for one cluster, set the repository (`REPOS`) or repository namelist (`REPOSNL`) attribute of the queue manager object to blank, depending on which contains a value.

     Note: In MQSC, enclose the blank value in single quotation marks:

     ```
     ALTER QMGR REPOS(' ')
     ```

   - If the queue manager hosts a full repository for multiple clusters and is only to be removed from one, alter the namelist object specified in the repository namelist (`REPOSNL`) attribute of the queue manager to remove the cluster from the list of names.
3. Suspend the queue manager in the cluster, as described in 8.3.1, “Suspending and resuming a queue manager within a cluster” on page 202.
4. Inform the queue managers in the cluster that the queue manager is leaving the cluster. To do this:
   - If the cluster receiver used to join the cluster was used to join the cluster to only one cluster, set the cluster (`CLUSTER`) or cluster namelist (`CLUSNL`) attribute of that cluster receiver channel object to blank, depending on which contains a value.

Note: In MQSC, enclose the blank value in single quotation marks:

```
ALTER CHANNEL('TO.QmgrName') CHLTYPE(CLUSRCVR) CLUSTER(' ')
```

– If the cluster receiver used to join the cluster was used to join multiple clusters and is only to be removed from one, alter the namelist object specified in the cluster namelist (CLUSNL) attribute of the cluster receiver channel object to remove the cluster from the list of names.

5. Issue a stop channel command against the cluster receiver channel object used to join the queue manager to the cluster. For example, in MQSC use the following command:

   ```
   STOP CHANNEL('TO.QmgrName')
   ```

   Note: After performing this step, any messages that are sent to this queue manager though the cluster become stuck on the cluster transmit queues of the sending queue managers. Therefore, wait for the messages to stop flowing to the queue manager before issuing the command. Because the queue managers in the cluster have been informed of this queue manager's removal, workload balancing does not send new messages to this queue manager.

6. If the cluster or cluster namelist was set to blank during these steps, and the cluster receiver channel is not required to rejoin this queue manager to a cluster, the cluster receiver channel object can now be deleted.
7. If the cluster sender channel object used to join the queue manager to the cluster was only used to join the queue manager to one cluster, it can also be stopped and deleted at this stage. Otherwise, follow the same procedure as was followed for the cluster receiver channel to remove the reference to the cluster from the definition of the cluster sender channel object or the namelist it references.
8. Issue a refresh cluster command against the queue manager for the cluster that the queue manager is leaving. This ensures that the information about the cluster is cleared from the repository held by the queue manager.

Note: If the queue manager that left the cluster held a full repository for that cluster, it is possible that partial repositories had explicitly defined cluster sender channels to that queue manager. Define new cluster sender channel objects on these queue managers to point to a full repository that remains in the cluster and remove the old cluster sender channel object. Otherwise, these queue managers will be unable to leave and rejoin the cluster if required in the future. This includes performing a REFRESH CLUSTER command with REPOS(YES) specified.

8.4 Workload balancing

The most powerful feature provided by a queue manager cluster is workload balancing. This enables the same service to be hosted through multiple queue managers in the cluster and for the requests from applications to be workload balanced across those service instances. This is transparent to the operation of the application accessing the service.

The capacity of services hosted by a cluster can be scaled efficiently by adding resources to the infrastructure. New machines hosting queue managers and services are automatically sent new requests by the applications that connect to queue managers in that cluster to request services.

Workload balancing can also provide high service availability, because it automatically detects if a queue manager becomes unavailable. New requests for services are workload balanced across the remaining available queue managers in the cluster.

No special configuration is required in a cluster in order to benefit from workload balancing. In order to add an instance of a service to the cluster, a new queue manager joins the cluster and shares an instance of the queue that represents that service identified by the queue name. All queue managers within the cluster that have accessed, or subsequently access, that queue name within the cluster are automatically notified of the new instance. These queue managers can then begin sending messages to that instance.

The choice of which instance of a queue to which to send a particular message is made by the queue manager, usually a partial repository, to which the application sending that message is connected.

The first time a queue of a particular name is accessed by any application connected to a queue manager, it subscribes to information about the instances of that queue hosted in the cluster from a full repository for that cluster. It is then automatically informed of changes to those instances or the removal or addition of instances.

In order to benefit from workload balancing, an application must not specify the name of a particular queue manager when it opens a queue using an MQOPEN call.

Note: If a queue manager name *is* specified when opening a queue, the uses of queue objects described in 8.1.7, "Sharing queue objects within clusters" on page 189 can be used to allow workload balancing to still occur. This is often the case for messages that arrive from a queue manager outside of a cluster where the receiving MCA opens a queue with the queue manager name specified in the transmission queue header.

8.4.1 Bind on open and bind not fixed

In the execution of a particular application, the application might send a number of messages to the same service, identified by a queue name, within a cluster.

Often, each message can be delivered to a different instance of the service within the cluster with no adverse effect on the operation of the application requesting that service. If the service being contacted is using a request/reply model, each message sent by the requester is marked with an explicit destination for the service to reply to, so no context is necessary. This form of operation is very efficient for workload balancing and can increase the tolerance of the application to an individual service instance becoming unavailable.

However, messages exchanged between two applications can be conversational, with requests followed by responses followed by requests based on the context established in the previous request. The relation between these messages is called *message affinity*. Due to message affinity, an application might need to *bind* to a particular instance of a service within the cluster.

To provide for this, an application can explicitly specify one of the following options in the options provided to the queue manager when opening the queue.

- Bind on open:
  All messages sent using the object handle returned from the MQOPEN call, until the MQCLOSE is issued, are sent to the same instance of a queue in the cluster. Workload balancing is only performed once when the queue is first opened by the application.
- Bind not fixed:
  Workload balancing occurs each time a message is put by the application using the object descriptor.

The default behavior is determined by the default bind (`DEFBIND`) attribute of the cluster instance of the queue, which is chosen during the MQOPEN call. This attribute is specified on the queue object shared in the cluster on the queue manager that created that object and shared it in the cluster.

> **Note:** Existing applications that use the MQPUT1 call, or perform an MQOPEN and MQCLOSE with each message put, cannot benefit from bind on open functionality. These applications need to be modified to perform multiple puts with the same object handle, or preferably to remove the message affinity.

8.4.2 The workload balancing algorithm

Each time a queue manager chooses a destination for messages within a cluster based on a queue name, the workload balancing algorithm of the queue manager is invoked.

At a basic level, with a default configuration and channels to all queue managers in the cluster successfully established, this workload balancing algorithm can be considered to perform a round-robin type distribution of messages. Therefore, if 1000 messages are put by an application with bind not fixed, and 10 queues of the name specified are hosted by 10 queue managers within the cluster, approximately 100 messages arrive at the applications hosted on that queue on each of those 10 queue managers.

However, this is not exact and is affected by many considerations. This section provides an overview of how the workload balancing algorithm operates and can be customized using the facilities of WebSphere MQ V6.0. For full details of the decision process made by the workload balancing algorithm each time it is called, refer to *WebSphere MQ Queue Managers Clusters*, SC34-6589.

8.4.3 Destination sequence numbers

In order to perform a round-robin type algorithm, a *destination sequence number* is held locally to a queue manager for each queue manager it has knowledge of in the cluster.

> **Note:** This is performed on a per cluster channel basis, but under most circumstances, each cluster channel known to a queue manager for a cluster reflects one queue manager in the cluster.

The cluster workload algorithm preferentially chooses a suitable destination queue manager with the lowest destination sequence number.

This destination sequence number is incremented each time a message is sent to that queue manager through the cluster. It is not held on a per queue basis, so messages sent to a queue instance of one name on a queue manager affect workload balancing of messages sent through the same cluster to queue instances of other names on that queue manager. This includes where commands are sent to other queue managers in the cluster to maintain information within the repositories and where applications specify a particular queue manager when sending messages through a cluster.

In the default configuration, this value is incremented by the same amount each time a message is sent. Therefore, in the simple case previously described, an even distribution of messages occurs.

However, there are a number of other factors that affect the operation of the workload balancing algorithm. The rest of this chapter discusses these considerations.

These are placed in the order in which they are considered by the workload balancing algorithm. The channel sequence number is the last factor to be considered, because it provides balancing only between the suitable candidate destinations for the message.

Note: The sequence number is maintained by a WebSphere MQ V6.0 queue manager while it is running to account for changes such as resuming queue managers in the cluster. A reset of all channel sequence numbers can be forced by restarting the queue manager.

8.4.4 Put disabling queues

When a queue shared in a cluster is put disabled, this is published to all interested queue managers within the cluster.

The workload balancing only attempts to send messages to a put disabled cluster queue instance if no other instances are available within the cluster.

Note: If messages are explicitly sent to that queue manager, or no other cluster queue instances of the correct name are available in the cluster, messages are still delivered to the queue manager. These messages are processed by the receiving MCA, as described in 7.4.1, “Message transmission” on page 167.

8.4.5 Workload balancing and locally hosted queues

If an application is connected to a queue manager and puts a message specifying the name of a local queue object on that queue manager, the default behavior is to always use that local instance as long as it is not put disabled.

This default behavior can be overridden on WebSphere MQ V6.0 to allow workload balancing to occur, treating the locally hosted queue in the same way as queues hosted remotely in the cluster.

The default behavior for all queues hosted on a queue manager is overridden using the cluster workload use queue (`CLWLUSEQ`) attribute of the queue manager object.

The queue manager wide default can be overridden on a queue by queue basis using the cluster workload use queue (`CLWLUSEQ`) attribute of a local queue object.

8.4.6 Ranking queue managers and queues

In WebSphere MQ V6.0, queue managers within a cluster can be ranked by specifying or changing the cluster workload rank (`CLWLRANK`) attribute on the cluster receiver channel, which was used by a queue manager to join a cluster.

Queue instances hosted on a queue manager with a higher rank are *always* chosen over instances hosted on a queue manager with a lower rank unless no other put enabled queue instances of the correct name are known in the cluster.

Individual queue instances can be also be ranked using the cluster workload rank (`CLWLRANK`) attribute on the queue object that is shared in the cluster. Ranking of individual queues is considered after ranking of queue managers by the cluster workload algorithm.

Ranking can force the final destination for messages in the cluster. This is useful when interconnecting multiple clusters and defining routes between those clusters. Ranking is useful in order to specify how messages should be routed through interconnected clusters to get to a preferred destination.

8.4.7 Suspending queue managers in the cluster

If a queue manager is suspended in a cluster, as described in 8.3.1, “Suspending and resuming a queue manager within a cluster” on page 202, the workload balancing algorithm chooses other queue managers in preference to that queue manager.

This is useful before performing planned maintenance on that queue manager to recommend to applications that they do not route messages to the queue manager while it is unavailable.

However, if no other queue instances of the correct name are available, messages are still sent to a suspended queue manager.

8.4.8 Channel status

Queue instances hosted on queue managers to which channels are already running or have become naturally inactive are chosen in preference to other queue managers.

If no such instances are found, instances on queue managers to which channels are in the process of starting are chosen.

If it has still not been possible to find instances, queue managers to which channels that have failed to start previously but are in the process of retrying the connection are chosen. Messages wait on the cluster transmission queue until the channel restarts.

If the only instances are hosted on queue managers to which channels must be manually restarted, such as stopped channels, messages must be sent to those queue managers and remain on the transmission queue until the channels are started manually.

This enables the cluster to automatically route around individual failures of queue managers within the cluster.

8.4.9 Prioritizing queue managers and queues

In WebSphere MQ V6.0, queue managers within a cluster can be prioritized by specifying or changing the cluster workload priority (`CLWLPRTY`) attribute on the cluster receiver channel, which was used by a queue manager to join a cluster.

Queue instances hosted on a queue manager with a higher priority are usually chosen over instances hosted on a queue manager with a lower priority.

However, instances hosted on a queue manager with a lower priority are chosen if a queue manager is considered unavailable for some reason. For example, queue managers might be suspended in the cluster or channels to queue managers might have failed.

Individual queue instances can be also prioritized using the cluster workload priority (`CLWLRANK`) attribute on the queue object that is shared in the cluster.

Prioritization of individual queues is considered after prioritization of queue managers by the cluster workload algorithm.

Prioritization allows secondary, or failover, resources to be allocated for hosting services. These resources might be in a geographically remote location, thus affecting performance. They might be designated as primary resources for other services in the system, so sending requests to them under normal operation affects the performance of their primary services. If a failure occurs for the resources normally hosting the service, they are able maintain service availability.

8.4.10 Limiting cluster connections from a queue manager

If a large number of applications access a service in a system, and a number of instances of the service exist to manage this workload, it might not be efficient for all applications to workload balance over all instances. This can cause a large number of channels to be active to each queue manager hosting the service.

WebSphere MQ V6.0 allows the number of instances over which a queue manager workload balances to be limited. Then, workload balancing using a destination sequence number only occurs over these instances.

A limit is configured on the maximum number of channels that are established from a queue manager to other queue managers in a cluster. This is configured using the cluster workload manager recently used channel (`CWMRUC`) attribute on the queue manager object.

By enabling this limit on all queue managers accessing a service, the number of channels established concurrently to a queue manager hosting a service can be reduced.

8.4.11 Weighting queue managers

Some machines hosting a service might be more powerful, and thus able to service more requests than other machines hosting the same service within the system.

In WebSphere MQ V6.0, queue managers within a cluster can be weighted between 1 and 99 by specifying or changing the cluster workload weight (`CLWLWGHT`) attribute on the cluster receiver channel, which was used by a queue manager to join a cluster.

The cluster workload algorithm sends more messages to queue managers with a higher weight than those with a lower weight. The number of messages sent are proportional to the weight.

The algorithm achieves this by increasing the destination sequence number of each queue manager by a value related to the weight. A higher weight results in less being added to the destination sequence number when each message is sent. The increment is 1000 divided by the weight of the queue manager.

For example, two available queue managers have weights of 60 and 20, and the same destination sequence number. Four messages are workload balanced between a queue name hosted by both queue managers. Three of these messages are sent to the queue manager with a weight of 60, and only 1 is sent to the queue manager with a weight of 20. This is because destination sequence numbers both increase by 50:

- (1000/60) x 3 = 50
- (1000/20) x 1 = 50

9

Hands-on introduction to messaging with WebSphere MQ

In this chapter, you create, start, and administer local WebSphere MQ queue managers. You learn the use of both the WebSphere MQ Explorer and WebSphere MQ control commands. We provide equivalent steps for common queue manager configuration tasks using the WebSphere MQ Explorer and MQSC script commands. You perform both point-to-point and publish/subscribe messaging using the local WebSphere MQ infrastructure you build. This infrastructure provides a queue manager hosting a service with a request/reply interface, triggered to start when messages arrive on a queue.

We discuss the following topics:

- Overview of the hands-on chapters of this book
- Environment setup
- Messaging with a local queue manager
- Host a request/reply service on a queue
- WebSphere MQ publish/subscribe with JMS

9.1 Overview of the hands-on chapters of this book

The hands-on introduction to WebSphere MQ in this book is split into two chapters:

- In this chapter, you create and administer individual queue managers. You provide and access services through these queue managers, using point-to-point and publish/subscribe messaging models.
- In Chapter 10, “Hands-on guide to building WebSphere MQ infrastructure” on page 265, you connect multiple queue managers together over a TCP/IP network. You build a hub and spoke infrastructure and then a queue manager cluster. Through these infrastructures, you provide and access the same services as in this chapter.

Throughout the hands-on introduction, you use the administration interfaces and sample applications provided by WebSphere MQ.

9.1.1 Administration of queue managers

Many administration tasks can be performed using either the WebSphere MQ Explorer or MQSC scripting interface. This book provides equivalent steps using both methods.

We recommend that those without previous WebSphere MQ experience use the WebSphere MQ Explorer. However, the MQSC steps can be useful later as your knowledge progresses in order to build scripts based on the tasks in this chapter. For example, your change control procedures might require an MQSC script to be submitted for any changes to a production environment.

Note: We use lowercase naming conventions, because these are well-suited for administration with the WebSphere MQ Explorer.

When using MQSC commands, ensure that any attribute values and object names that contain lowercase characters are enclosed in single quotation marks. Regularly using quotation marks is general good practice in MQSC to avoid confusion.

9.1.2 WebSphere MQ sample programs

WebSphere MQ is supplied with a variety of sample programs.

These sample programs are provided in source code form, and many samples are also provided precompiled and ready to use.

This book uses the samples provided in the C programming language, using the message queue interface (MQI), to demonstrate point-to-point messaging. Similar samples are provided for other programming languages with the WebSphere MQ product.

We use the samples provided in the Java language, using the JMS standardized application programming interface (API), to demonstrate publish/subscribe messaging. Very similar principles apply to the IBM Message Service (XMS) client.

Publish/subscribe messaging can also be performed by building WebSphere MQ publish/subscribe commands and issuing them against the WebSphere MQ publish/subscribe broker. You can be perform this using any of the point-to-point messaging APIs provided for use with WebSphere MQ.

Application programmers are encouraged to inspect the source code of the samples provided by WebSphere MQ when beginning to implement a solution based on the WebSphere MQ product. Many of the samples provided in other programming languages by WebSphere MQ match the function of the C samples used in these steps. Therefore, using these steps and the infrastructures developed as a starting point for development and testing can be beneficial.

9.2 Environment setup

This chapter assumes a WebSphere MQ V6.0 server installation on your desktop Microsoft Windows or Linux machine. However, with the exception the WebSphere MQ Explorer steps, you can also perform the steps in this chapter with remote access to a UNIX machine that has a WebSphere MQ V6.0 server installation.

Note: If you use a Linux installation of WebSphere MQ, follow the UNIX instructions where a platform-specific choice is given.

9.2.1 WebSphere MQ V6.0 installation

For information about performing a WebSphere MQ server installation, refer to the *WebSphere MQ V6.0 Quick Beginnings* guide appropriate to your platform:

- *WebSphere MQ for Windows V6.0 Quick Beginnings*, GC34-6476
- *WebSphere MQ for Linux V6.0 Quick Beginnings*, GC34-6480

Install all optional components and prerequisites.

9.2.2 WebSphere MQ administrator privileges

The user under which you log in to the machine must have WebSphere MQ administrator privileges.

- On Windows:

  Your user identifier must be a member of either the following groups:

  – Administrators

  – mqm

- On UNIX:

  Your user identifier must be a member of the following group:

  – mqm

9.2.3 Accessing the WebSphere MQ samples

The WebSphere MQ samples need to be in your operating system search path. For WebSphere MQ for Windows installations, this is performed automatically.

The WebSphere MQ samples are installed into the following directory:

- Windows:

  C:\Program Files\IBM\WebSphere MQ\Tools\c\Samples\Bin

- UNIX, except IBM AIX 5L:

  /opt/mqm/samp/bin

  Add this to the path for the current terminal session:

  ```
  PATH=$PATH:/opt/mqm/samp/bin
  export PATH
  ```

- IBM AIX 5L:

  /usr/mqm/samp/bin

  Add this to the path or the current terminal session:

  ```
  PATH=$PATH:/usr/mqm/samp/bin
  export PATH
  ```

9.2.4 Java considerations

The steps in 9.5, “WebSphere MQ publish/subscribe with JMS” on page 255 require a Java Development Kit (JDK™) installation. A JDK is supplied in the prereqs directory of the WebSphere MQ V6.0 installation media.

If a JDK is not available, all publish/subscribe-related steps in this book must be skipped.

9.3 Messaging with a local queue manager

This section helps you to become familiar with the graphical and command line interfaces used to create, start, end, and delete queue managers. It also shows how you can put and get test messages to queues and view the contents of queues using the WebSphere MQ Explorer and the WebSphere MQ sample programs.

Future sections assume some familiarity with these interfaces and samples as developed in this section.

9.3.1 Create a default queue manager on the machine

This step creates a new queue manager on the machine and makes it the default queue manager for the machine.

This step can either be performed using the WebSphere MQ Explorer Create Queue Manager wizard, or the **`crtmqm`** WebSphere MQ control command.

Using the WebSphere MQ Explorer

Perform the following steps:

1. Right-click the **Queue Managers** folder in the navigator view, and select **New** → **Queue Manager**. The Create Queue Manager wizard opens, showing Enter Basic Values (Step 1)
2. Type `host1/qm1` into the Queue manager name field.
3. Select the **Make this the default queue manager** option.
4. Leave the other fields with their default values, and click **Next**. Do not click Finish. The wizard shows Enter log values (Step 2).
5. Leave the fields with their default values, and click **Next**. Do not click Finish. The wizard shows Enter configuration options (Step 3).
6. Clear the **Start queue manager** check box.

Note: The Start queue manager check box can be used to automatically perform the steps 9.3.2, "Start the default queue manager on the machine" on page 223.

An Auto start queue manager option is provided on Windows only to cause the queue manager to be automatically started when the machine is turned on or restarted.

7. Leave the other fields with their default values, and click **Next**. Do not click Finish. The wizard shows Enter listener options (Step 4).

Note: If you have a queue manager already created on the machine with a listener configured on the well-known TCP/IP port for WebSphere MQ, this page might show "The port is already used by another WMQ Listener."

8. Clear the **Create listener configured for TCP/IP** check box.

Note: This option can be used to automatically perform the steps in 10.2.1, "Create and start a listener" on page 267.

9. Leave the other fields with their default values, and click **Next**. Do not click Finish. The wizard shows Enter explorer options (Step 5).
10. Leave the fields with their default values, and click **Finish**.
11. A status window opens with the title Creating Queue Manager "host1/qm1".

Note: You can click **Show details** on this window to see the WebSphere MQ control commands being executed by the WebSphere MQ Explorer to create the queue manager. If you do so, the status windows need to be closed manually after the original window shows Finished.

12. After this task completes, notice that the queue manager is shown in the navigator view under the Queue Managers folder with the correct icon for a stopped local queue manager.

Using WebSphere MQ control commands

Execute the following command to create a queue manager with a default configuration and make it the default queue manager for the machine.

```
crtmqm -q host1/qm1
```

Note: If the `-q` option is not specified, the queue manager is not made the default queue manager for the machine.

The expected output from this command is:

```
WebSphere MQ queue manager created.
Creating or replacing default objects for host1/qm1.
Default objects statistics : 43 created. 0 replaced. 0 failed.
Completing setup.
Setup completed.
```

Issuing the following command checks that the queue manager has been created:

```
dspmq
```

The following line is expected in this output:

```
QMNAME(host1/qm1)                         STATUS(Ended immediately)
```

9.3.2 Start the default queue manager on the machine

This step starts the queue manager created in the previous step. It can be performed using either the WebSphere MQ Explorer or the **`amqmdain`**/**`strmqm`** WebSphere MQ control commands.

The WebSphere MQ control commands do not require the queue manager name, because it has been created as the default queue manager for the machine.

Using the WebSphere MQ Explorer

Right-click the icon for the queue manager host1/qm1 in the navigator view, and select **Start**. The same form of progress window opens as was shown when creating the queue manager.

After this task completes, notice that the queue manager is shown in the navigator view under the Queue Managers folder with the correct icon for a running local queue manager.

Using WebSphere MQ control commands

The control commands used on UNIX and Windows differ as follows:

- Windows:

  ```
  amqmdain qmgr start
  ```

Note: This starts the queue manager in a way that allows you to log off and then log back on and for the queue manager to continue running. The WebSphere MQ control command under the UNIX heading works, but logging off ends the queue manager.

On WebSphere MQ for Windows V5.3, use:

```
amqmdain start host1/qm1
```

- UNIX:

  ```
  strmqm
  ```

  The expected output from this command is:

  ```
  WebSphere MQ queue manager 'host1/qm1' starting.
  5 log records accessed on queue manager 'host1/qm1' during the log replay
  phase.
  Log replay for queue manager 'host1/qm1' complete.
  Transaction manager state recovered for queue manager 'host1/qm1'.
  WebSphere MQ queue manager 'host1/qm1' started.
  ```

Note: If you get the following output, this probably indicates that the queue manager was not created as the default queue manager for the machine:

```
AMQ8118: WebSphere MQ queue manager does not exist.
```

To check this, or to make the queue manager the default, perform the following steps:

- Using the WebSphere MQ Explorer (on Windows):
  a. Right-click **IBM WebSphere MQ** in the navigator view, and select **Properties**.
  b. Inspect or change the Default queue manager name field. For these steps, it should contain:
     ```
     host1/qm1
     ```
  c. Click **OK**.
- Using the mqs.ini file on UNIX:
  a. Open the mqs.ini file in a text editor, such a vi or emacs. The mqs.ini file is located in the following location:
     /var/mqm/mqs.ini
  b. Search for the following line:
     ```
     DefaultQueueManager:
     ```
  c. If that line does not exist, create it at the end of the file.
  d. The next line after this line is:
     ```
     Name=host1/qm1
     ```
  e. Edit or add this line as appropriate.

9.3.3 Define a new locally hosted queue

This section shows how to define a new local queue object on the queue manager. A local queue object represents an actual queue hosted by that queue manager to which messages can be put and got.

You can perform this task using the WebSphere MQ Explorer or by issuing a DEFINE MQSC command from the **`runmqsc`** command. The **`runmqsc`** command is used to issue MQSC commands against a queue manager.

Using the WebSphere MQ Explorer

Perform the following steps:

1. Expand the icon for the queue manager host1/qm1 in the navigator view, which displays a Queues folder under the queue manager.
2. Right-click the **Queues** folder and select **New** → **Local Queue**. The Create Local Queue wizard opens showing “Enter a name” and a red cross.
3. Enter `queue1` into the Name field.
4. Click **Next**. Do not click Finish. The Change the properties of the new Local Queue window opens. This is the window from where the initial attributes of the local queue object can be specified.
5. Enter the following value in the Description field:

   ```
   Redbook example queue1: Newly defined
   ```

6. Click **Finish**. A progress window might briefly open. The result window opens, containing the following message:

   ```
   The object was created successfully. (AMQ4148)
   ```

Using MQSC commands

Perform the following steps:

1. Start an interactive MQSC session for the default queue manager using the following command:

   ```
   runmqsc
   ```

2. The interactive MQSC session starts and waits for input with a flashing cursor. The initial output is as follows:

   ```
   5724-H72 (C) Copyright IBM Corp. 1994, 2004.  ALL RIGHTS RESERVED.
   Starting MQSC for queue manager host1/qm1.
   ```

   The session is ready to accept input; you do not see a subsequent message saying that the session has started.

   Note: If the queue manager has not been started, or has not been specified as the default, you see the following output and a normal operating system prompt is displayed:

   ```
   5724-H72 (C) Copyright IBM Corp. 1994, 2004.  ALL RIGHTS RESERVED.
   AMQ8146: WebSphere MQ queue manager not available.

   No MQSC commands read.
   No commands have a syntax error.

   All valid MQSC commands were processed.
   ```

3. Issue the following command to create the queue:

```
DEFINE QLOCAL('queue1') +
      DESCR('Redbook example queue1: Newly defined')
```

> **Note:** It is important to use the single quotation marks, as shown in the previous command. This allows the name and description to contain lowercase characters.
>
> Note that the + symbol is the *last* character on the first line, and that it is preceded by a blank space after the closing parenthesis. This allows the second line to be part of the same MQSC command.

The output is as follows:

```
AMQ8006: WebSphere MQ queue created.
```

> **Note:** If you make multiple attempts to create the queue, subsequent attempts might fail with the following message:
>
> ```
> AMQ8150: WebSphere MQ object already exists.
> ```
>
> If this occurs, add the REPLACE attribute to the command to replace any existing definition. For example:
>
> ```
> DEFINE QLOCAL('queue1') REPLACE +
>       DESCR('Redbook example queue1: Newly defined')
> ```

4. Exit the interactive MQSC session using the following command:

```
END
```

You are returned to an operating system prompt. Output is displayed showing a summary of the commands you executed during that session, for example:

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

9.3.4 Display the attributes of the newly created queue

This step shows that the description (DESCR) attribute specified has been stored in the definition of the object. All other attributes contain their default values. For example, the (default) persistence (DEFPSIST) attribute is set to not persistent (NO). The number of messages on the queue, as reflected in the read-only current queue depth (CURDEPTH) attribute, is zero for the newly created queue.

You can perform this step using the WebSphere MQ Explorer, showing the schemes feature provided by the WebSphere MQ Explorer. It can also be performed using **`runmqsc`**, demonstrating how wildcards and attribute values can be used in MQSC DISPLAY commands.

Using the WebSphere MQ Explorer

Perform the following steps:

1. Highlight the **Queues** folder under queue manager host1/qm1 in the Queue Managers folder in the navigator view.
2. An entry in the table is displayed for the queue queue1. The table columns display the attribute values. Scroll the table to the right to view the Description, Persistence, and Current queue depth attributes.
3. You can change the default order of these columns, or save your own set of customized columns using WebSphere MQ *schemes*. At the bottom of the table, you see the following text displayed:

   Scheme: Standard for Queues - Distributed

   This shows that the default column scheme is being used, and that this queue manager is not a WebSphere MQ for z/OS queue manager. The term *distributed* generally represents all non-mainframe platforms.

   To the right of this text is an arrow. Click this arrow to open a menu from which you can apply existing schemes to the table, edit the current scheme, or manage the schemes.

   Click **Manage Schemes** to access a window where you can create your own schemes or edit the default scheme for all tables showing queues on your WebSphere MQ Explorer.

   For example, you can move the Description, Persistence, and Current queue depth columns to near the beginning of the column order.

Using an MQSC command

Perform the following steps:

1. Start an interactive MQSC session for the default queue manager using the following command:

   ```
   runmqsc
   ```

2. Issue the following MQSC command to list all local queue objects defined:

   ```
   DISPLAY QLOCAL(*)
   ```

In the output from this command, which summarizes *all* local queue objects, `queue1` is displayed as follows:

```
AMQ8409: Display Queue details.
   QUEUE(queue1)                          TYPE(QLOCAL)
```

3. Issue the following MQSC command to display all attributes of the queue:

   ```
   DISPLAY QLOCAL('queue1') ALL
   ```

   You see a large number of attributes displayed in this output.

   Note: It is important you use the single quotation marks as shown in the previous command. This allows the name of the object to contain lowercase characters. If you do not do this, you see the following output:

   ```
   AMQ8147: WebSphere MQ object QUEUE1 not found.
   ```

4. Limit the output to only the description, default persistence, and current queue depth attributes using the following MQSC command:

   ```
   DISPLAY QLOCAL('queue1') DEFPSIST DESCR CURDEPTH
   ```

   The output is as follows:

   ```
   AMQ8409: Display Queue details.
      QUEUE(queue1)                          TYPE(QLOCAL)
      CURDEPTH(0)                            DEFPSIST(NO)
      DESCR(Redbook example queue1: Newly defined)
   ```

5. Issue the following command to exit the MQSC interactive session:

   ```
   END
   ```

Note: You can use the DISPLAY QUEUE MQSC command to display the attribute of any queue object, regardless of its type. For example, the following command shows all queue objects defined on a queue manager:

```
DISPLAY QUEUE(*)
```

9.3.5 Alter the attributes of a queue object

This step alters the attributes of the previously defined local queue object. For this step, you can use the WebSphere MQ Explorer or MQSC commands.

Using the WebSphere MQ Explorer

Perform the following steps:

1. Highlight the **Queues** folder under queue manager host1/qm1 in the Queue Managers folder in the navigator view.

2. Right-click the entry for queue1 in the table and select **Properties**, or double-click the entry in the table. This displays the properties window for the local queue object.
3. Enter the following value in the Description field:

   ```
   Redbook example queue1: Altered description
   ```

4. Click **OK**. A progress window opens and then closes.
5. Note the change in the description attribute of the local queue object in the queues table.

Using MQSC commands

Perform the following steps:

1. Start an interactive MQSC session for the default queue manager using the following command:

   ```
   runmqsc
   ```

2. Issue the following MQSC command to alter the description attribute of the queue:

   ```
   ALTER QLOCAL('queue1') +
         DESCR('Redbook example queue1: Altered description')
   ```

   Note: It is important to use the single quotation marks as shown in the previous command. This allows the name of the object and the description to contain lowercase characters. If you do not do this, you see the following output:

   ```
   AMQ8147: WebSphere MQ object QUEUE1 not found.
   ```

3. Display the updated attribute of the queue using the following command:

   ```
   DISPLAY QLOCAL('queue1') DESCR
   ```

4. Issue the following command to exit the MQSC interactive session:

   ```
   END
   ```

9.3.6 Put test messages onto this queue

This step places messages, containing your own custom text, onto the queue previously created. For this step, you can use the WebSphere MQ Explorer the **`amqsput`** WebSphere MQ sample program against the default queue manager on the machine.

After you have put these messages, you see the current queue depth (`CURDEPTH`) attributes on the queue change to the number of messages put to the queue. In

9.3.4, “Display the attributes of the newly created queue” on page 227, we describe how to view this attribute.

Using the WebSphere MQ Explorer

Perform the following steps:

1. Highlight the **Queues** folder under queue manager host1/qm1 in the Queue Managers folder in the navigator view.
2. Right-click the row in the table for queue1 and select **Put Test Message**. This opens the Put test message window.
3. Type a message into the Message data field.
4. Click **Put message**. The Message data field becomes blank. You can enter multiple test messages, clicking **Put message** after each one.
5. Click **Close**. The window closes.

Using the WebSphere MQ sample programs

Note: Ensure that the WebSphere MQ sample programs have been added to your operating system command search path.

Perform the following steps:

1. Execute the following command:

   ```
   amqsput queue1
   ```

   The following output is displayed, and the command waits for user input:

   ```
   Sample AMQSPUT0 start
   target queue is queue1
   ```

2. Type a message and press Enter. You can enter multiple messages, pressing Enter after each one.
3. Leave a blank line, and press Enter to exit.

Note: If you receive output similar to the following output (including where 2059 is replaced by 2058), check that the queue manager is correctly configured as the default, as described in 9.3.2, "Start the default queue manager on the machine" on page 223, and is running by issuing the **dspmq** command:

```
Sample AMQSPUT0 start
MQCONN ended with reason code 2059
```

If you receive output similar to the following output, check that the queue name you specified is correct (including the case) and check that the object you defined has the correct name (including the case) by viewing it, as described in 9.3.4, "Display the attributes of the newly created queue" on page 227:

```
Sample AMQSPUT0 start
target queue is QUEUE1
MQOPEN ended with reason code 2085
unable to open queue for output
Sample AMQSPUT0 end
```

These reason code numbers can be viewed in a more readable form by using the **mqrc** command, followed by the reason code. Example output from calls to mqrc for 2058, 2059, and 2085 is as follows:

```
2058  0x0000080a  MQRC_Q_MGR_NAME_ERROR
2059  0x0000080b  MQRC_Q_MGR_NOT_AVAILABLE
2085  0x00000825  MQRC_UNKNOWN_OBJECT_NAME
```

If you see another reason code from MQCONN, MQOPEN, or MQPUT displayed in the output from the sample, use the **mqrc** command to display it in a readable form. For more information about the reason code, refer to the "API completion and reason codes" section of *WebSphere MQ Messages*, GC34-6601.

9.3.7 Browse messages put to the queue

This step shows how to view, or *browse*, the messages currently on a queue in WebSphere MQ without removing any of those messages. This includes both the data the message contains and the information about that message held in the message descriptor.

The messages put are *nonpersistent* messages. The *reply-to queue manager* is automatically filled in as host1/qm1 by the queue manager to which the messages were put.

The messages generated in 9.3.6, "Put test messages onto this queue" on page 230 were *datagram* messages, meaning they do not require a reply.

Because of this, no *reply-to queue* was specified by the WebSphere MQ Explorer or **amqsput** sample when putting the message.

For this step, you can use the WebSphere MQ Explorer or the **amqsbcg** WebSphere MQ sample program.

Note: These methods can be very useful in a real system during development, testing, or when monitoring information on queues. Browsing a queue in this way does not affect the messages it contains.

Using the WebSphere MQ Explorer

Perform the following steps:

1. Right-click the queue in the table of queues for the queue manager and select **Browse Messages**. A progress window opens, followed by the Message browser window.
2. Each row in the table of the Message browser window represents a message on the queue. The table columns displays the information about that message contained in the message descriptor and the *message data* that message contains. You can use schemes on this window to change the column order, for example, to bring the Persistence, Message type, Message data, Reply-to queue, and Reply-to queue manager columns near the beginning of the column order.
3. Some messages are best viewed as binary data, rather than as text. To do this, double-click a row in the table representing a message. A properties window opens. Select the **Data** section in this properties window.

Using the WebSphere MQ sample programs

Perform the following steps:

1. Execute the following command to browse the messages on the queue, which is hosted on the default queue manager, and redirect the output to a file called queue1.txt:

   ```
   amqsbcg queue1 > queue1.txt
   ```

2. Open the queue1.txt file in a text editor of your choice. For each message, the information about the message stored in the message descriptor is displayed, and the message data contained in the message is shown in a binary display with the text version to the right of the data.

   The information in this output is not as easy to read as the data in the WebSphere MQ Explorer. Extracts (not necessarily full lines) from such an output, which are of interest, are explained as follows:

   ```
   MsgType : 8
   ```

A message type of 8 is a datagram message. Request messages are 1, reply messages are 2, and report messages are 4.

This shows the message is nonpersistent; a value of 1 means that the message is persistent:

```
Persistence : 0
```

This shows the full 48 characters of the reply-to queue name, with blanks to the right of the data. This is called *blank-padded data* and is used frequently in WebSphere MQ structures. Here, the field in the message descriptor is entirely blank:

```
ReplyToQ       : '                                                '
```

This shows the 48 character blank padded reply-to queue manager name, as automatically filled in by the queue manager when the message was put:

```
ReplyToQMgr    : 'host1/qm1                                       '
```

This shows a binary display of the message data, with a text-based display to the right. This is provided because some message data, especially as contained in large messages, is more easily viewed in a binary format. However, for our simple test messages, the data is not as easy to read.

```
****   Message      ****
 length - 20 bytes
00000000:  5265 6462 6F6F 6B20 7465 7374 206D 6573 'Redbook test mes'
00000010:  7361 6765                               'sage            '
```

9.3.8 Defining and putting to an alias of a locally hosted queue

This step defines a queue alias object that has a different name that the previously created queue but references the same queue containing messages.

The attributes of a queue alias object affect the default behavior for messages put to the target queue. This step shows how specifying a default persistence of *persistent* on a queue alias can causes the messages put to a queue through that queue alias object to be persistent.

Note: In this step, both persistent and nonpersistent messages are placed on the same queue for demonstration purposes. However, WebSphere MQ is most efficient when a queue contains either persistent or nonpersistent messages. Therefore, we do not generally recommend placing a mixture of persistent and nonpersistent messages on a queue.

Using the WebSphere MQ Explorer

Perform the following steps:

1. Right-click the **Queues** folder for host1/qm1, and select **New** → **Alias Queue**. This opens the New Alias Queue wizard, with “Enter a name” displayed with a red cross.
2. Type the name `queue1.persistent` in the Name field.
3. Click **Next**. Do not click Finish. The wizard displays Change the properties of the new Alias Queue. This window is used to specify the attributes of the queue alias object.
4. Type `queue1` in the Base queue field.

   Note: This specifies the target queue (`TARGQ`) attribute of the queue alias object.

5. Enter the following value in the Description field:

   ```
   Redbook example alias to queue1: For persistent messages
   ```

6. Select **Persistent** for the Persistence field.
7. Click **Finish**, and click **OK** on the `AMQ4148` message that opens stating that the object was created successfully.
8. Right-click the new entry in the Queues table called **queue1.persistent**, select **Put Test Message**, and put some messages to the queue.
9. Notice that the current queue depth of queue1 has increased. Browse the messages on queue1, as described in 9.3.7, “Browse messages put to the queue” on page 232. Notice that the new messages put to the queue through the queue alias have a persistence of Persistent, while the messages previously put directly to the queue have a persistence of Not Persistent.

Using MQSC commands

Perform the following steps:

1. Execute the following MQSC command (using **`runmqsc`**):

   ```
   DEFINE QALIAS('queue1.persistent') TARGQ('queue1') DEFPSIST(YES) +
         DESCR('Redbook example alias to queue1: For persistent messages')
   ```

2. Verify the attributes of the alias queue object using the following MQSC command:

   ```
   DISPLAY QALIAS('queue1.persistent') TARGQ DEFPSIST DESCR
   ```

   Note: Remember that `queue1` is lowercase, so it is important to enclose the TARGQ in single quotation marks when defining the queue alias object.

3. After exiting **runmqsc** using END, execute the following command and put some messages to the alias queue using the WebSphere MQ sample program (a blank line exits the program):

   ```
   amqsput queue1.persistent
   ```

4. Browse the queue, as described in 9.3.7, “Browse messages put to the queue” on page 232. Notice that the new messages put to the queue are persistent, while the existing messages put directly to the queue are nonpersistent.

9.3.9 End and restart the queue manager

This step ends the queue manager, using either the WebSphere MQ Explorer or the **endmqm** WebSphere MQ control command. The queue manager is then restarted, and the queue containing messages is browsed. You will see that the nonpersistent messages are lost, while the persistent messages remain on the queue.

This step performs a quiesced shutdown of the queue manager, informing all applications connected to the queue manager that the queue manager is ending, but not denying new actions from those applications.

Note: An immediate shutdown can also be performed. This allows the current action of each application to complete successfully, but allows no further actions to be performed. Use this method if a queue manager does not end within a reasonable amount of time. The immediate shutdown action can be performed while an existing quiesced shutdown action is in progress.

Using the WebSphere MQ Explorer

Perform the following steps:

1. Right-click the icon for the queue manager under the Queue Managers folder, and select **Stop**. This opens the End Queue Manager window.
2. Leave the **Controlled** radio button selected.

   Note: The controlled end method in the WebSphere MQ Explorer is the same as the wait end method available with the **endmqm** command. This performs a quiesced shutdown of the queue manager and displays the status window until the end has completed.

3. Wait for the status window to complete.

   Notice that the icon for the queue manager has changed to the correct icon for a stopped local queue manager.

4. Restart the queue manager by right-clicking the queue manager and selecting **Start**. Wait for the progress window to complete.
5. Browse the contents of queue1. Notice that only the persistent messages put through the queue alias object remain on the queue.

Using WebSphere MQ control commands

Perform the following steps:

1. Use the following command to end queue manager, waiting for a quiesced shutdown to complete:

   ```
   endmqm -w host1/qm1
   ```

   > **Note:** The queue manger name must be specified on an **endmqm** command, even for the default queue manager. This is to avoid an accidental shutdown of the default queue manager.
   >
   > Another shutdown option for the **endmqm** command is:
   >
   > ```
   > endmqm Queue_Manager_Name
   > ```
   >
   > Without arguments, this initiates a quiesced shutdown and then returns immediately. The **dspmq** command can then be used to determine when the shutdown has completed.
   >
   > This performs an immediate shutdown of the queue manager:
   >
   > ```
   > endmqm -i Queue_Manager_Name
   > ```

2. Restart the queue manager using the **amqmdain qmgr start** (Windows) or **strmqm** (UNIX) command, as described in 9.3.2, “Start the default queue manager on the machine” on page 223.
3. Browse the messages on queue1, as described in 9.3.7, “Browse messages put to the queue” on page 232. Notice that only the persistent messages put through the queue alias object remain on the queue.

Note: You can configure an individual local queue to retain nonpersistent messages during a normal restart by specifying an NPM class (`NPMCLASS`) of `High`. This attribute is in the Storage section of the properties window for the queue. You can try using this now, but configuring queue1 for a high nonpersistent message class and performing this step again with nonpersistent messages on the queue.

However, this is *not* the same as using persistent messaging. WebSphere MQ does not log actions on nonpersistent messages and does not attempt to recover the messages after an abrupt failure of the queue manager, so nonpersistent messages on queues with a high non persistent message class *can still be lost*. Always mark business-critical data as persistent.

9.3.10 Get messages from a queue

This step shows how to *get* messages currently on a queue in WebSphere MQ. This permanently retrieves a message from a queue. Only one application can successfully get any given message. Perform this step using the WebSphere MQ sample programs.

Perform the following steps:

1. If no messages remain on queue1, put some test messages onto the queue, either directly or through the queue alias.
2. Execute the following command to get all messages on the queue, which is hosted on the default queue manager:

   ```
   amqsget queue1
   ```

   This command shows a line for each message that is on the queue and then waits for 10 seconds for messages to arrive on that queue before ending. If you put more messages within that 10 seconds, they are displayed instantly, and the command waits another 10 seconds.

   Example output is as follows:

   ```
   Sample AMQSGET0 start
   message <Redbook test message 1>
   message <Redbook test message 2>
   message <Redbook test message 3>
   no more messages
   Sample AMQSGET0 end
   ```

3. Notice that if you browse the queue after the command has ended, no messages remain.

9.3.11 Delete a queue object

This step demonstrates how to delete a queue object using the alias queue object previously defined. For this step, you can use the WebSphere MQ Explorer or the DELETE MQSC command.

Using the WebSphere MQ Explorer

Perform the following steps:

1. Right-click the alias queue object **queue1.persistent** in the queues table for the queue manager, and select **Delete**. This opens a confirmation window.
2. Check the name of the object, and click **Yes**.
3. After the status window closes, a confirmation opens. Click **OK**.
4. Notice that the object is no longer listed in the queues table.

Using MQSC commands:

Perform the following steps:

1. Issue the following MQSC command (using **runmqsc**):

   ```
   DELETE QALIAS('queue1.persistent')
   ```

   You receive the following output:

   ```
   AMQ8007: WebSphere MQ queue deleted.
   ```

2. Notice that the queue object is no longer displayed in the output of either of the following MQSC commands:

   ```
   DISPLAY QUEUE(*)
   DISPLAY QALIAS(*)
   ```

9.3.12 Define a queue manager alias using a remote queue object

A queue manager alias is just one example of a use of a remote queue object. This step shows how one can be created. Use the WebSphere MQ Explorer or the DEFINE QREMOTE MQSC command.

Note: A queue manager alias is a remote queue object with a blank remote queue name attribute. We demonstrate the use of a remote queue object as a local definition of a remote queue in 10.3.12, “Create a local definition of a remote queue” on page 288.

Using the WebSphere MQ Explorer

Perform the following steps:

1. Right-click the **Queues** folder for host1/qm1, and select **New** → **Remote Queue**. This opens the New Remote Queue wizard, with “Enter a name” displayed with a red cross.
2. Type the name `host1/qm1.alias` in the Name field.
3. Click **Next**. Do not click Finish. The wizard displays Change the properties of the new Alias Queue. This window is used to specify the attributes of the queue alias object.
4. Type `host1/qm1` in the Remote queue manager field.
5. Enter the following value in the Description field:

   ```
   Redbook example queue manager alias to host1/qm1
   ```
6. Click **Finish**, and click **OK** on the `AMQ4148` message that opens stating that the object was created successfully.

Using MQSC commands

Perform the following steps:

1. Execute the following MQSC command (using **`runmqsc`**):

   ```
   DEFINE QREMOTE('host1/qm1.alias') RNAME('') RQMNAME('host1/qm1') +
          DESCR('Redbook example queue manager alias to host1/qm1')
   ```
2. Verify the attributes of the remote queue object defining the queue manager alias using the following MQSC command:

   ```
   DISPLAY QREMOTE('host1/qm1.alias') RNAME RQMNAME DESCR
   ```

   Note: Remember that `host1/qm1` is lowercase, so it is important to enclose `RQMNAME` in single quotation marks when defining the queue alias object.

9.3.13 Specify a queue manager name when opening a queue

This step demonstrates how a specific queue manager name can be specified when opening a queue to put a message. Use the WebSphere MQ sample programs to perform this step. This example demonstrates the flexibility the different types of queue objects provided for controlling queue name resolution.

The WebSphere MQ sample programs are flexible in the options they provide, and this example uses some extra options on the **`amqsput`** sample.

Note: The particular command shown in this example can be useful to test a route to a remote queue manager by sending messages to a queue of a particular name on that queue manager through a local queue manager.

Using the WebSphere MQ sample programs

Perform the following steps:

1. Execute the WebSphere MQ sample command as follows, and put some messages:

   ```
   amqsput queue1 host1/qm1 8208 0 host1/qm1.alias
   ```

   The following parameters are passed to this sample:

   – `queue1` is the name of the queue.

   – `host1/qm1` is the name of the queue manager being connected to.

   – `8208` is a decimal value requesting options to be passed to the MQOPEN call. *This is not important to the example*. It is requesting that the queue is opened for output to put messages and to fail any subsequent puts if the queue manager is quiescing.

   – `0` requests that no options are passed to the MQCLOSE call. *This is not important to the example*.

   – `host1/qm1.alias` is the name of the object queue manager specified in the MQOPEN call. This is the extra parameter used in this example. You can see that this name matches the queue manager alias that was previously defined, not the queue manager name to which the application is connected. Queue name resolution can resolve this to the name of the local queue manager, which is host1/qm1, through the queue manager alias.

   Note: If you delete the remote queue object defining the queue manager alias, or specify an incorrect queue manager name (case-sensitive) in either the name or attribute of the queue remote definition or on the **`amqsput`** command, you see a `2087 MQRC_UNKNOWN_REMOTE_Q_MGR` return code from the MQOPEN call.

2. By browsing queue1, you can see that the messages are delivered to queue1 on queue manager host1/qm1, even though addressed to host1/qm1.alias.

9.3.14 Delete the queue manager

This step shows how to delete a queue manager. All messages, including persistent messages, held on the queues of that queue manager are lost and cannot be recovered by WebSphere MQ. For this step, you can use the WebSphere MQ Explorer or the `dltmqm` WebSphere MQ control command.

Note: The `dltmqm` WebSphere MQ control command does not request confirmation before deleting a queue manager. Take care whenever issuing a `dltmqm` command. The action of deleting a queue manager cannot be undone by WebSphere MQ.

Using the WebSphere MQ Explorer

Perform the following steps:

1. If queue manager host1/qm1 is running, right-click the queue manager in the navigator view and select **Stop** to end the queue manager.
2. After choosing the end method and clicking **OK**, a status window opens. When the status window completes, right-click queue manager **host1/qm1** in the navigator tree and select **Delete**.
3. Check the name of the queue manager displayed in the confirmation window. If it is correct, click **Yes** to complete the deletion; otherwise, click **No** to return to the WebSphere MQ Explorer. A status window opens if you choose to complete the deletion.
4. When the status window completes, the queue manager is deleted and thus no longer displayed in the navigator view.

Using WebSphere MQ control commands

Perform the following steps:

1. End the queue manager, as described in 9.3.9, “End and restart the queue manager” on page 236, for example using:

   ```
   endmqm -w host1/qm1
   ```

2. When the queue manager ends, the following command can be used to delete the queue manager:

   ```
   dltmqm host1/qm1
   ```

   Note: As with the `endmqm` command, the `dltmqm` command requires you to provide the name of the queue manager, even if deleting the default queue manager on the machine.

9.4 Host a request/reply service on a queue

This section demonstrates how a service can be hosted on a queue manager and provides a request/reply interface through a queue. This includes configuring triggering to automatically start the service when messages arrive on that queue. We demonstrate this using the WebSphere MQ samples.

9.4.1 Create and start a queue manager to host the service

This step creates a new queue manager, called host1/echo.hub. This reflects that the queue manager is a hub for a particular service and that this service echoes the text in a request back in the reply.

Using the WebSphere MQ Explorer

Perform the following steps:

1. Right-click the **Queue Managers** folder, and select **New** → **Queue Manager**.
2. Type `host1/echo.hub` into the Queue manager name field.
3. Ensure Make this queue manager the default queue manager is not selected.
4. Click **Next**.
5. Click **Next**, leaving the default logging values.
6. Ensure that **Start queue manager** check box is selected to cause the queue manager to start automatically.
7. Click **Next**.
8. Clear **Create listener configured for TCP/IP**.
9. Click **Finish**.

Using WebSphere MQ control commands

Perform the following steps:

1. Create the queue manager without specifying that it should be the default queue manager using the following command:

   ```
   crtmqm host1/echo.hub
   ```

2. Start the queue manager using the following command:

   – Windows:

     ```
     amqmdain qmgr start host1/echo.hub
     ```

   – UNIX:

     ```
     strmqm host1/echo.hub
     ```

9.4.2 Create a queue to host the service

This step creates a local queue object from which the service is later configured to be triggered and process requests. For this step, you can use the WebSphere MQ Explorer or MQSC commands. This reply-to queue has the default queue attributes. By default, requests put to this queue are nonpersistent.

Using the WebSphere MQ Explorer

Perform the following steps:

1. Right-click the **Queues** folder for host1/echo.hub, and select **New** → **Local Queue**.
2. Type `echo` into the Name field.
3. Click **Next**.
4. Enter the following value in the Description field:

   ```
   Queue hosting the echo service
   ```
5. Click **Finish**.

Using MQSC commands

Issue the following MQSC command against host1/echo.hub:

```
DEFINE QLOCAL('echo') +
       DESCR('Queue hosting the echo service')
```

9.4.3 Manually define a reply-to queue

This step manually creates a local queue object, which the requesting application uses to receive replies.For this step, you can use the WebSphere MQ Explorer or MQSC commands. This reply-to queue has the default queue attributes.

Using the WebSphere MQ Explorer

Perform the following steps:

1. Right-click the **Queues** folder for host1/echo.hub, and select **New** → **Local Queue**.
2. Type `echo.replies.manual` into the Name field
3. Click **Next**.
4. Enter the following value in the Description field:

   ```
   Manually defined reply-to queue for echo service
   ```
5. Click **Finish**.

Using MQSC commands

Note: Because host1/echo.hub is not the default queue manager, an interactive MQSC session is started using the following command:

```
runmqsc host1/echo.hub
```

Remember that the END command is used to end an interactive MQSC session.

Issue the following MQSC command against host1/echo.hub:

```
DEFINE QLOCAL('echo.replies.manual') +
      DESCR('Manually defined reply-to queue for echo service')
```

9.4.4 Put an example request message and inspect it

This step puts an example request message and inspects the message to see how a request message differs from a datagram. For this step, you can use the WebSphere MQ sample programs and browse the queue as discussed in 9.3.7, “Browse messages put to the queue” on page 232.

Perform the following steps:

1. Issue the **amqsreq** WebSphere MQ sample command as follows:

   ```
   amqsreq echo host1/echo.hub echo.replies.manual
   ```

   Note: This command uses the following parameters:

   - `echo` is the name of the queue hosting the service that provides the request/reply interface. Currently, no service is active against this queue.
   - `host1/echo.hub` is the name of the queue manager to which the requesting application connects.
   - `echo.replies.manual` is the name of the reply-to queue that the requesting application specifies in the message descriptors of requests and waits for replies to arrive on.

   The initial output from this command is:

   ```
   Sample AMQSREQ0 start
   server queue is echo
   replies to echo.replies.manual
   ```

Note: Remember to check the case of the name of the queues in MQSC and the WebSphere MQ Explorer, as well as all of the parameters specified on the command. Use the **`mqrc`** command for a textual description of any unexpected return codes.

2. Type some messages into the command. Press Enter after each message. Enter a blank line to stop sending messages. The sample then waits for 10 seconds for any responses. No responses are received, because the service is not configured at this time.
3. Browse the messages placed on the echo queue. Notice the following information:
   - The message type (`MsgType`) is request (or the number 1).
   - The reply-to queue is `echo.replies.manual`.
   - The reply-to queue manager is `host1/echo.hub`.
   - If using the WebSphere MQ Explorer, select the **Identifiers** section of the message properties window. Notice that the message identifier (`MsgId`) contains a value, which is uniquely generated by WebSphere MQ.
   - Notice that the correlation identifier (`CorrelId`) is blank.

9.4.5 Clear the requests from the queue hosting the service

This step clears the echo queue that hosts the service. This step ensures that a message already on the queue is not preventing the WebSphere MQ triggering rules from generating a trigger message when a subsequent message arrives. You might want to refer back to this step if you experience problems in later steps.

Note: This step is not strictly required. In a real environment, clearing a queue containing requests might cause the loss of important data.

For this step, you can use the WebSphere MQ Explorer or MQSC commands.

Using the WebSphere MQ Explorer

Perform the following steps:

1. Select the **Queues** folder for host1/echo.hub.
2. Right-click the row in the table for queue echo, and select **Clear Messages**.
3. On the Clear queue window, leave **Queue will be cleared with the CLEAR command** selected, and click **Clear**.

4. Notice that the current queue depth attribute returns to zero.

Using MQSC commands

Issue the following MQSC command against host1/echo.hub:

```
CLEAR QLOCAL('echo')
```

9.4.6 Create a process definition for the service

These steps use the **amqsech** WebSphere MQ sample program to provide a request/reply service hosted on the echo queue. This sample program is designed to be started using the WebSphere MQ triggering mechanism.

This step creates a process definition, which is used to start the **amqsech** WebSphere MQ sample application performing the service. For this step, you can use the WebSphere MQ Explorer or MQSC commands.

Using the WebSphere MQ Explorer

Perform the following steps:

1. Right-click the **Process Definitions** folder for host1/echo.hub and select **New → Process Definition**.

   Note: This folder is under the Advanced subfolder for the queue manager in the navigator tree.

2. Type `amqsech` into the Name field.
3. Click **Next**.
4. Enter the following value in the Description field:

   ```
   The amqsech WebSphere MQ sample program
   ```

5. Choose an appropriate value for the Application type, as follows:
   – For Windows, use `Windows NT`.
   – For UNIX, use `Unix`.
6. Type the path of the sample in the Application ID, as follows:
   – For Windows:

     ```
     C:\Program Files\IBM\WebSphere MQ\Tools\c\samples\bin\amqsech.exe
     ```

   – For UNIX (except AIX 5L):

     ```
     /opt/mqm/samp/bin/amqsech
     ```

– For AIX 5L:

```
/usr/mqm/samp/bin/amqsech
```

7. Click **Finish**.

Using MQSC commands

Perform the following steps:

1. Issue the following MQSC command against host1/echo.hub:

   – Windows:

```
DEFINE PROCESS('amqsech') +
       DESCR('The amqsech WebSphere MQ sample program') +
       APPLTYPE(WINDOWSNT) +
APPLICID('C:\Program Files\IBM\WebSphere MQ\Tools\c\samples\bin\amqsech.exe')
```

   – UNIX (except AIX 5L):

```
DEFINE PROCESS('amqsech') +
       DESCR('The amqsech WebSphere MQ sample program') +
       APPLTYPE(UNIX) APPLICID('/opt/mqm/samp/bin/amqsech')
```

   – AIX 5L:

```
DEFINE PROCESS('amqsech') +
       DESCR('The amqsech WebSphere MQ sample program') +
       APPLTYPE(UNIX) APPLICID('/usr/mqm/samp/bin/amqsech')
```

2. Display the attributes of the process definition object created using the following MQSC command:

```
DISPLAY PROCESS('amqsech')
```

Note: Check the case of the attributes and the name of the process definition object. On UNIX platforms the path is case-sensitive.

9.4.7 Create a queue to use as an initiation queue

This step creates a local queue with default attributes that is subsequently configured as the initiation queue for trigger messages for the echo queue. For this step, you can use the WebSphere MQ Explorer or MQSC commands.

Using the WebSphere MQ Explorer

Perform the following steps:

1. Right-click the **Queues** folder for host1/echo.hub, and select **New** → **Local Queue**.
2. Type `echo.initq` into the Name field.
3. Click **Next**.

4. Enter the following value in the Description field:

   ```
   Initiation queue for triggering the echo service
   ```

5. Click **Finish**.

Using MQSC commands

Issue the following MQSC command against host1/echo.hub:

```
DEFINE QLOCAL('echo.initq') +
      DESCR('Initiation queue for triggering the echo service')
```

9.4.8 Enable triggering on the queue hosting the service

This step alters the echo queue to enable triggering. The triggering mechanism being used is *first*. This generates a trigger message on the initiation queue when the first message arrives on the queue.

Using the WebSphere MQ Explorer

Perform the following steps:

1. Select the **Queues** folder for host1/echo.hub.
2. Right-click the row in the table for queue echo and select **Properties**.
3. Select the **Triggering** section.
4. Change the Trigger control field to **On**.
5. Change the Trigger type to **First**.
6. Type `echo.initq` into the Initiation queue field.
7. Type `amqsech` into the Process name field.
8. Click **OK**.

Using MQSC commands

Perform the following steps:

1. Issue the following MQSC command against host1/echo.hub:

   ```
   ALTER QLOCAL('echo') +
         TRIGGER TRIGTYPE(FIRST) INITQ('echo.initq') +
         PROCESS('amqsech')
   ```

2. Check the attributes of the queue object using the following MQSC command:

   ```
   DISPLAY QLOCAL('echo') TRIGGER TRIGTYPE INITQ PROCESS
   ```

   Note: Check that the case of the attributes is correct.

9.4.9 Start the WebSphere MQ trigger monitor

This step starts the WebSphere MQ trigger monitor, listening for trigger messages arriving on the echo.initq initiation queue previously defined.

Perform the following steps:

1. Start a new command window or terminal session. When started, the WebSphere MQ trigger monitor should remain running indefinitely for the echo service, because it is used later in this chapter. Refer back to this section if the trigger monitor is ended at any point. If the trigger monitor needs to be ended, press Ctrl+C after selecting the command window or terminal session in which it is running.
2. Issue the following command to start the WebSphere MQ trigger monitor for queue manager host1/echo.hub and initiation queue echo.initq:

   ```
   runmqtrm -m host1/echo.hub -q echo.initq
   ```

   The initial output from this command is as follows:

   ```
   5724-H72 (C) Copyright IBM Corp. 1994, 2004.  ALL RIGHTS RESERVED.
   WebSphere MQ trigger monitor started.

   __________________________________________________
   Waiting for a trigger message
   ```

9.4.10 Issue a request against the service

This step sends a request message to the echo queue, which causes a trigger message to be placed on the initiation queue, due to the attributes of the echo queue.

This trigger message is processed by the WebSphere MQ trigger monitor, which starts the **amqsech** WebSphere MQ sample program using the process definition object specified on the parameters of the echo queue.

The WebSphere MQ trigger monitor passes all information to the **amqsech** program from the trigger message. This allows the **amqsech** program to determine the queue from which to process messages.

The **amqsech** program sends replies that contain the original messages echoed back to the reply-to queue specified in the message descriptor of the request message.

Perform the following steps:

1. Issue the **amqsreq** WebSphere MQ sample command as follows:

   ```
   amqsreq echo host1/echo.hub echo.replies.manual
   ```

2. Type a single message and press Enter. Do not place a blank line after the message yet.
3. Look at the output from the trigger monitor. Notice that some extra output is displayed. This shows that a trigger message was generated on the initiation queue, processed by the trigger monitor, and that the **amqsech** sample program was started. Extra output is displayed after about 10 seconds when the **amqsech** sample stops waiting for messages and ends. Example output is as follows:

```
C:\PROGRA~1\IBM\WEBSPH~1\Tools\c\samples\bin\amqsech.exe "TMC    2echo
              amqsech
                              C:\Program Files\IBM\WebSphere
MQ\Tools\c\samples\bin\amqsech.exe




                                  host1/echo.hub
"
Sample AMQSECHA start
Example request message
MQGET ended with reason code 2033
Sample AMQSECHA end
End of application trigger.
```

4. This indicates that the **amqsech** sample, started by the WebSphere MQ trigger monitor, processed the message and sent a reply to the reply-to queue specified. This queue is the echo.replies.manual queue specified on the command line of the **amqsreq** sample that created the request message.

   Browse the message that has been produced on the echo.replies.manual queue. If you are using the **amqsbcg** sample to do this, you might need to open a new command window to run the browse command without ending the **amqsreq** sample.

   Notice the following information in the reply:

   – The message type (`MsgType`) is reply (or the number 2).

   – Both the message identifier (`MsgId`) and correlation identifier (`CorrelId`) have values. This is because the **amqsech** sample copied the message identifier from the request message descriptor into the correlation identifier of the reply message descriptor. This allows the reply to be correlated with the request by the application that performed the request. The message identifier of the reply is one uniquely generated by WebSphere MQ.

5. Return to the command window from which the request was issued using **amqsreq**. Press Enter on a blank line. After about 10 seconds, the **amqsreq** sample program completes. Example output is as follows:

```
Sample AMQSREQ0 start
server queue is echo
replies to echo.replies.manual
Example test message

response <Example test message>
no more replies
Sample AMQSREQ0 end
```

Note: If you experience problems during this step, check the attributes of all previously defined objects. Ensure that the case of object names and attributes is specified correctly. Also ensure that the case of parameters passed to commands, including the WebSphere MQ trigger monitor, is correct.

9.4.11 Define a model queue object for a dynamic reply-to queue

This step defines a model queue object from which temporary dynamic reply-to queues are created by the **amqsreq** sample when it requests the echo service. For this step, you can use the WebSphere MQ Explorer or MQSC commands.

Using the WebSphere MQ Explorer

Perform the following steps:

1. Right-click the **Queues** folder for host1/echo.hub, and select **New** → **Model Queue**.
2. Type `echo.replies.tempdyn` into the Name field.
3. Click **Next**.
4. Enter the following value in the Description field:

   ```
   Temporary dynamic reply-to queues for echo service
   ```
5. Select the **Extended** section.
6. Notice that Temporary Dynamic is specified in the Definition Type field.
7. Click **Finish**.

Using MQSC commands

Perform the following steps:

1. Issue the following MQSC command against host1/echo.hub:

   ```
   DEFINE QMODEL('echo.replies.tempdyn') +
          DESCR('Temporary dynamic reply-to queues for echo service')
   ```

2. Check the properties of the object defined using the following MQSC command:

   ```
   DISPLAY QMODEL('echo.replies.tempdyn') DESCR DEFTYPE
   ```

   Notice that the definition type (`DEFTYPE`) is temporary dynamic (`TEMPDYN`).

9.4.12 Issue requests using a temporary dynamic reply-to queue

This step requests the service in the same way as 9.4.10, "Issue a request against the service" on page 250. However, the reply-to queue specified on the **`amqsreq`** command line is the echo.replies.tempdyn model queue defined.

When the **`amqsreq`** sample opens the model queue, the queue manager automatically defines a temporary dynamic queue based on the attributes of the echo.replies.tempdyn model queue object. The queue manager automatically removes this object when the **`amqsreq`** sample completes. This includes when the queue manager detects the **`amqsreq`** sample completes uncleanly, for example, by pressing Ctrl+C during its operation. This might occur a number of seconds after the **`amqsreq`** sample is terminated.

Perform the following steps:

1. Issue the **`amqsreq`** WebSphere MQ sample command as follows:

   ```
   amqsreq echo host1/echo.hub echo.replies.tempdyn
   ```

   Example initial output is as follows:

   ```
   Sample AMQSREQ0 start
   server queue is echo
   replies to AMQ.42DD88DE02C70120
   ```

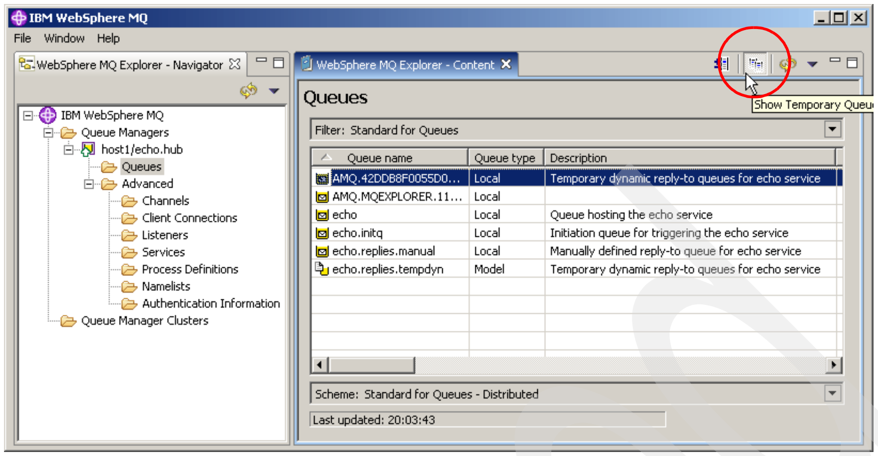
   `AMQ.42DD88DE02C70120` is the name of the dynamic queue that has been created.

2. View the properties of this temporary dynamic queue as follows:

   – Using the WebSphere MQ Explorer:

     i. Select the **Queues** folder for host1/echo.hub.

     ii. Enable the WebSphere MQ Explorer to display temporary queues, as shown in Figure 9-1 on page 254.



Queues

Filter: Standard for Queues

Queue name	Queue type	Description
AMQ.42DDB8F0055D0...	Local	Temporary dynamic reply-to queues for echo service
AMQ.MQEXPLORER.11...	Local	
echo	Local	Queue hosting the echo service
echo.initq	Local	Initiation queue for triggering the echo service
echo.replies.manual	Local	Manually defined reply-to queue for echo service
echo.replies.tempdyn	Model	Temporary dynamic reply-to queues for echo service

Scheme: Standard for Queues - Distributed

Last updated: 20:03:43

Figure 9-1 Showing temporary queues in the WebSphere MQ Explorer

iii. Notice that the description of the dynamic queue is the same as the description of the model queue object from which it was created.

Note: You might see temporary dynamic queues with names starting `AMQ.MQEXPLORER`. These are created by the WebSphere MQ Explorer based on the `SYSTEM.MQEXPLORER.REPLY.MODEL` model queue. The WebSphere MQ Explorer uses the request reply interface provided by the command servers of queue managers.

– Using MQSC commands:

i. Issue the following MQSC command against host1/echo.hub:

```
DISPLAY QL('AMQ.*') DEFTYPE DESCR
```

ii. Notice that the description (`DESCR`) attribute of the dynamic queue is the same as the description of the model queue object from which it was created.

3. Notice that the operation of the **`amqsreq`** application is the same as with a manually defined queue.
4. After the **`amqsreq`** application has completed, notice that the temporary dynamic queue is removed automatically.

Note: The **`amqsreq`** sample application does not support use of permanent dynamic queues, because it does not specify the required options to delete the queue when it closes the queue. However, you can to experiment with model queues with a definition type of permanent dynamic (`PERMDYN`) and the **`amqsreq`** sample to see how this allows persistent messages to be used. The temporary queues created need to be manually removed using the MQSC commands or the WebSphere MQ Explorer.

9.5 WebSphere MQ publish/subscribe with JMS

This section demonstrates the WebSphere MQ publish/subscribe broker, using the WebSphere MQ Java Message Service (JMS) publish/subscribe sample.

Note: This requires that a Java Development Kit (JDK) is installed on the machine from which these steps are run. A Version 1.4.2 JDK is supplied in the Prereqs directory of the WebSphere MQ V6.0 installation media.

Ensure that the directory containing the **`java`** and **`javac`** executables from this JDK have been added to the operating system path.

Some very basic editing of the WebSphere MQ JMS publish/subscribe sample is required, and the sample must be compiled. However, this can be performed without previous knowledge of the Java programming language.

9.5.1 Configure the JMS environment

Create a directory on the machine from which all commands are run and in which all files and other directories required for these steps are created. All command windows or terminal sessions should work within this directory.

The compiled components of a Java application are found by the Java Runtime Environment (JVM) within which a Java application runs by configuring a *class path*. This is configured using an environment variable.

On Windows, the WebSphere MQ installation adds the components required for running WebSphere MQ JMS applications to the Java class path. However, on UNIX platforms, this must be performed by running a script in the current shell.

On all platforms, these instructions assume the working directory is added to the Java class path.

Due to these considerations, the perform the following steps in each command window or terminal session that is used for these steps:

- Windows:

  a. Change directory to the chosen working directory. For example:

     ```
     C:
     cd \wmq_redbook\jmspubsub
     ```

  b. Execute the following commands to add the current directory to the Java class path:

     ```
     set CLASSPATH=%CLASSPATH%;.
     ```

- UNIX:

  a. Change directory to the chosen working directory. For example:

     ```
     cd ~/wmq_redbook/jmspubsub
     ```

  b. Execute the following command, appropriate to the UNIX platform (the space between the `.` and the `/` is important):

     - UNIX, except AIX 5L:

       ```
       . /opt/mqm/java/bin/setjmsenv
       ```

     - AIX 5L:

       ```
       . /usr/mqm/java/bin/setjmsenv
       ```

  c. Execute the following commands to add the current directory to the Java class path, as well as an additional Jar package not added by **`setjmsenv`**:

     - UNIX, except AIX 5L:

       ```
       CLASSPATH=$CLASSPATH:/opt/mqm/java/lib/jms.jar:.
       export CLASSPATH
       ```

     - AIX 5L:

       ```
       CLASSPATH=$CLASSPATH:/usr/mqm/java/lib/jms.jar:.
       export CLASSPATH
       ```

Note: If you experience errors in later steps containing the following text, use the solution here:

```
Exception in thread "main" java.lang.NoClassDefFoundError
```

Manually configure the CLASSPATH environment variable as follows:

- UNIX, except AIX 5L:

```
CLASSPATH=$CLASSPATH:/opt/mqm/java/lib/com.ibm.mq.jar
CLASSPATH=$CLASSPATH:/opt/mqm/java/lib/com.ibm.mqjms.jar
CLASSPATH=$CLASSPATH:/opt/mqm/java/lib/connector.jar
CLASSPATH=$CLASSPATH:/opt/mqm/java/lib/jms.jar
CLASSPATH=$CLASSPATH:/opt/mqm/java/lib/jndi.jar
CLASSPATH=$CLASSPATH:/opt/mqm/java/lib/jta.jar
CLASSPATH=$CLASSPATH:/opt/mqm/java/lib/providerutil.jar
CLASSPATH=$CLASSPATH:/opt/mqm/java/lib/fscontext.jar
CLASSPATH=$CLASSPATH:/opt/mqm/java/lib/ldap.jar
CLASSPATH=$CLASSPATH:.
export CLASSPATH
```

- AIX 5L:

```
CLASSPATH=$CLASSPATH:/usr/mqm/java/lib/com.ibm.mq.jar
CLASSPATH=$CLASSPATH:/usr/mqm/java/lib/com.ibm.mqjms.jar
CLASSPATH=$CLASSPATH:/usr/mqm/java/lib/connector.jar
CLASSPATH=$CLASSPATH:/usr/mqm/java/lib/jms.jar
CLASSPATH=$CLASSPATH:/usr/mqm/java/lib/jndi.jar
CLASSPATH=$CLASSPATH:/usr/mqm/java/lib/jta.jar
CLASSPATH=$CLASSPATH:/usr/mqm/java/lib/providerutil.jar
CLASSPATH=$CLASSPATH:/usr/mqm/java/lib/fscontext.jar
CLASSPATH=$CLASSPATH:/usr/mqm/java/lib/ldap.jar
CLASSPATH=$CLASSPATH:.
export CLASSPATH
```

9.5.2 Create and start a queue manager

Create and start a queue manager called host1/jmspubsub, which provides the publish/subscribe broker.

For this step, you can use WebSphere MQ control commands or the WebSphere MQ Explorer, as described in 9.4.1, “Create and start a queue manager to host the service” on page 243.

9.5.3 Start the broker on the queue manager

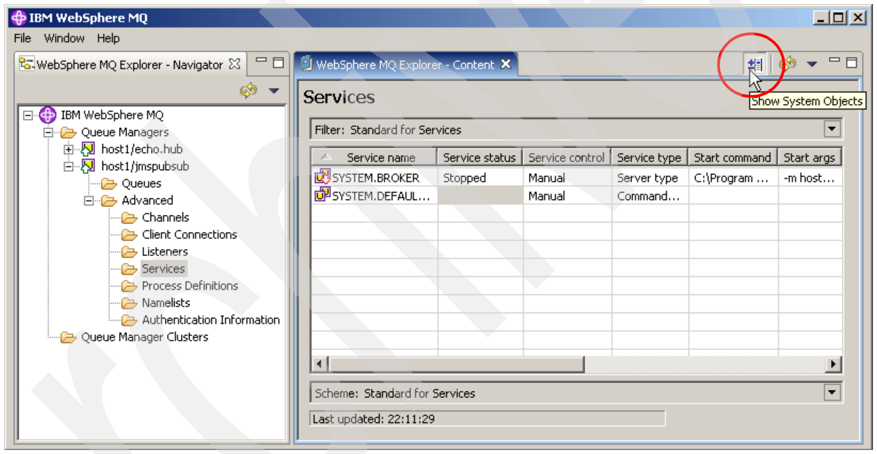
WebSphere MQ V6.0 provides a service object created with the queue manager that you can use to start the WebSphere MQ publish/subscribe broker on that queue manager.

The WebSphere MQ publish/subscribe broker is not started by default, but can be configured to be started with a queue manager. For this step, you can use the WebSphere MQ Explorer or MQSC commands.

Using the WebSphere MQ Explorer

Perform the following steps:

1. Select the **Services** folder for queue manager host1/jmspubsub.
2. Click the icon to display system objects (unless these are already displayed), as shown in Figure 9-2.



Services

Filter: Standard for Services

Service name	Service status	Service control	Service type	Start command	Start args
SYSTEM.BROKER	Stopped	Manual	Server type	C:\Program ...	-m host...
SYSTEM.DEFAUL...		Manual	Command...		

Scheme: Standard for Services

Last updated: 22:11:29

Figure 9-2 Show system objects in the WebSphere MQ Explorer

3. Right-click the service called **SYSTEM.BROKER** and select **Properties**.
4. Change the Service control field to **Queue manager**. This configures the publish/subscribe broker to start with the queue manager.
5. Click **OK**.
6. Right-click the service called **SYSTEM.BROKER** and select **Start** to start the WebSphere MQ publish/subscribe broker.

Using MQSC commands

Perform the following steps:

1. Issue the following MQSC command against host1/jmspubsub to configure the WebSphere MQ publish/subscribe broker to start with the queue manager:

   ```
   ALTER SERVICE('SYSTEM.BROKER') CONTROL(QMGR)
   ```

2. Issue the following MQSC command against host1/jmspubsub to start the publish/subscribe broker:

   ```
   START SERVICE('SYSTEM.BROKER')
   ```

9.5.4 Configure the queue manager for JMS publish/subscribe

WebSphere MQ requires that certain configuration is performed on the queue manager in order for JMS applications to access the capabilities of the publish/subscribe broker on that queue manager.

An MQSC script is provided to perform this configuration. Run this script against the host1/jmspubsub queue manager as follows:

- Windows (all one line):

  ```
  runmqsc host1/jmspubsub < "C:\Program Files\IBM\WebSphere MQ
                                                    \Java\bin\MQJMS_PSQ.mqsc"
  ```

- UNIX (except AIX 5L):

  ```
  runmqsc host1/jmspubsub < /opt/mqm/java/bin/MQJMS_PSQ.mqsc
  ```

- AIX 5L:

  ```
  runmqsc host1/jmspubsub < /usr/mqm/java/bin/MQJMS_PSQ.mqsc
  ```

9.5.5 Set up a simple JMS provider

JMS is a standardized interface that is not specific to WebSphere MQ. Therefore, WebSphere MQ-specific information is required to map JMS actions into WebSphere MQ actions.

Information about this mapping can be gained flexibly by an application by querying a *directory* for information about how JMS is provided to that application. This directory might not be a directory on the file system; it can be located in many places. Often, in a production environment, it is in an Lightweight Directory Access Protocol (LDAP) server, accessed over the network. This allows changes to the configuration to happen flexibly.

The application using JMS looks up details of how JMS is provided to that application by accessing the directory through an interface called the Java Naming and Directory Interface™ (JNDI).

In this simple example, a directory on the file system is set up to provide this information. We then configure it using the WebSphere MQ JMS Administration tool to set up the contents of this directory.

The following setup is required before the WebSphere MQ JMS Administration tool can be used:

1. Within the working directory for this example, create the following subdirectory:

   ```
   jms.provider
   ```

   Do not change the working directory to this directory.

2. Within the working directory, create a file called JMSAdmin.config with the following contents:

   ```
   INITIAL_CONTEXT_FACTORY=com.sun.jndi.fscontext.RefFSContextFactory
   PROVIDER_URL=file:jms.provider
   SECURITY_AUTHENTICATION=none
   ```

   This configures the WebSphere MQ JMS Administration tool to set up the contents of the file system directory created.

9.5.6 Use WebSphere MQ JMS Administration tool to configure JMS

The WebSphere MQ JMS Administration tool creates objects in a directory accessed through JNDI. After these objects have been configured, the same directory is accessed by applications accessing WebSphere MQ with JMS through JNDI to configure how JMS is provided to that application by WebSphere MQ.

The syntax of the commands executed within the WebSphere MQ JMS Administration tool are similar to MQSC. However, lowercase object names and attributes do not need to be enclosed in single quotation marks.

For details of the WebSphere MQ JMS Administration tool and the objects that can be created, refer to *WebSphere MQ V6.0 Using Java*, SC34-6591.

This step creates a Topic Connection Factory object, which defines how a JMS application can connect to provider of publish/subscribe capabilities, and a Topic object, which defines a topic on which that application can publish messages or subscribe publications.

Perform the following steps:

1. Run the WebSphere MQ JMS Administration tool as follows:
   - Windows:

     ```
     "C:\Program Files\IBM\WebSphere MQ\Java\Bin\JMSAdmin.bat"
     ```
   - UNIX (except AIX 5L):

     ```
     /opt/mqm/java/bin/JMSAdmin
     ```
   - AIX 5L:

     ```
     /usr/mqm/java/bin/JMSAdmin
     ```

   The initial output is as follows:

   ```
   5724-H72, 5655-L82, 5724-L26 (c) Copyright IBM Corp. 2002,2005. All Rights
   Reserved.
   Starting Websphere MQ classes for Java(tm) Message Service Administration

   InitCtx>
   ```
2. Create the Topic Connection Factory (TCF) object to connect to the publish/subscribe broker on queue manager host1/jmspubsub using the following command:

   ```
   DEFINE TCF(PubSub.TCF) QMANAGER(host1/jmspubsub) TRANSPORT(BIND)
   ```
3. Create a Topic object, which uses a WebSphere MQ topic string of MQJMS/Samples/PubSub, using the following command:

   ```
   DEFINE T(PubSub.T) TOPIC(MQJMS/Samples/PubSub)
   ```
4. End the WebSphere MQ JMS Administration tool using the following command:

   ```
   END
   ```

9.5.7 Make a copy of the WebSphere MQ sample JMS application

The WebSphere MQ JMS publish/subscribe sample application is in the following location:

- Windows:

  C:\Program Files\IBM\WebSphere MQ\Tools\Java\jms\JMSPubSub.java
- UNIX (except AIX 5L):

  /opt/mqm/samp/java/jms/JMSPubSub.java
- AIX 5L:

  /usr/mqm/samp/java/jms/JMSPubSub.java

Copy this file to the working directory used for these steps.

9.5.8 Modify the WebSphere MQ sample JMS application

This step modifies the JMS sample to access the same directory, through JNDI, as the WebSphere MQ JMS Administration tool that created objects within it.

It also changes the names of the objects accessed within this directory to use our basic naming convention, rather than an LDAP naming convention. The details of the LDAP naming convention are beyond the scope of this book.

Perform the following steps:

1. Open the JMSPubSub.java file in a text editor.
2. Locate the following lines in the file (located at line 88 in the WebSphere MQ V6.0 sample):

```
String CTX_FACTORY = "com.sun.jndi.ldap.LdapCtxFactory";
String INIT_URL    = "ldap://polaris/cn=PubSub,o=ibm_us,c=us";
```

3. Change these lines to the following lines to match the WebSphere MQ JMS Administration tool configuration:

```
String CTX_FACTORY = "com.sun.jndi.fscontext.RefFSContextFactory";
String INIT_URL    = "file:jms.provider";
```

4. Locate the following lines in the file (located at line 98 in the WebSphere MQ V6.0 sample):

```
TopicConnectionFactory tcf =
                   (TopicConnectionFactory)ctx.lookup( "cn=PubSub.TCF" );
```

5. Change these lines to the following lines to match the name of the Topic Connection Factory object defined in the WebSphere MQ JMS Administration tool:

```
TopicConnectionFactory tcf =
                   (TopicConnectionFactory)ctx.lookup( "PubSub.TCF" );
```

6. Locate the following line in the file (located at line 112 in the WebSphere MQ V6.0 sample):

```
Topic t = (Topic)ctx.lookup( "cn=PubSub.T" );
```

7. Change this line to the following line to match the name of the Topic object defined in the WebSphere MQ JMS Administration tool:

```
Topic t = (Topic)ctx.lookup( "PubSub.T" );
```

8. Save the file.

9.5.9 Compile the sample application

The WebSphere MQ JMS publish/subscribe sample is compiled as follows:

```
javac JMSPubSub.java
```

This creates a file called JMSPubSub.class in the working directory.

9.5.10 Start the sample as a subscriber

The WebSphere MQ JMS publish/subscribe sample can be started as either a publisher or a subscriber. Multiple instances of publishers can be started, and multiple instances of subscribers can be started.

To start an instance of the sample as a subscriber, perform the following steps:

1. Open a command window or terminal session to execute the publishing instance. Configure this as described in 9.5.1, “Configure the JMS environment” on page 255.
2. Execute the following command to start the sample as a subscriber:
   - Windows:

     ```
     java JMSPubSub -sub
     ```
   - UNIX (except AIX 5L):

     ```
     java -D"java.library.path=/opt/mqm/java/lib" JMSPubSub -sub
     ```
   - AIX 5L:

     ```
     java -D"java.library.path=/usr/mqm/java/lib" JMSPubSub -sub
     ```

   The initial output is as follows:

   ```
   [R]eceiveBlock, Receive[N]oWait, Receive[5]Secs, [Q]uit?
   ```
3. To cause the subscriber to wait for the next message on the topic to be published, press R and then Enter.

   Note: This needs to be pressed again after each publication is received.

9.5.11 Start the sample as a publisher

The WebSphere MQ JMS publish/subscribe sample can be started as either a publisher or a subscriber. Multiple instances of publishers can be started, and multiple instances of subscribers can be started.

To start an instance of the sample as a publisher, perform the following steps:

1. Open a command window or terminal session to execute the publishing instance. Configure this as described in 9.5.1, “Configure the JMS environment” on page 255.

2. Execute the following command to start the sample as a publisher:
   – Windows:
     ```
     java JMSPubSub -pub
     ```
   – UNIX (except AIX 5L):
     ```
     java -D"java.library.path=/opt/mqm/java/lib" JMSPubSub -pub
     ```
   – AIX 5L:
     ```
     java -D"java.library.path=/usr/mqm/java/lib" JMSPubSub -pub
     ```

   The initial output is as follows:
   ```
   [P]ublish message, [Q]uit?
   ```
3. To publish a message, press P and then Enter.
4. Type the message to be published and then press Enter.

The message is published to all waiting subscribers. The following output shows example output from a subscriber receiving a message:

```
JMS Message class: jms_text
  JMSType:         null
  JMSDeliveryMode: 2
  JMSExpiration:   0
  JMSPriority:     4
  JMSMessageID:    ID:414d5120686f7374312f6a6d73707562d1dcdd4220001e08
  JMSTimestamp:    1121839520615
  JMSCorrelationID:ID:414d5120686f7374312f6a6d73707562d1dcdd4220006f05
  JMSDestination:  topic://MQJMS/Samples/PubSub
  JMSReplyTo:      null
  JMSRedelivered:  false
  JMS_IBM_PutDate:20050720
  JMSXAppID:host1/jmspubsub
  JMS_IBM_Format:MQSTR
  JMS_IBM_PutApplType:26
  JMS_IBM_MsgType:8
  JMSXUserID:pbroad
  JMS_IBM_PutTime:06052061
  JMSXDeliveryCount:1
Redbook test message
```

10

Hands-on guide to building WebSphere MQ infrastructure

In this chapter, you extend the local infrastructure built in the previous chapter. You access queue managers as clients to those queue managers, including performing remote administration using the WebSphere MQ Explorer. You build a hub and spoke infrastructure, manually defining channels between one or more spoke queue managers connected to the hub queue manager created in the previous chapter. The service provided by the hub is accessed through the spokes of the hub and spoke infrastructure. You build a queue manager cluster, seeing how administration is simplified and the queue manager cluster provides additional workload balancing features. These hub and spoke and queue manager cluster infrastructures are interconnected, sharing access to the service between them.

We discuss the following topics:

- Environment setup
- Connect as a client to a queue manager
- Build a hub and spoke infrastructure
- Create a queue manager cluster

10.1 Environment setup

The same environment is required in this chapter as in Chapter 9, “Hands-on introduction to messaging with WebSphere MQ” on page 217.

The only addition is that this chapter uses network connections over a TCP/IP network to interconnect applications and queue managers.

Multiple machines can be used, and the host names and IP addresses used in this chapter reflect this. However, all tasks in this chapter can be performed using a single Windows or Linux desktop workstation, and no external network access, or network configuration, is required.

Note: If you are using a single desktop machine that does not have an identity on a wider network, replace *all* of the host names in this chapter with the following host name, which identifies your local machine:

```
localhost
```

This includes the following host names provided as examples in this chapter:

```
host1.example.com
host2.example.com
```

In this chapter, all port numbers on which queue managers listen are unique. This means that the same port numbers can be used whether the queue managers are all on the same machine or on multiple machines.

10.2 Connect as a client to a queue manager

This section shows how an application can connect to a queue manager remotely, gaining access to the same functionality as is available to an application connected on the same machine.

These steps show this by allowing you to access and administer the queue manager host1/echo.hub created in 9.4, “Host a request/reply service on a queue” on page 243. This enables you to access the service hosted by that queue manager over a client connection.

Alternatively, you can create and start a new queue manager to which to connect. If doing this, substitute `host1/echo.hub` with the name of that queue manager in all steps.

Note: All of these steps can be performed on one machine. Simply substitute `host1.example.com` for the host name or IP address of the machine you are using.

You can use the generic host name localhost for your machine if you do not have a host name or IP address on a network or if your IP address is likely to change.

10.2.1 Create and start a listener

This step creates and starts a listener for the queue manager, which provides an identity for that queue manager in the network. In WebSphere MQ V6.0, listeners are WebSphere MQ objects defined on a queue manager. For this step, you can use the WebSphere MQ Explorer or MQSC commands.

Note: This step assumes that you have a WebSphere MQ V6.0 queue manager. Listeners are created and started differently for queue managers in previous versions.

The listener listens on a particular port on the TCP/IP network. A port is a fundamental concept of TCP/IP networks. There are a large range of ports that can be listened on by the network services provided by a machine. These instructions assume that no network services are currently listening on the ports used. If you know that a network service is listening on a port suggested, choose a different port number, remembering to substitute that port number in all future steps.

If there is only one queue manager on a system, it is often chosen for this queue manager to listen on port 1414. This is the well-known port for WebSphere MQ. These instructions choose an arbitrary port range that is unlikely to be used by other queue managers, or network services, on the machine.

Using the WebSphere MQ Explorer

Perform the following steps:

1. Right-click the **Listeners** folder for the queue manager, and select **New** → **TCP Listener**.

Note: The Listeners folder is shown under the Advanced folder for a queue manager in the navigator view.

If you selected the **Create listener configured for TCP/IP** check box in Enter listener options (Step 4) of the Create Queue Manager wizard, a listener object might already exist called LISTENER.TCP.

If so, right-click the listener object in the table and select **Stop**. After the listener stops, right-click again, select **Delete**, and then confirm the deletion.

2. Enter `LISTENER.TCP` in the name field.
3. Click **Next**.
4. Enter the following value in the Description field:

   ```
   TCP/IP Listener for queue manager
   ```
5. Enter `9001` in the Port field.
6. Change the Control field to **Queue Manager** to cause the listener to be automatically started and stopped with the queue manager in the future.
7. Click **Finish**.
8. Right-click the **LISTENER.TCP** entry created in the table, and select **Start**. This starts the listener.
9. Check that the Listener status column contains Running. If not, click the **Refresh** button in the top-right corner of the display and check the status again.

Note: If the status still contains Stopped, it is likely that another queue manager defined on the system already has a listener running on port 1414. Check the LISTENER.TCP listener object of each queue manager defined.

Each queue manager on a machine *must* listen on a different TCP/IP port number.

Using MQSC commands

Perform the following steps:

1. Create a listener that is started and stopped automatically with the queue manager by issuing the following MQSC command against host1/echo.hub:

   ```
   DEFINE LISTENER('LISTENER.TCP') +
         TRPTYPE(TCP) PORT(9001) CONTROL(QMGR) +
         DESCR('TCP/IP Listener for queue manager')
   ```

2. Start the listener using the following MQSC command:

   ```
   START LISTENER('LISTENER.TCP')
   ```

Note: If you are using the WebSphere MQ Explorer, you might want to use the **Create listener configured for TCP/IP** check box in Enter listener options (Step 4) of the Create Queue Manager wizard when creating future queue managers that require listeners.

Ensure that you check the port number before clicking Finish. The port of the listener object can be changed after the queue manager is created. However, ensure that you stop and restart the listener object if you change the port number.

10.2.2 Create a server-connection channel object

A server-connection channel object defines a channel name and channel attributes for a client connection that can be established to a queue manager.

One significant attribute that can be configured on a server connection channel object is the local user identifier that remote applications assume when they connect over that channel. This is configured using the MCA user identifier (`MCAUSER`).

This is useful, because applications accessing the queue manager from remote machines might run under different user identifiers, for example, if one machine is a UNIX machine and another is a Windows machine. In these steps, you can optionally configure this to the same user identifier as which you are logged into the machine. This allows applications from any machine to access the machine with the same WebSphere MQ administrator authority as you have.

This step creates a server connection channel object called all.clients that is used throughout this section by applications connecting to the queue manager. For this step, you can use the WebSphere MQ Explorer or MQSC commands.

Using the WebSphere MQ Explorer

Perform the following steps:

1. Right-click the **Channels** folder for host1/echo.hub, and select **New** → **Server-connection Channel**.
2. Type `all.clients` in the Name field.
3. Click **Next**.
4. Optionally, select the **MCA** section, and specify your user identifier in the MCA user ID field.
5. Click **Finish**.

Using MQSC commands

Issue the following MQSC command against host1/echo.hub:

```
DEFINE CHANNEL('all.clients') CHLTYPE(SVRCONN) MCAUSER('userid')
```

10.2.3 Connect using the MQSERVER environment variable

The client message channel agent (MCA) provided when using the MQI directly, for example, from applications written in C, can be configured by using environment variables. The basic attributes, including the channel name and connection name, are configured using the `MQSERVER` environment variable.

Applications that use this core client MCA are built against a different set of WebSphere MQ libraries than those which that to a queue manager directly. These are generally called the WebSphere MQ *client libraries*.

The base WebSphere MQ sample programs, such as **`amqsput`**, **`amqsget`**, **`amqsbcg`**, **`amqsreq`**, and **`amqsech`**, are all supplied with WebSphere MQ in versions that are prebuilt against the WebSphere MQ client libraries. These are invoked by adding a `c` to the end of the application name, for example, **`amqsputc`**, **`amqsgetc`**, **`amqsbcgc`**, **`amqsreqc`**, and **`amqsechc`**.

Note: The client MCAs provided for use from other APIs, such as Java, JMS, .NET, and XMS, are configured in different ways. For details, see the appropriate WebSphere MQ guide for the API you are using.

The queue manager name specified when connecting to a queue manager as a client usually matches the queue manager name. However, an asterisk (*) character can be used to specify to connect to any queue manager.

To put and get messages from a queue as a client to a queue manager using the `MQSERVER` environment variable and the WebSphere MQ sample programs, perform the following steps:

1. Define a local queue on the queue manager, using a name of your choosing. This step assumes queue1.
2. Set the `MQSERVER` environment variable as follows:
   - Windows:
     ```
     set MQSERVER=all.clients/TCP/host1.example.com(9001)
     ```
   - UNIX:
     ```
     MQSERVER=all.clients/TCP/'host1.example.com(9001)'
     export MQSERVER
     ```
3. Put messages onto the queue using **amqsputc**:
   ```
   amqsputc queue1 host1/echo.hub
   ```
4. Browse messages on the queue using **amqsbcgc**:
   ```
   amqsbcgc queue1 host1/echo.hub > queue1.txt
   ```
5. Get messages from the queue using **amqsgetc**:
   ```
   amqsgetc queue1 host1/echo.hub
   ```
6. Issue requests against the triggered echo service on host1/echo.hub:
   ```
   amqsreqc echo host1/echo.hub echo.replies.manual
   ```

Note: If you experience problems, such as 2058 or 2059 return codes, check the queue manager and WebSphere MQ system error logs, as described in 5.3.15, "Error logs" on page 119.

10.2.4 Connect using a client-connection channel object

Client-connection channel objects are provided by WebSphere MQ to allow full configuration of the attributes of a client MCA.

Defining a client-connection channel object within a queue manager creates an entry in a file called the client channel definition table (CCDT) for that queue manager. This can be used by clients to gain information about available queue managers. This file can be created on *any* queue manager, not necessarily the one to which an application connects. It can then be copied to remote machines, or be made available over the network.

This step configures the base WebSphere MQ samples to use the channel configuration contained within a CCDT when connecting to a queue manager.

Perform the following steps:

1. Create two server-connection channels on host1/echo.hub, as described in 10.2.2, “Create a server-connection channel object” on page 269. The names of these channels should be `client.channel1` and `client.channel2`. Alternatively, you can define these two channels on different queue managers that have listeners configured on different ports of the same machine or on different machines.

2. Create two client-connection channel objects, both on queue manager host1/echo.hub, matching the names of the server-connection channel objects defined.

   – Using the WebSphere MQ Explorer:

     i. Right-click the **Client Connections** folder for host1/echo.hub, and select **New** → **Client-connection Channel**.

     ii. Type `client.channel1` into the Name field.

     iii. Click **Next**.

     iv. Enter `echo.hub` into the Queue manager name field. This is intentionally not the actual name of the queue manager.

     v. Enter *`host1.example.com`*`(9001)` into the Connection name field.

     vi. Click **Finish**.

     vii. Repeat steps i to vi to create another client-connection channel. Call this second channel `client.channel2`. The queue manager name attribute is `echo.hub`. The connection name can be the same if both server-connection channel objects are on the same queue manager or the connection name of a second queue manager.

   – Using MQSC commands:

     Issue the following MQSC commands against host1/echo.hub:

```
DEFINE CHL('client.channel1') CHLTYPE(CLNTCONN) +
      QMNAME('echo.hub') CONNAME('host1.example.com(9001)')
DEFINE CHL('client.channel2') CHLTYPE(CLNTCONN) +
      QMNAME('echo.hub') CONNAME('host1.example.com(9001)')
```

     The connection name (`CONNAME`) attribute for the second definition can be the connection name of a different queue manager, where a server-connection channel called client.channel2 is defined.

3. This step assumes the client applications are on the same machine as the host1/echo.hub queue manager. If not, copy the CCDT across to the other machine, and change the environment variables below to reflect the location local to the client applications.

To configure the core client MCA used by the WebSphere MQ samples to find this CCDT, specify the following environment variables:

– Windows:

```
set MQSERVER=
set "MQCHLLIB=C:\Program Files\IBM\WebSphere MQ\Qmgrs\host1&echo!hub\@ipcc"
set MQCHLTAB=AMQCLCHL.TAB
```

– UNIX:

```
unset MQSERVER
MQCHLLIB='/var/mqm/qmgrs/host1&echo!hub/@ipcc'
MQCHLTAB=AMQCLCHL.TAB
export MQCHLLIB
export MQCHLTAB
```

4. Run the **`amqsputc`** sample program against a queue defined on host1/echo.hub:

   ```
   amqsputc queue1 *echo.hub
   ```

   Note: The asterisk (*) character specifies that the application does not require a connection to a specific queue manager. Instead, any entries in the CCDT that have a queue manager name attribute of `echo.hub` are valid for use.

5. Leave the **`amqsputc`** sample waiting for input so that it remains connected to the queue manager.
6. Display which channel is being used. If using two queue managers, perform this on both queue managers:

   – Using the WebSphere MQ Explorer:

   Select the **Channels** folder (for host1/echo.hub). Notice that the status of server-connection channel client.channel1 or client.channel2 is running.

   Note: For more information about the connection, you can right-click **host1/echo.hub** in the navigator view, and select **Application Connections**. Then, select the connection in which you are interested.

   – Using MQSC commands:

   Issue the following MQSC command, and notice that one of the server-connection channels has a STATUS of `RUNNING`:

   ```
   DIS CHSTATUS('client.*')
   ```

7. Enter a blank line into the input of the running **`amqsputc`** command to end it.

8. Disable the server-connection channel previously used by the command:
   - Using the WebSphere MQ Explorer:

     Select the **Channels** folder (for host1/echo.hub). Right-click the channel and select **Stop**. Ensure that **Stopped** is selected in the New State.
   - Using MQSC commands:

     Execute the following command against host1/echo.hub, specifying the name of the channel which was previously running:

     ```
     STOP CHANNEL('client.channel1') STATUS(STOPPED)
     ```
9. Run the **`amqsputc`** command again. Notice that the command can still operate even though the channel previously used is not available. This is because the second entry in the CCDT is being used for the connection. This can be to a different queue manager, as previously discussed.

10.2.5 Perform remote administration of a queue manager

This step shows how the WebSphere MQ Explorer can be used to perform remote administration of queue managers using a client connection to those queue managers.

In this example, it is likely that the queue manager is on the same machine as the WebSphere MQ Explorer. However, the WebSphere MQ Explorer can remotely administer queue managers on multiple different remote machines, including different platforms. WebSphere MQ for z/OS V6.0 queue managers can also be administered using the WebSphere MQ Explorer.

Note: Queue managers that are remotely administered using the WebSphere MQ Explorer must have:

- A listener defined nd running on a known port.
- A server-connection channel defined of a known name.
- A command server running.
- A model queue defined called SYSTEM.MQEXPLORER.REPLY.MODEL.

For Windows and UNIX, queue managers created with WebSphere MQ V6.0 have a running command server and the correct model queue by default. However, queue managers created with WebSphere MQ V5.3, and previous versions, require these to be configured manually. This includes queue managers created with WebSphere MQ V5.3 and migrated to WebSphere MQ V6.0. To do this:

1. Use the following WebSphere MQ control command to manually start the command server for the queue manager:

   ```
   strmqcsv Queue_Manager_Name
   ```

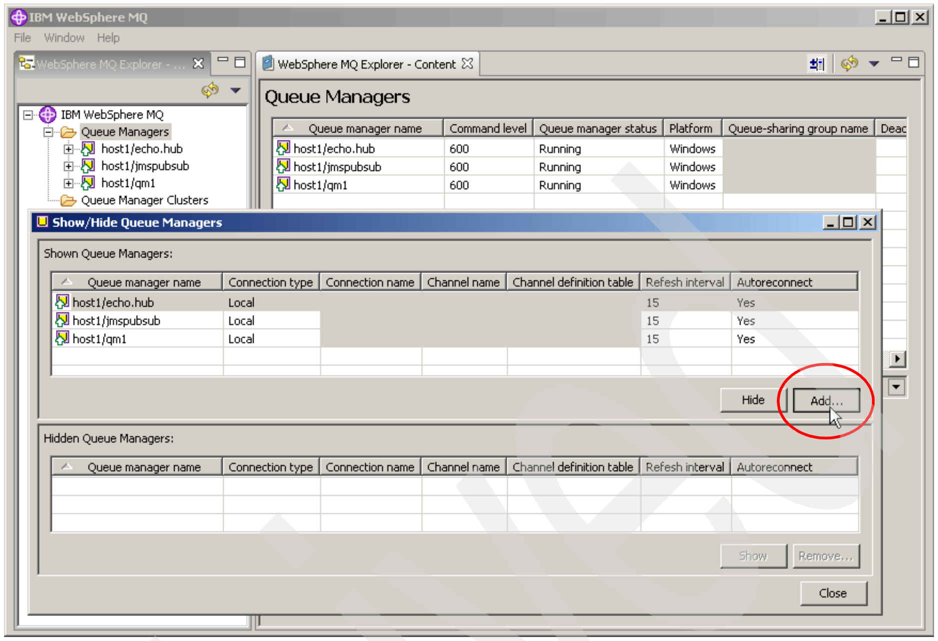
2. Define the required model queue in MQSC:

   ```
   DEFINE QMODEL('SYSTEM.MQEXPLORER.REPLY.MODEL') DEFTYPE(TEMPDYN)
   ```

Perform the following steps:

1. Right-click the **Queue Managers** folder in the WebSphere MQ Explorer, and select **Show/Hide Queue Managers**.

   Figure 10-1 on page 276 shows the Show/Hide Queue Managers window.



Queue Managers

Queue manager name	Command level	Queue manager status	Platform	Queue-sharing group name	Deac
host1/echo.hub	600	Running	Windows		
host1/jmspubsub	600	Running	Windows		
host1/qm1	600	Running	Windows		

Show/Hide Queue Managers

Shown Queue Managers:

Queue manager name	Connection type	Connection name	Channel name	Channel definition table	Refesh interval	Autoreconnect
host1/echo.hub	Local				15	Yes
host1/jmspubsub	Local				15	Yes
host1/qm1	Local				15	Yes

Hidden Queue Managers:

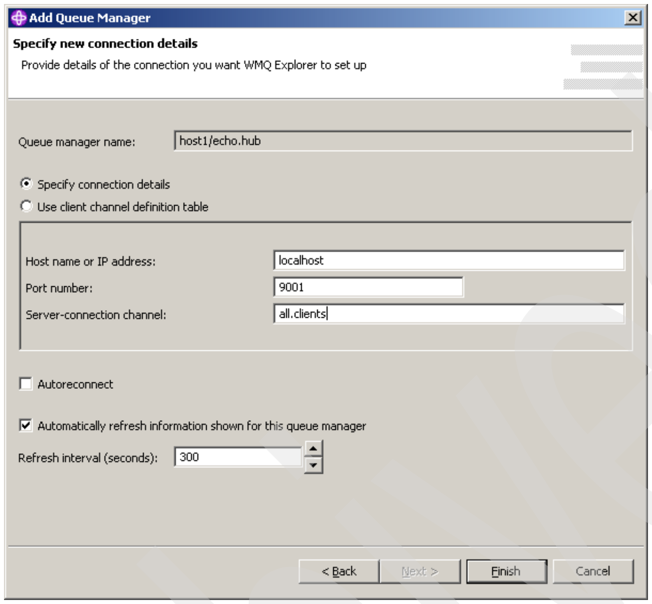
Queue manager name	Connection type	Connection name	Channel name	Channel definition table	Refesh interval	Autoreconnect

Figure 10-1 WebSphere MQ Explorer Show/Hide Queue Managers window

2. In the Show/Hide Queue Managers window, click **Add**. The Add Queue Manager wizard opens, where you specify the way the WebSphere MQ Explorer connects to the queue manager. In this example, we connect by specifying the same information as was provided to the WebSphere MQ sample programs in 10.2.3, "Connect using the MQSERVER environment variable" on page 270.
3. On the first page of the wizard, enter the name of the queue manager into the Queue manager name field, for example, `host1/echo.hub`.
4. Ensure that **Connect directly** is selected.
5. Click **Next**.
6. Specify the host name or IP address of the computer on which host1/echo.hub is running. We use `host1.example.com` as an example in these instructions, and this can be localhost for the local machine.
7. Specify the port on which the listener is running for the queue manager, for example, `9001`.
8. Specify the name of the server-connection channel object defined on the queue manager, which is `all.clients` in this example.

9. Click **Finish**.

   Figure 10-2 shows an example of the information to provide.



Add Queue Manager

Specify new connection details

Provide details of the connection you want WMQ Explorer to set up

Queue manager name: host1/echo.hub

(●) Specify connection details
( ) Use client channel definition table

Host name or IP address: localhost
Port number: 9001
Server-connection channel: all.clients

[ ] Autoreconnect

[✓] Automatically refresh information shown for this queue manager

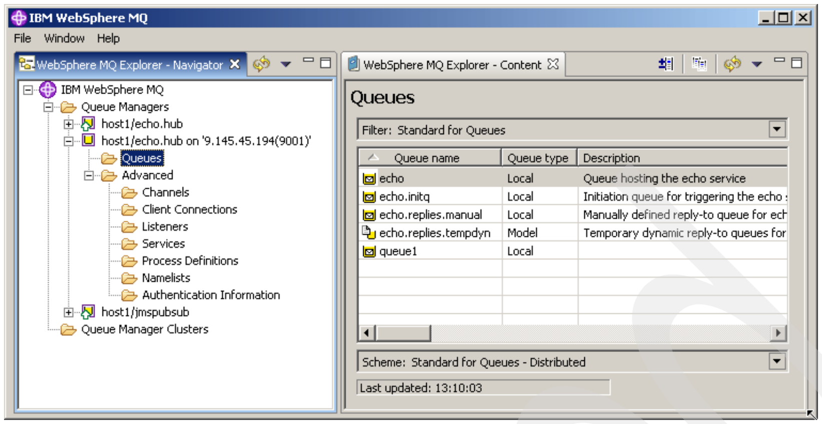
Refresh interval (seconds): 300

Figure 10-2 Add Queue Manager wizard for direct connection to queue manager

   This adds the queue manager to the Shown Queue Managers table in the Show/Hide Queue Managers window.

10. Click **Close** on the Show/Hide Queue Managers window.

11. Notice that the queue manager is available under the Queue Managers folder in the same way as the locally connected queue managers. The name of the queue manager is appended with the connection name for the queue manager to show that it is remote. This icon is different from the locally connected queue managers. Figure 10-3 on page 278 provides an example.


IBM WebSphere MQ
File Window Help
WebSphere MQ Explorer - Navigator
WebSphere MQ Explorer - Content

IBM WebSphere MQ
- Queue Managers
  - host1/echo.hub
  - host1/echo.hub on '9.145.45.194(9001)'
    - Queues
    - Advanced
      - Channels
      - Client Connections
      - Listeners
      - Services
      - Process Definitions
      - Namelists
      - Authentication Information
  - host1/jmspubsub
- Queue Manager Clusters

Queues

Filter: Standard for Queues

Queue name	Queue type	Description
echo	Local	Queue hosting the echo service
echo.initq	Local	Initiation queue for triggering the echo :
echo.replies.manual	Local	Manually defined reply-to queue for ech
echo.replies.tempdyn	Model	Temporary dynamic reply-to queues for
queue1	Local	

Scheme: Standard for Queues - Distributed

Last updated: 13:10:03

Figure 10-3 WebSphere MQ Explorer remotely administering a queue manager

10.2.6 JMS publish/subscribe sample using a client connection

In order to modify the JMS publish/subscribe example, set up in 9.5, "WebSphere MQ publish/subscribe with JMS" on page 255, to use a client connection to the queue manager, modification is only required to the objects within the directory as accessed using JNDI. No modification is required of the sample itself or the way it is invoked.

This step changes the configuration of the objects within the directory accessed by JNDI to cause the JMS sample to connect as a client to the queue manager.

Perform the following steps:

1. Ensure that the current command window or terminal session is configured as described in 9.5.1, "Configure the JMS environment" on page 255.
2. Configure a listener for queue manager `host1/jmspubsub` on port `9010`, as described in 10.2.1, "Create and start a listener" on page 267.
3. Define a server connection channel object on queue manager host1/jmspubsub called `jms.clients`.
4. Start the WebSphere MQ JMS Administration tool, as described in 9.5.6, "Use WebSphere MQ JMS Administration tool to configure JMS" on page 260.

5. Alter the Topic Connection Factory to specify a client connection to connect remotely to the queue manager. Previously, the Topic Connection Factory used a bindings connection to connect locally to the queue manager. This is performed by issuing the following command in the WebSphere MQ JMS Administration tool (all one line):

   ```
   ALTER TCF(PubSub.TCF) HOSTNAME(host1.example.com) PORT(9010)
                                          TRANSPORT(CLIENT) CHANNEL(jms.clients)
   ```

6. Exit the WebSphere MQ JMS Administration tool using END.
7. Without performing any modification of the Java sample, or recompiling, run the sample as publishers and subscribers. We describe this in 9.5.10, “Start the sample as a subscriber” on page 263 and 9.5.11, “Start the sample as a publisher” on page 263.

10.3 Build a hub and spoke infrastructure

This section makes queue manager host1/echo.hub the hub of a hub and spoke infrastructure. We create spoke queue managers and manually configure intercommunication between the hub and spoke queue managers.

All spoke queue managers are then able to access the echo service provided by the host1/echo.hub queue manager.

10.3.1 Create a dead letter queue on the hub queue manager

This step configures the host1/echo.hub queue manager to have a dead letter queue. This is important to ensure that nonpersistent messages do not get lost if the destination is incorrectly specified. This can be frustrating when beginning to use channels and trying to find messages placed into the infrastructure. For this step, you can use the WebSphere MQ Explorer or MQSC commands.

Using the WebSphere MQ Explorer

Perform the following steps:

1. Right-click the **Queues** folder for host1/echo.hub, and select **New** → **Local Queue**.
2. Type `dead.letters` in the Name field.
3. Click **Finish**.
4. Right-click the icon for queue manager host1/echo.hub and select **Properties**.
5. Select the **Extended** section of the properties window.

6. Type `dead.letters` into the Dead letter queue field.
7. Click **OK**.

Using MQSC commands

Perform the following steps:

1. Issue the following MQSC command against host1/echo.hub to create a local queue object:

   ```
   DEFINE QLOCAL('dead.letters')
   ```

2. Issue the following MQSC command against host1/echo.hub to configure the queue manager object to use the dead letter queue created:

   ```
   ALTER QMGR DEADQ('dead.letters')
   ```

Note: Make sure that the `DEADQ` attribute of the queue manager object exactly matches the name of the local queue object defined.

10.3.2 Create a receiver channel object on the hub queue manager

In this step, we define a channel object on the hub queue manager to allow channels to be established from the spoke queue managers in the infrastructure. This example uses a single channel object of a receiver type to allow connections from all spoke queue managers in the infrastructure. Later steps configure communication to and from the hub queue manager for each spoke individually.

For this step, you can use the WebSphere MQ Explorer or MQSC commands.

Using the WebSphere MQ Explorer

Perform the following steps:

1. Right-click the **Channels** folder for host1/echo.hub, and select **New** → **Receiver Channel**.
2. Type `to.host1/echo.hub` in the Name field.
3. Click **Finish**.

Using MQSC commands

1. Issue the following MQSC command against host1/echo.hub:

   ```
   DEFINE CHANNEL('to.host1/echo.hub') CHLTYPE(RCVR)
   ```

2. Check the attributes of the channel using the following MQSC command:

   ```
   DISPLAY CHANNEL('to.host1/echo.hub')
   ```

10.3.3 Create and start a spoke queue manager with a listener

We recommend that you run through the instructions from this point once, adding a single spoke queue manager. Then, you can repeat them using different queue managers, and connection names for the spokes.

Create and start a new queue manager. The first spoke queue manager created is assumed to have the name host2/spoke.

If more queue managers are created, increase the number in the queue manager name or replace host2 with the name of the machine if you are using multiple machines.

To distinguish the machine on which a spoke queue manager is running from the machine on which the hub is running, these steps use the host namehost2.example.com.

Replace this name with the actual host name or IP address of the machine. As in previous steps, all queue managers can be on the same machine. In this case, replace `host2.example.com` in all steps with the host name or IP address of the local machine, or `localhost`.

Each queue manager must have a listener defined. Because a number of these queue managers can be hosted on the same machine, we recommend that the port number is relevant to the queue manager name. The first spoke queue manager created should have a listener running on the 9002 port.

We recommend that you increase the port number by one for each queue manager to avoid confusion.

Note: Different queue managers on different machines can have a listener running on the same port number. However, two queue managers on the same machine cannot.

We also recommend that you define a dead letter queue on each spoke queue manager. Refer to 10.3.1, “Create a dead letter queue on the hub queue manager” on page 279, replacing `host1/echo.hub` with the name of the spoke queue manager.

10.3.4 Create a transmission queue on the spoke queue manager

Creating a transmission queue gives a queue manager a temporary place to store messages that are transferred to another queue manager within the infrastructure. However, a transmission queue does not, in itself, cause messages to be transferred to that queue manager.

A transmission queue is a local queue object that has been specified to be used as a transmission queue.

For this step, you can use the WebSphere MQ Explorer or MQSC commands.

Using the WebSphere MQ Explorer

Perform the following steps:

1. Right-click the **Queues** folder for queue manager host2/spoke, and select **New** → **Local Queue**.
2. Type `host1/echo.hub` in the Name field.
3. Click **Next**.
4. Select **Transmission** in the Usage field.
5. Enter the following value in the Description field:

   ```
   Transmission queue for messages to host1/echo.hub
   ```
6. Click **Finish**.

Using MQSC commands

Issue the following MQSC command against host2/spoke:

```
DEFINE QLOCAL('host1/echo.hub') USAGE(XMITQ) +
       DESCR('Transmission queue for messages to host1/echo.hub')
```

Note: Ensure that the case of the queue manager name is correct and is enclosed in single quotation marks.

10.3.5 Create a sender channel object on the spoke queue manager

A sender channel object allows a channel to be established from a queue manager to another queue manager in the infrastructure. The name of the sender channel object must match the name of a remote channel object on the destination queue manager of a compatible type. A receiver channel object has already been defined on the hub queue manager, which is used by all connecting spoke queue managers.

A channel takes messages from a single transmission queue on one queue manager and transfers them to the remote queue manager. For this step, you can use the WebSphere MQ Explorer or MQSC commands.

Using the WebSphere MQ Explorer

Perform the following steps:

1. Right-click the **Channels** folder for queue manager host2/spoke, and select **New** → **Sender Channel**.
2. Type `to.host1/echo.hub` in the Name field.
3. Click **Next**.
4. The sender channel object is used to establish channels *to* the hub queue manager, so specify the Connection name field as follows:

   ```
   host1.example.com(9001)
   ```
5. Type `host1/echo.hub` in the Transmission queue field.
6. Click **Finish**.

Using MQSC commands

Perform the following steps:

1. Issue the following MQSC command against host2/spoke:

   ```
   DEFINE CHANNEL('to.host1/echo.hub') CHLTYPE(SDR) +
          CONNAME('host1.example.com(9001)') XMITQ('host1/echo.hub')
   ```
2. Check the attributes of the channel using the following MQSC command:

   ```
   DISPLAY CHANNEL('to.host1/echo.hub')
   ```

10.3.6 Test the channel using a WebSphere MQ ping

This step issues a WebSphere MQ **ping** command to test that communication can occur across a channel. However, it does not cause messages to start to flow across the channel. For this step, you can use the WebSphere MQ Explorer or MQSC commands.

Using the WebSphere MQ Explorer

Perform the following steps:

1. Select the **Channels** folder for queue manager host2/spoke.
2. Right-click the entry in the table called **to.host1/echo.hub** and select **Ping**.
3. A window opens to show the result.

Using MQSC commands

Issue the following MQSC command against host2/spoke:

```
PING CHANNEL('to.host1/echo.hub')
```

Note: If an error is returned by the **ping** command, check the following information:

- The connection name in the sender channel object has the correct host name or IP address and port number or the host1/echo.hub queue manager.
- The name of the sender channel object matches the name of the receiver channel object defined on host1/echo.hub, including the case.
- A listener is running on the host1/echo.hub queue manager.

10.3.7 Configure the channel to the hub to be initiated

This step causes the channel from the spoke queue manager to the hub queue manager to be automatically started by the WebSphere MQ channel initiator of the queue manager when messages arrive on the transmission queue. For this step, you can use the WebSphere MQ Explorer or MQSC commands.

Using the WebSphere MQ Explorer

Perform the following steps:

1. Select the **Queues** folder for host2/spoke.
2. Right-click the entry for queue **host1/echo.hub** and select **Properties**.
3. Select the **Triggering** section of the properties window.
4. Change the Trigger control field to **On**.
5. Ensure that the Trigger type field contains **First**.
6. Type `to.host1/echo.hub` in the Trigger data field.
7. Type `SYSTEM.CHANNEL.INITQ` in the Initiation queue field
8. Click **OK**.

Using MQSC commands

Issue the following MQSC command against host2/spoke:

```
ALTER QLOCAL('host1/echo.hub') TRIGGER TRIGTYPE(FIRST) +
     TRIGDATA('to.host1/echo.hub') INITQ('SYSTEM.CHANNEL.INITQ')
```

Note: It is important that all of these attributes are correct in order for the channel initiation to occur.

10.3.8 Put a test message through the channel to the hub

This step sends a test message through the channel to the hub queue manager. In this step, we manually specify that the destination of the message is on host1/echo.hub so that the queue manager does not need any local knowledge of the queue. The queue manager knows about the host1/echo.hub queue manager, because it has a transmission queue of the correct name. It is by this mechanism that replies are sent through a WebSphere MQ infrastructure to a specific queue on a specific queue manager.

Perform the following steps:

1. Issue the following WebSphere MQ sample command, and type messages to send to the queue hosted on the hub queue manager:

   ```
   amqsput queue1 host2/spoke 8208 0 host1/echo.hub
   ```

   The following parameters are passed to this sample:

   – `queue1` is the name of the queue on host1/echo.hub.

   – `host2/spoke` is the name of the queue manager being connected to.

   – `8208` is a decimal value requesting options to be passed to the MQOPEN call. *This is not important to the example.* It is requesting that the queue is opened for output to put messages and to fail any subsequent puts if the queue manager is quiescing.

   – `0` requests that no options are passed to the MQCLOSE call. *This is not important to the example.*

   – `host1/echo.hub` is the name of the object queue manager specified in the MQOPEN call. This is the parameter used in this example. You can see that this name matches both the name of the remote queue manager and the name of the transmission queue on host2/spoke used to send messages to that queue manager.

2. The message is delivered onto the transmission queue by the spoke queue manager. This causes a trigger message to be passed to the queue manager's channel initiator through the SYSTEM.CHANNEL.INITQ. The channel starts, and the messages flow across to the remote queue manager. The remote message channel agent (MCA), created based on the receiver channel object on host1/echo.hub, puts this message to the specified queue on that queue manager.

Note: Ensure that the queue name specified exists as a local queue object on the hub queue manager. If you are unable to find the message on the transmission queue at the spoke side, or the target queue at the remote side, browse the dead.letters dead letter queue configured for the hub queue manager. If you do this using the WebSphere MQ Explorer, you can see the contents of the dead letter header for any messages that are sent there.

At the sender side, you can view the status of the channel in the WebSphere MQ Explorer using the Status column of the row in the Channels table. Alternatively, you can view the status using the following MQSC command:

```
DISPLAY CHSTATUS('to.host1/echo.hub')
```

You can manually start a channel from the sender side by right-clicking the channel and selecting **Start**. Alternatively, you can start the channel using the following MQSC command:

```
START CHANNEL('to.host1/echo.hub')
```

To stop a channel, but not disable it from being started by the channel initiator, select **Inactive** from the New State drop-down list when stopping a channel in the WebSphere MQ Explorer. Alternatively, stop a channel, without disabling it, using the following MQSC command:

```
STOP CHANNEL('to.host1/echo.hub') MODE(INACTIVE)
```

10.3.9 Create a receiver channel object on the spoke queue manager

Communication has been successfully established between the spoke queue manager and the hub. Later, this is used to issue requests against the service hosted by that queue manager. However, in order for the replies to be routed back to the requesting application, communication must be established between the hub and each spoke queue manager.

This step defines a receiver channel object, which is used by the hub when establishing a channel to the spoke queue manager. For this step, you can use the WebSphere MQ Explorer or MQSC commands.

Using the WebSphere MQ Explorer

Perform the following steps:

1. Right-click the **Channels** folder for host2/spoke, and select **New** → **Receiver Channel**.
2. Type `to.host2/spoke` in the Name field.
3. Click **Finish**.

Using MQSC commands

Issue the following MQSC command against host1/echo.hub:

```
DEFINE CHANNEL('to.host2/spoke') CHLTYPE(RCVR)
```

10.3.10 Create a transmission queue on the hub queue manager

This step defines a transmission queue on the hub queue manager host1/echo.hub to provide a temporary queue for messages for the spoke queue manager. The triggering required for channel initiation is configured in this step, but the sender channel object on which channel initiation occurs is defined in subsequent steps. For this step, you can use the WebSphere MQ Explorer or MQSC commands.

Using the WebSphere MQ Explorer

Perform the following steps:

1. Right-click the **Queues** folder for queue manager host1/echo.hub, and select **New** → **Local Queue**.
2. Type `host2/spoke` in the Name field.
3. Click **Next**.
4. Select **Transmission** in the Usage field.
5. Enter the following value in the Description field:

   ```
   Transmission queue for messages to host2/spoke
   ```

6. Select the **Triggering** section.
7. Change the Trigger control field to **On**.
8. Ensure that the Trigger type field contains **First**.
9. Type `to.host2/spoke` in the Trigger data field. This channel is defined in subsequent steps.
10. Type `SYSTEM.CHANNEL.INITQ` in the Initiation queue field
11. Click **Finish**.

Using MQSC commands

Issue the following MQSC command against host1/echo.hub:

```
DEFINE QLOCAL('host2/spoke') USAGE(XMITQ) +
       TRIGGER TRIGTYPE(FIRST) +
       TRIGDATA('to.host2/spoke') INITQ('SYSTEM.CHANNEL.INITQ') +
       DESCR('Transmission queue for messages to host2/spoke')
```

10.3.11 Create a sender channel object on the hub queue manager

This step creates the sender channel object used to establish communication with the spoke queue manager.

For this step, you can use the WebSphere MQ Explorer or MQSC commands.

Using the WebSphere MQ Explorer

Perform the following steps:

1. Right-click the **Channels** folder for queue manager host1/echo.hub, and select **New** → **Sender Channel**.
2. Type `to.host2/spoke` in the Name field.
3. Click **Next**.
4. The sender channel object is used to establish channels *to* the hub queue manager, so specify the Connection name field as follows:

   host2.example.com(9002)
5. Type `host2/spoke` in the Transmission queue field.
6. Click **Finish**.

Using MQSC commands

Issue the following MQSC command against host2/spoke:

```
DEFINE CHANNEL('to.host2/spoke') CHLTYPE(SDR) +
      CONNAME('host1.example.com(9002)') XMITQ('host2/spoke')
```

Note: It is beneficial at this stage to ping the channel from the sender side at host1/echo.hub and to ensure that messages can flow across it. To send messages across the channel, you need to define a test queue on the spoke queue manager to which to send messages. Then, refer back to 10.3.8, "Put a test message through the channel to the hub" on page 285, and change the parameters on the command. For example:

```
amqsput queue1 host1/echo.hub 8208 0 host2/spoke
```

10.3.12 Create a local definition of a remote queue

Communication has been successfully established in both directions between the spoke queue manager and the hub queue manager. You have seen how messages can be explicitly targeted for the partner queue manager.

However, when requesting a service, the application should not specify an explicit queue manager name. This limits the ability of the infrastructure to change without making modifications to the requesting application.

This step creates a local definition on the spoke queue manager of a queue hosted on the hub queue manager. This is performed using a remote queue object. The local definition of the remote queue means that an application only needs to specify the queue name when sending messages. The infrastructure decides where that message is sent, rather than the application.

Note: Remote queue objects have a number of uses. This is one common use in a hub and spoke architecture, where channels are manually defined between queue managers.

For this step, you can use the WebSphere MQ Explorer or MQSC commands.

Using the WebSphere MQ Explorer

Perform the following steps:

1. Right-click the **Queues** folder for host2/spoke, and select **New** → **Remote Queue Definition**.
2. Type `echo` in the Name field.
3. Click **Next**.
4. Enter the following value in the Description field:

   ```
   Local definition for routing requests for the echo service
   ```
5. Type `echo` in the Remote queue field. This field enables the queue to be known by a different name locally to the local queue object defined remotely. In these steps, the same name is used.
6. Type `host1/echo.hub` in the Remote queue manager field.
7. Leave the Transmission queue field blank, because the transmission queue has the same name as the remote queue manager specified.
8. Click **Finish**.

Using MQSC commands

Issue the following command against host2/spoke:

```
DEFINE QREMOTE('echo') RNAME('echo') RQMNAME('host1/echo.hub') +
       DESCR('Local definition for routing requests for the echo service')
```

10.3.13 Define a reply-to queue on the spoke queue manager

Because applications can only retrieve messages from queues hosted on the queue manager to which they are connected, a reply-to queue is required on each spoke queue manager that requests the service.

This step defines a model queue from which temporary dynamic reply-to queues are generated when requesting the service. For this step, you can use the WebSphere MQ Explorer or MQSC commands.

Using the WebSphere MQ Explorer

Perform the following steps:

1. Right-click the **Queues** folder for host2/spoke, and select **New** → **Model Queue**.
2. Type `echo.replies` in the Name field.
3. Click **Finish**.

Using MQSC commands

Issue the following MQSC command against host2/spoke:

```
DEFINE QMODEL('echo.replies')
```

10.3.14 Request the echo service using a spoke queue manager

This step issues a request as an application connected to a spoke queue manager processed by the hub queue manager.

Note: This step assumes that the host1/echo.hub queue manager remains configured to provide the echo service through the echo queue it hosts. It is important that the trigger monitor is running against the echo.initq initiation queue. We advise that you ensure that the service can still be requested by applications locally connected to host1/echo.hub before proceeding. Refer to 9.4.10, “Issue a request against the service” on page 250.

1. Issue the following WebSphere MQ sample command:

   ```
   amqsreq echo host2/spoke echo.replies
   ```

   This command has the following parameters:

   – `echo` is the name used during queue name resolution on the local queue manager. The local definition of the remote queue causes this message to be routed to host1/echo.hub using the transmission queue and initiated sender/receiver channel pair.

   – `host2/spoke` is the name of the local spoke queue manager to which the application connects.

   – `echo.replies` is the name of the model queue opened in order to dynamically create a reply-to queue for the application.

Note: The **amqsreq** command does not specify that t**he host1/echo.hub** queue manager is contacted to provide the service. The infrastructure has been configured to route this request, transparent to the operation of the application.

2. Type a message and press Enter.
3. Leave a blank line and press Enter.
4. Wait for 10 seconds.
5. If successful, the output from the command contains a line of output containing your original message, as follows:

```
response <Example test message>
```

Note: If you experience problems, refer back to Refer to 9.4.10, "Issue a request against the service" on page 250 for detailed instructions.

10.4 Create a queue manager cluster

As shown in the previous section, there are a number of administration tasks required on multiple queue managers to join a queue manager to a hub and spoke infrastructure.

This section shows how clusters reduce the number and scope of administration tasked required when adding a queue manager to a WebSphere MQ infrastructure. This is because all queue managers within a cluster automatically have knowledge of, and communication links to, the other queue managers in the cluster.

The name of this cluster is example.cluster.

10.4.1 Create queue managers

Create four new queue managers with listeners created and running.

For this section, the naming of these queue managers reflects their roles within the cluster. Table 10-1 on page 292 shows the names of the queue managers to create and the ports on which to create and start listeners.

Table 10-1 Queue managers to be members of example.cluster

Name	Host name	Listener port	Repository	Channel name
host1/full	*host1.example.com*	9031	Full	clus.host1/full
host1/partial	*host1.example.com*	9032	Partial	clus.host1/partial
host2/full	*host2.example.com*	9033	Full	clus.host2/full
host2/partial	*host2.example.com*	9034	Partial	clus.host2/partial

Two of these queue managers host full repositories for the cluster, containing all the information about that cluster, and two contain partial repositories, containing only the information they require.

The example host names provided in the table reflect that this example can be usefully performed with access to two machines. However, all of these host names can be replaced with the host name or IP address of the local machine.

Note: We do not recommend using the host name `localhost` or IP address `127.0.0.1` in clusters in a real environment. However, they can be used in this example if all queue managers are hosted on the same machine.

In clusters, the channel receiver object that a queue manager publishes in a cluster is distributed to all queue managers within that cluster. Therefore, the connection name in that definition must be accessible from *all* machines that host queue managers in that cluster.

10.4.2 Assign queue managers as full repositories

This step is performed for the queue managers host1/full and host2/full.

These queue managers are configured to hold a full repository for the cluster example.cluster.

For this step, you can use the WebSphere MQ Explorer or MQSC commands.

Using the WebSphere MQ Explorer

Perform the following steps:

1. Right-click the icon for the queue manager in the navigator view and select **Properties**.
2. Select the **Repository** section in the properties window.
3. Select **Full repository for a cluster**.

4. Enter `example.cluster` in the field that becomes enabled.
5. Click **OK**.

Using MQSC commands

Issue the following MQSC command against the queue manager:

```
ALTER QMGR REPOS('example.cluster')
```

10.4.3 Create cluster receiver channel objects

This step is performed for all queue managers in the cluster.

A cluster receiver channel object defines how other queue managers within the cluster can establish communication with this queue manager.

Due to this, the connection attribute of this object defines the host name or IP address and port that other queue managers should use when connecting to this queue manager. Because a queue manager can be a member of multiple clusters and might want queue managers connecting from those clusters to use different attributes when establishing a connection, a cluster receiver channel object is configured to apply to particular cluster names.

For this step, you can use the WebSphere MQ Explorer or MQSC commands.

Using the WebSphere MQ Explorer

Perform the following steps:

1. Right-click the **Channels** folder for the queue manager, and select **New** → **Cluster-receiver Channel**.
2. Enter `clus.` followed by the name of the queue manager in the Name field. For example, for queue manager host1/full, enter:

   ```
   clus.host1/full
   ```

3. Click **Next**.
4. Enter the connection name for the queue manager being administered in the Connection name field. For example, for queue manager host1/full, enter:

   ```
   host1.example.com(9031)
   ```

   Note: Ensure that the connection name (especially the port) and the channel name are correct for the queue manager being administered.

5. Select the **Cluster** section.
6. Select **Shared in cluster**.

7. Type `example.cluster` in the field that becomes enabled.
8. Click **Finish**.

Using MQSC commands

Issue the following MQSC command against the queue manager, replacing `host1/full` with the name of the queue manager, and *`host1.example.com`*`(9031)` with the connection name for the queue manager:

```
DEFINE CHANNEL('clus.host1/full') CHLTYPE(CLUSRCVR) +
      CONNAME('host1.example.com(9031)') CLUSTER('example.cluster')
```

Note: Ensure that the connection name (especially the port) and the channel name are correct for the queue manager being administered.

10.4.4 Create cluster sender channel objects

This step is performed first for the two full repository queue managers host1/full and host2/full. It is then performed for the partial repository queue managers, host1/partial and host2/partial.

In order to join a cluster, a queue manager publishes its cluster receiver channel object definition to one full repository within the cluster. This queue manager automatically shares this information with all other full repositories within the cluster. To initially contact this full repository, the details of how to establish a connection are required. These are provided within a single, manually defined, cluster sender channel object.

The choice of full repository for the cluster sender channel objects of partial repositories is arbitrary. However, for the two full repository queue managers, the cluster sender channel object must be defined to the partner full repository.

The name of the cluster sender channel object must match the name of the cluster receiver channel object defined by the remote full repository to specify how queue managers should connect to the queue manager in the cluster.

For this step, you can use the WebSphere MQ Explorer or MQSC commands.

Using the WebSphere MQ Explorer

Perform the following steps:

1. Right-click the **Channels** folder for the queue manager, and select **New** → **Cluster-sender Channel**.

2. Enter `clus.` followed by the name of the full repository queue manager that is going to be contacted to join the cluster, in the Name field. For example, for queue manager host1/full, enter `clus.` followed by `host2/full`:

   ```
   clus.host2/full
   ```

3. Click **Next**.
4. Type the connection name for the queue manager that is going to be contacted to join the cluster in the Connection name field. For example, for queue manager host1/full, enter:

   ```
   host2.example.com(9033)
   ```

   Note: Ensure that the connection name (especially the port) is correct for the *same* full repository queue manager as the channel name.

5. Select the **Cluster** section.
6. Select **Shared in cluster**.
7. Type `example.cluster` in the field that becomes enabled.
8. Click **Finish**.

Using MQSC commands

Issue the following MQSC command against the queue manager, replacing `host2/full` with the name of the full repository queue manager being contacted to join the cluster, and `host2.example.com(9033)` with the connection name for the queue manager being contacted to join the cluster:

```
DEFINE CHANNEL('clus.host2/full') CHLTYPE(CLUSSDR) +
      CONNAME('host2.example.com(9033)') CLUSTER('example.cluster')
```

Note: Ensure that the connection name (especially the port) is correct for the *same* full repository queue manager as the channel name.

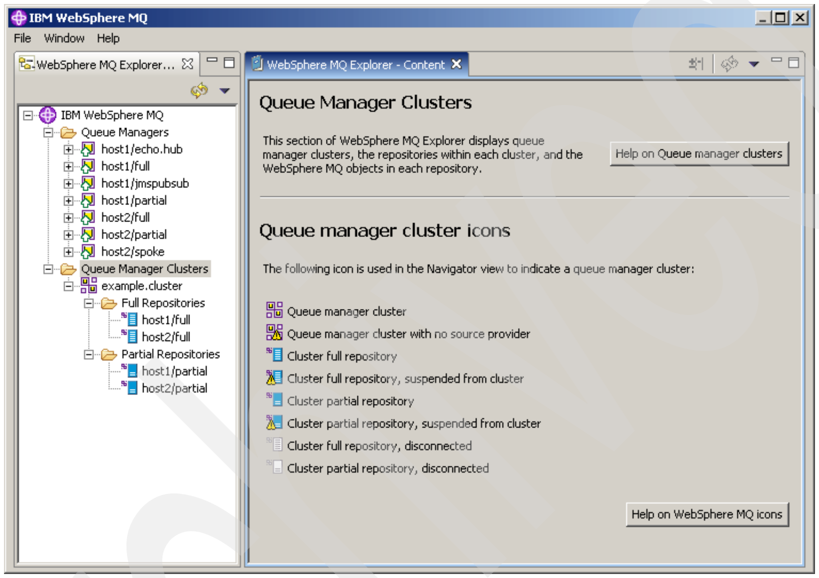
10.4.5 View information about the cluster

You have now established a cluster between the four queue managers. See 8.2, “Viewing cluster repository information” on page 194 for information about how information about this cluster can be viewed using the WebSphere MQ Explorer and MQSC commands.

As long as one of the full repository queue managers is shown in the WebSphere MQ Explorer, either because it is on the same machine or it is being remotely administered from the WebSphere MQ Explorer, the structure of the cluster is displayed under the Queue Manager Clusters folder.

Note: Initially after creation, each of the partial repositories only has knowledge of itself and the full repositories of the cluster. This is because it has not yet had reason to request information about the other partial repositories in the cluster from the full repositories.

Figure 10-4 shows the structure of the cluster, as shown in the WebSphere MQ Explorer under the Queue Manager Clusters folder.



- IBM WebSphere MQ
  - Queue Managers
    - host1/echo.hub
    - host1/full
    - host1/jmspubsub
    - host1/partial
    - host2/full
    - host2/partial
    - host2/spoke
  - Queue Manager Clusters
    - example.cluster
      - Full Repositories
        - host1/full
        - host2/full
      - Partial Repositories
        - host1/partial
        - host2/partial

Queue Manager Clusters

This section of WebSphere MQ Explorer displays queue manager clusters, the repositories within each cluster, and the WebSphere MQ objects in each repository.

Help on Queue manager clusters

Queue manager cluster icons

The following icon is used in the Navigator view to indicate a queue manager cluster:

- Queue manager cluster
- Queue manager cluster with no source provider
- Cluster full repository
- Cluster full repository, suspended from cluster
- Cluster partial repository
- Cluster partial repository, suspended from cluster
- Cluster full repository, disconnected
- Cluster partial repository, disconnected

Help on WebSphere MQ icons

Figure 10-4 The structure of example.cluster displayed in the WebSphere MQ Explorer

10.4.6 Share queues in the cluster

This step is performed for all queue managers in the cluster.

This step creates a queue, of the same cluster.queue name, on each queue manager within the cluster and shares it in the cluster.

For this step, you can use the WebSphere MQ Explorer or MQSC commands.

Using the WebSphere MQ Explorer

Perform the following steps:

1. Right-click the **Queues** folder for the queue manager under the Queue Managers folder in the navigator view, and select **New** → **Local Queue**.
2. Type `cluster.queue` in the Name field.

3. Click **Next**.
4. Select the **Cluster** section.
5. Select **Shared in cluster**.
6. Type `example.cluster` in the field that becomes enabled.
7. Click **Finish**.

Using MQSC commands

Issue the following MQSC command against the queue manager:

```
DEFINE QLOCAL('cluster.queue') CLUSTER('example.cluster')
```

10.4.7 Enable workload balancing with a local instance of a queue

This step is performed for all queue managers in the cluster.

By default, when messages are sent to a queue, and a queue of that name exists on the queue manager to which an application is connected, messages are always delivered to that local queue.

In WebSphere MQ V6.0, this behavior can be overridden to allow workload balancing to occur, including the local instance and all other instances of the queue shared in the cluster.

For this step, you can use the WebSphere MQ Explorer or MQSC commands.

Using the WebSphere MQ Explorer

Perform the following steps:

1. Right-click the icon for the queue manager under the Queue Managers folder in the navigator view, and select **Properties**.
2. Select the **Cluster** section in the properties window.
3. Change the CLWL use queue field to **Any**.
4. Click **OK**.

Using MQSC commands

Issue the following MQSC command against the queue manager:

```
ALTER QMGR CLWLUSEQ(ANY)
```

10.4.8 Workload balance messages across available queue instances

This step can be performed on any queue manager in the cluster.

Perform the following steps:

1. Put a number of messages connecting to a single queue manager in the cluster. The following example commands put 100 messages:
   - Windows:
     ```
     FOR /L %1 IN (1,1,100) DO echo Message%1 | amqsput cluster.queue host1/partial
     ```
   - UNIX (with a shell of either the Bash or Korn shell):
     ```
     i=0; while [ $i -lt 100 ]; do let i=i+1;
     echo Message$i | amqsput cluster.queue host1/partial;
     done
     ```
2. Notice the current queue depth and browse the cluster.queue queue hosted on each queue manager in the cluster. See 9.3.7, “Browse messages put to the queue” on page 232 for details about how to browse the queue.

   Note: The number of messages delivered to each queue instance shared in the cluster might not be exactly equal.

10.4.9 Share the echo service in the cluster

This step joins the host1/echo.hub queue manager to the example.cluster cluster and shares the echo service to all queue managers within that cluster. Administration is only required on this queue manager in order for it to join the cluster.

We provide an overview of the actions in this step. Refer to 10.4.3, “Create cluster receiver channel objects” on page 293 and 10.4.4, “Create cluster sender channel objects” on page 294 for further details.

Perform the following steps:

1. Create a cluster receiver channel object as follows:
   - Name: `clus.host1/echo.hub`
   - Connection name (`CONNAME`): `*host1.example.com*(9001)`
   - Shared in cluster (`CLUSTER`): `example.cluster`
2. Create a cluster sender channel object as follows:
   - Name: `clus.host1/full`
   - Connection name (`CONNAME`): `*host1.example.com*(9031)`
   - Shared in cluster (`CLUSTER`): `example.cluster`

3. Alter the echo local queue object to share it in the cluster:
   - Using the WebSphere MQ Explorer:
     i. Select the **Queues** folder for host1/echo.hub under the Queue Managers folder of the navigator view, and select **Properties**.
     ii. Select the **Cluster** section.
     iii. Select **Shared in cluster**.
     iv. Enter `example.cluster` in the field that is enabled.
     v. Click **OK**.
   - Using the following MQSC command:

     ```
     ALTER QLOCAL('echo') CLUSTER('example.cluster')
     ```

Note: Remember to enclose attributes in single quotation marks when using MQSC.

10.4.10 Share the queue providing the echo service in the cluster

This step can be performed by requesting the echo service by issuing the **`amqsreq`** WebSphere MQ sample against *any* queue manager in the cluster.

Note: This step assumes that the host1/echo.hub queue manager remains configured to provide the echo service through the echo queue it hosts. It is important that the trigger monitor is running against the echo.initq initiation queue. We advise you to ensure that the service can still be requested by applications locally connected to host1/echo.hub before proceeding. Refer to 9.4.10, “Issue a request against the service” on page 250.

Perform the following steps:

1. Define a model queue, or local queue, on the queue manager called `echo.replies` with the default attributes.
2. Issue the following command, specifying the queue manager name:

   ```
   amqsreq echo host2/partial echo.replies
   ```

3. Type a message and press Enter.
4. Leave a blank line and press Enter.
5. Wait for 10 seconds.
6. If successful, the output from the command contains a line of output containing your original message, as follows:

   ```
   response <Example test message>
   ```

Note: If you experience problems, refer back to 9.4.10, “Issue a request against the service” on page 250 for detailed instructions.

11

Securing a WebSphere MQ infrastructure

This chapter introduces the security model provided by WebSphere MQ and some of the security considerations for building a WebSphere MQ infrastructure. This includes securing access to the resources of each queue manager. It also includes securing communication between queue managers and applications over network links using the Secure Sockets Layer (SSL) or Transport Layer Security (TLS). We provide a summary of how to configure the WebSphere MQ Explorer and other Java client applications to connect to a queue manager over a secure connection.

We discuss the following topics:

- Administering a WebSphere MQ installation
- Granting access to queue manager resources
- Establishing identity context for client applications
- Secure Sockets Layer (SSL)
- WebSphere MQ internet pass-thru (IPT)

11.1 Administering a WebSphere MQ installation

In order to initially configure a WebSphere MQ installation, or create, start, and stop queue managers, a user logs directly on to the machine with the WebSphere MQ installation.

This user requires the ability to execute WebSphere MQ control commands on Microsoft Windows and UNIX, issue CL commands on IBM @server iSeries, or issue commands against queue manager subsystems on z/OS. These users are referred to as WebSphere MQ administrators in this book.

This same user, or users, maintains the WebSphere MQ installation and queue managers. For example, these users regularly check the queue manager error logs of the queue managers hosted by that machine and check for FFSTs.

Note: The `runmqsc` command is a WebSphere MQ control command. Therefore, on Windows, UNIX, and iSeries, only users that have been authorized as WebSphere MQ administrators can issue MQSC commands directly against a queue manager.

How a WebSphere MQ installation restricts access to these operations is individual to each platform as follows:

- Windows:
  Access to WebSphere MQ control commands is restricted to members of the mqm and Administrators groups.
- UNIX:
  Access to WebSphere MQ control commands is restricted to members of the mqm group.
- iSeries:
  Members of the QMQMADM group have access to all WebSphere MQ CL commands. However, more granular access can be granted to individual CL commands on iSeries. Refer to the “Authority to administer WebSphere MQ on i5/OS” section in *WebSphere MQ Security*, SC34-6588.
- z/OS:
  All authority checks performed by WebSphere MQ for z/OS are routed through an External Security Manager (ESM), which is assumed to be the z/OS Security Server Resource Access Control Facility (RACF®) in this book. The administration of profiles within RACF is required in order to grant access to queue manager data sets and commands that can be issued against queue manager subsystems. Refer to the “Authority to administer WebSphere MQ on z/OS” section in *WebSphere MQ Security*, SC34-6588.

11.2 Granting access to queue manager resources

The security model of WebSphere MQ is based on configuring access profiles for individual WebSphere MQ objects or groups of WebSphere MQ objects.

WebSphere MQ objects represent all resources owned and controlled by a queue manager, and we have discussed a number of these in this book. For example, WebSphere MQ objects configure the queues hosted by a queue manager and the channels that define the network communication to and from that queue manager.

This security model can be used to provide fine-grain security control over the entities connecting to a queue manager, allowing entities access only to the resources they require.

11.2.1 The Object Authority Manager (OAM)

On Windows, UNIX, and iSeries, each queue manager has a component called the Object Authority Manager (OAM). Whenever an entity connected to the queue manager attempts to perform an action against a WebSphere MQ object, the OAM is queried to determine if that entity has authority to perform the action.

The entity is identified by an operating system user identifier. For applications running on the same machine, this is the real user identifier that issued that application (not the effective user identifier on UNIX). We discuss considerations for establishing the identity context of entities connecting as clients from remote systems in 11.3, “Establishing identity context for client applications” on page 308.

OAM authorities

Each action that can be performed against an object has an associated authority that can be granted or revoked using the OAM. The following actions can be performed against WebSphere MQ objects by entities connected to a queue manager:

- Connecting to a queue manager:

  There is a WebSphere MQ object that represents the queue manager. In order to connect to a queue manager, an entity must have first been granted the connect authority on the queue manager object.

- Message queue interface (MQI) operations:

  Any action performed programmatically against a queue manager, regardless of the application programming interface (API) used, resolves into one or more MQI operations, for example, MQPUT, MQGET, MQINQ, and MQSET.

The MQI interface requires that an MQOPEN operation is performed before performing these operations to access the resource before using it. Most commonly, MQOPEN is used to open a queue before putting and getting messages.

It is during the MQOPEN call that the OAM is queried to determine whether the entity is authorized based on the requested operations specified in the MQOPEN call. Each operation that can be requested has a matching authority maintained by the OAM: get, put, browse, inquire, set. After performing the MQOPEN call, the application can only perform the MQI actions that were requested and authorized during the MQOPEN.

Note: Performing this authorization takes a small amount of time. Therefore, it is generally more efficient to perform multiple puts or gets after a single MQOPEN call, rather than issuing MQOPEN and MQCLOSE calls around each operation. This includes using the MQPUT1 call, which resolves to MQOPEN, MQPUT, and MQCLOSE.

The object name specified on the MQOPEN call is often the name of a queue object defined on the queue manager to which the application is connected. When opening a queue to get or browse, this is always the name of a local or alias queue object. When opening a queue to put messages, this can be the name of a local, model, alias, or remote queue object. In these cases, the entity must have been granted the requested authority on that object.

However, when putting a message to a destination on a remote queue manager, the name specified in the MQOPEN call might not be the name of an object defined on the queue manager to which the application is connected. For example, an application might specify an object queue manager name in the MQOPEN call, or the queue might reside on another queue manager within a queue manager cluster. In these cases, the application must be granted put authority on the transmission queue identified during the queue name resolution.

Table 6-1 on page 148 provides a summary of objects on which authority checks are performed when opening a queue to send a message.

Note: In order to gain authority to put to all queues shared within a queue manager cluster, an application can be granted put authority on the SYSTEM.CLUSTER.TRANSMIT.QUEUE of the queue manager to which it connects. However, more granular access can be provided by defining model queue objects specifying the names of queues shared in the cluster in the target queue (`TARGQ`), and granting put authority only to those model queue objects.

- Performing actions under a different user context:

  In some circumstances, an application might be required to perform actions as though the application had connected under a different user context, or to specify a different user identity context in the MQMD of messages put by the application.

  One example is a service that maintains the identity information from a request message in a reply message.

  If an application requires the ability to perform these operations, it must specify this when performing the MQOPEN operation on the queue. WebSphere MQ has a set of authorities that match the options available on the MQOPEN call: altusr, passid, passall, setid, setall.

  Refer to the “MQOPEN - Open object” section in *WebSphere MQ Application Programming Reference*, SC34-6596.

- Performing administrative operations on WebSphere MQ objects:

  Only users designated as WebSphere MQ administrators can perform the direct administration of queue managers using WebSphere MQ control commands and CL commands, including issuing MQSC commands using **`runmqsc`**. We discuss this in 11.1, “Administering a WebSphere MQ installation” on page 302.

  However, WebSphere MQ allows administration to be performed on queue managers by other users using the WebSphere MQ Explorer or by issuing PCF commands directly against the command server of a queue manager.

  Note: A command server is started by default for queue managers created with WebSphere MQ V6.0 on platforms other than z/OS.

  These commands are authorized based on the user identifier in the MQMD of the command message. This user identifier is based on the identity context of the application that generated the message on the queue manager to which it was connected. However, that application might have had authority to put messages with an alternate user context.

  To allow granular control of which objects can be administered by which users, the OAM provides the following set of authorities related to administration:

  – Create (`crt`) and delete (`dlt`):
    When granted against any individual object of a particular type, these authorities allow an entity to create or delete *any* object of that type.

  – Display (`dis`) and change (`chg`):
    When granted against an individual object, these authorities allow an entity to display or change the attributes of that individual object.

– Clear (`clr`), ping (`ping`), control (`ctrl`), and control-extended (`ctrlx`): When granted against an individual object of an appropriate type, these authorities allow an entity to perform particular control operations against that object. For example, clear (`clr`) authority allows an entity to issue a clear of a queue, control (`ctrl`) authority allows an entity to start and stop a channel, and control-extended (`ctrlx`) authority allows an entity to reset or resolve a channel.

Note: Some administrative actions against a queue manager can only be performed by a WebSphere MQ administrator. Prior to WebSphere MQ V6.0, this included starting, stopping, and resolving channels. Refer to the "Authority checking for PCF commands" section in *WebSphere MQ Programmable Command Formats and Administration Interface*, SC34-6598.

We discuss considerations for establishing the identity of users connecting to a queue manager over a network and issuing commands against the command server later in this chapter.

Granting, revoking, and displaying OAM authorities

A set of WebSphere MQ control commands are provided to control the authorities granted to entities against individual objects or groups of objects.

WebSphere MQ V6.0 also provides a set of PCF commands that you can use to programmatically remotely administer these authorities. For more information about these PCF commands, refer to *WebSphere MQ Programmable Command Formats and Administration Interface*, SC34-6598.

Authority can only be granted to user identifiers and group identifiers known to the local operating system.

Note: On WebSphere MQ for UNIX platforms, authority is always granted or revoked at an operating system user group level. If an authority is granted or revoked on an individual user identifier, that authority is granted or revoked on the primary group of that user identifier, not the user identifier itself.

The following WebSphere MQ control commands are used to administer OAM authorities:

- **`setmqaut`**:
  The **`setmqaut`** command is used to grant or revoke OAM authorities for WebSphere MQ objects of a particular queue manager with a particular type and a particular name. The name parameter can contain a wildcard asterisk (*) character to allow a group of names to be specified. The queue manager object itself does not require the name parameter.

  > **Note:** After making any changes using the **`setmqaut`** command for a running queue manager, it is important to issue a REFRESH SECURITY MQSC command against that queue manager. This is because queue managers hold a cache of authority information for efficiency, so might not react to changes made to authorities until this cache is cleared using a REFRESH SECURITY MQSC command. This is also important after making changes to group memberships for user identifiers in the operating system.

- **`dspmqaut`**:
  The **`dspmqaut`** command is used to display the authorities that are resolved against a particular user identifier or group identifier for a particular object.
- **`dmpmqaut`**:
  The **`dmpmqaut`** command has a similar purpose as the **`dspmqaut`** command, but provides significantly more details. This command is especially useful if attempting to identify why a particular user identifier is being granted the authorities shown by the **`dspmqaut`** command, for example, because of group memberships.

For more information about these commands, refer to the "WebSphere MQ control commands" section in *WebSphere MQ System Administration Guide*, SC34-6584.

11.2.2 Object authority in WebSphere MQ for z/OS

Authority checks on WebSphere MQ for z/OS objects are performed externally to WebSphere MQ by RACF.

We describe the circumstances in which they are performed in 11.2.1, "The Object Authority Manager (OAM)" on page 303. However, the checks are performed based on RACF profiles for individual objects, rather than OAM authorities.

These profiles can be individual to a queue manager, or shared between the queue managers within a queue sharing group. Different operations performed on WebSphere MQ objects of a queue manager require different RACF access levels.

For more information, refer to the “Setting up security” section in *WebSphere MQ for z/OS V6.0 System Setup Guide*, SC34-6583.

11.3 Establishing identity context for client applications

Establishing the identity context is simplified for applications running on the same machine as a queue manager and connecting to that queue manager using a bindings connection. In these cases, the queue manager can be sure of the operating system user identifier under which that application is running, and administration of the user identifiers on that machine can be relied on to ensure that OAM authentication is accurate.

However, starting a listener gives a queue manager an identity on a network. Applications can connect as client applications to the queue manager from remote machines through this listener.

The administration of these remote machines might not be as tightly controlled. For example, some of these machines might be desktop workstations, where users have administrator or root authority, and thus can define their own user identifiers. In some cases, the host name and port of the queue manager might be accessible over public networks, such as the Internet.

11.3.1 WebSphere MQ default behavior for establishing identity

When an application connects to a queue manager as a client over a network, the client MCA flows a user identifier over that channel for the queue manager to use to establish the identity context of the application.

This user identifier is based on the user identifier under which the application is running on the remote machine. This same user identifier must exist on the machine hosting the queue manager in order for the connection to be successful. Authority checks are then performed using that established identity context.

Note: When connecting to a remote queue manager, the user identifier flowed across the channel by the client MCA might not be identical to that on the remote machine. For example, when connecting to UNIX machines, it is based on the first 12 characters, in lowercase, of the user identifier under which the application is running. Administrator is flowed to the UNIX queue manager as admin. Therefore, a user identifier called admin must exist and be authorized on the machine hosting the queue manager in order for the connection to be successful.

Consider that there are some common user identifiers, such as mqm, and common channel names, such as SYSTEM.DEF.SVRCONN, that a remote application can use to connect to a queue manager with a running listener on an accessible network.

Many networks are secured using an intranet and firewalls, and such considerations are not considered necessary. However, WebSphere MQ provides a set of features that allow such connections to be further secured.

11.3.2 MCA user identifier

The MCA user identifier (`MCAUSER`) attribute of the server connection channels allows the default behavior previously described for client connections to be overridden. When specified, all client applications using that channel name are given the same authority on the queue manager. This is regardless of the user identifier under which they are running on the remote machine.

The value of the parameter is specified as a user identifier defined on the machine where the queue manager is hosted, noting that this might be case-sensitive. All applications that then connect to the queue manager as a client using that channel name have the same authority as though they were running under that user identifier.

This can be used to reduce administration on the remote machines, allowing the client application to run under any user identifier on the remote machine.

It can also be used to limit access to resources on a queue manager by configuring the MCA user identifier for server connection channels to provide the correct level of access. However, in some cases, full privileges on a machine are required to be made available, for example, to allow remote administration using the WebSphere MQ Explorer.

Some queue managers never receive client connections, but have a listener active for message channels. In this case, an invalid MCA user identifier attribute can be specified on server connection channel objects. This disables client applications from connecting to that queue manager.

Use of the MCA user identifier alone is not always sufficient to provide the level of security required for communication with a queue manager. In these circumstances, identity context can be assured, and privacy of communication protected, using Secure Sockets Layer (SSL) or Transport Layer Security (TLS).

11.4 Secure Sockets Layer (SSL)

Secure Sockets Layer (SSL) is an industry-standard technology for securing the flow of confidential data across public networks and providing assured identity context.

SSL builds on a number of fundamental concepts, including asymmetric and symmetric cryptography, message digests, digital signatures, digital certificates, CipherSuites, and certificate authorities. These concepts are common to all SSL implementations.

Refer to the “Introduction” section in *WebSphere MQ Security*, SC34-6588, for a full description of each of these concepts.

11.4.1 Support for SSL in WebSphere MQ

WebSphere MQ allows channels of all types to be secured using SSL. This includes distributed message channels, cluster message channels, and client channels.

SSL can be used to provide authentication of the identity of a remote application or queue manager.

SSL can also be used to assure confidentiality and data integrity for the communication link after it is established.

11.4.2 CipherSpecs

A channel is configured to require SSL authentication by specifying a single *CipherSpec* string in the SSL Cipher Specification (`SSLCIPH`) attribute of the channel object.

The name of a CipherSpec identifies the encryption algorithm and message authentication code (MAC) algorithm that is used after the channel is established.

Each CipherSpec available in WebSphere MQ correlates with a *CipherSuite* used by other SSL applications, where that CipherSuite uses the RSA key exchange protocol.

Different CipherSpecs provide different levels of security and performance. For information about the CipherSpecs available, refer to the “Working with CipherSpecs” section in *WebSphere MQ Security*, SC34-6588.

Note: Only one CipherSpec can be specified for a channel. Therefore, this must match the CipherSpec specified on the partner channel. For connections with Java client applications or WebSphere MQ internet pass-thru, the CipherSpec must be compatible with a CipherSuite specified by the Java client or WebSphere MQ IPT instance.

11.4.3 Transport Layer Security (TLS)

Transport Layer Security (TLS) Version 1.0 is a similar protocol to SSL Version 3.0, with some additions that can increase the security of the established link.

WebSphere MQ V6.0 uses the TLS protocol for certain CipherSpecs. Refer to the “Transport Layer Security (TLS) concepts” section in *WebSphere MQ Security*, SC34-6588, for more information.

Note: This this point on the discussion of SSL in this chapter equally relates to TLS.

11.4.4 Required or optional SSL client authentication

In an SSL handshake, the side that initiates the communication is called the SSL client and the side that responds to that communication is called the SSL server.

The SSL handshake allows for the circumstance where the identity of the SSL server is authenticated, but the identity of the SSL client is not. For example, when connecting an Internet browser to an Internet shopping Web site, the identity of the Web site is authenticated using its certificate, but the Web browser itself does not usually require a certificate.

For each valid channel pairing in WebSphere MQ, one MCA acts as the SSL client and one MCA acts as the SSL server. For a sender to receiver pair, the receiver is the SSL server. For a client application to server connection pair, the server connection is the SSL server.

The channel object at the SSL server side can be configured to require a valid certificate from the SSL client, or to allow the SSL client to optionally provide this certificate. This is performed using the SSL Client Authentication (`SSLCAUTH`) attribute. Regardless of the setting of this attribute, if the SSL client provides a certificate, the certificate must be authenticated correctly by the SSL server.

11.4.5 Queue manager certificate repositories

Each queue manager has its own certificate repository. This repository contains the personal certificate and private key used to identify that queue manager within the infrastructure. It also contains the public certificates of all trusted entities. These can be the public certificates of other entities that have self-signed certificates or the public certificates of trusted certificate authorities that sign the certificates in the infrastructure.

The SSL Key Repository (`SSLKEYR`) attribute of the queue manager object identifies the key repository for the queue manager. For details of how this attribute is specified on individual platforms, refer to the “ALTER QMGR” section in *WebSphere MQ Script (MQSC) Command Reference*, SC34-6597.

The queue manager identifies which certificate within this repository it must use as a personal certificate based on the certificate label, or friendly name, of the certificate within the repository. This label must be of the correct format for WebSphere MQ to be able to use the certificate. For more information, see the “Setting up SSL communications” section in *WebSphere MQ Security*, SC34-6588.

Note: The certificate label, or friendly name, is an identifier for the certificate within the repository. Do not confuse this with the distinguished name in the certificate. If you receive a personal certificate and private key in a PKCS12 file from a certificate authority, the label might not be of the correct format.

The IBM KeyMan utility can be useful in editing the label of the certificate in the PKCS12 file before importing it into a WebSphere MQ key repository. This tool is available from the following Web page:

`http://www.alphaworks.ibm.com/tech/keyman`

11.4.6 Administering certificate repositories for WebSphere MQ

For information about how to perform certificate administration tasks, refer to the “Working with WebSphere MQ SSL support” section in *WebSphere MQ Security*, SC34-6588. This includes performing the following tasks:

- Creating a key repository for a queue manager.
- Creating a self-signed certificate.
- Creating a certificate request to send to a certificate authority.

- Receiving a signed certificate from a certificate authority.
- Adding the certificate of trusted certificate authority to a repository.
- Importing a personal certificate and private key from a PKCS12 file.
- Exporting a personal certificate and private key to a PKCS12 file.
- Extracting the public part of a certificate from a repository.

For information about the full function available from the IBM GSKit iKeycmd command line tools, supplied with WebSphere MQ for Windows and UNIX platforms, refer to the “Managing keys and certificates” section in *WebSphere MQ System Administration Guide*, SC34-6584.

Note: It is important to consider the file system security of the key repository files for a queue manager. Specifically, you need to control access to .kdb and .sth files.

11.4.7 WebSphere MQ client applications

For client applications written in C or C++, the administration of a certificate repository is very similar to the administration of a queue manager certificate repository.

The certificate repositories are of the same format, and the same administration interface is used to create them. The `MQSSLKEYR` environment variable can be used in the same way as the `SSLKEYR` queue manager attribute to specify the location of the repository. Requirements are placed on the label, or friendly name, of the personal certificate in the certificate repository. Refer to the “Setting up SSL communications” and “Working with the Secure Sockets Layer (SSL) on UNIX and Windows systems” sections in *WebSphere MQ Security*, SC34-6588, for more information:

In order to configure a CipherSpec to use on the channel when connecting to the queue manager, you can use a client channel definition table (CCDT).

Alternatively, the location of the key repository and the CipherSpec can be specified explicitly. This is performed by attaching MQCD and MQSCO structures to the MQCNO structure passed into an MQCONNX call. Refer to *WebSphere MQ Application Programming Reference*, SC34-6596.

11.4.8 Java applications accessing WebSphere MQ as clients

Java applications can access a WebSphere MQ infrastructure as clients over an SSL secured connection. This includes the WebSphere MQ Explorer and applications running within a WebSphere MQ application server.

The client MCA in these cases uses a Java Secure Sockets Extension (JSSE) implementation to perform the SSL authentication with the queue manager. Refer to the “Secure Sockets Layer (SSL) support” section in *WebSphere MQ Using Java*, SC34-6591, for full information.

> **Note:** JSSE does not use the same file format for key repositories as is used by WebSphere MQ queue managers. Instead a Java KeyStore (JKS) file is used. However, the KeyMan tools provided with the WebSphere MQ product on Windows and UNIX can administer JKS key repositories. This includes both the **`gsk7cmd`** command line tool and the **`gsk7ikm`** graphical user interface (GUI). Alternatively, the keytool utility, regularly supplied with a Java Runtime Environment (JRE), can be used.

Refer to the following summary of the steps involved in configuring a Java application to connect using SSL with the default JSSE implementation:

- Specify a CipherSuite that corresponds to the CipherSpec of the server connection channel definition on the server:
  - For applications using the base Java API:
    A CipherSuite can be specified in the `sslCipherSuite` attribute of the MQEnvironment, or the `MQC.SSL_CIPHER_SUITE_PROPERTY` of the hashtable passed into the constructor of the MQQueueManager object.
    Alternatively in WebSphere MQ V6.0, a CCDT can be specified on the constructor of the MQQueueManager object. This should contain a channel with an appropriate SSL Cipher Specification (`SSLCIPH`) attribute.
  - For JMS applications:
    The `SSLCIPHERSUITE` attribute of the ConnectionFactory object can be used to specify a CipherSuite.
    Alternatively in WebSphere MQ V6.0, a CCDT can be specified in the `CCDTURL` attribute of the ConnectionFactory object. This should contain a channel with an appropriate SSL Cipher Specification (`SSLCIPH`) attribute.
- Specify the location and password for a JSSE KeyStore:
  The KeyStore should contain at least one personal certificate, including the private key. The label associated with this certificate in the KeyStore is not important for Java client applications. This certificate is password protected in the KeyStore, so a password needs to be provided for the KeyStore. The default method for specifying a JSSE KeyStore is to set the following Java properties:

```
javax.net.ssl.keyStore
javax.net.ssl.keyStorePassword
```

- Specify the location of a JSSE TrustStore:
  The TrustStore should contain the public certificates of all certificate authorities trusted by the Java application. These certificates are not usually password protected in the JKS file. Therefore, the password is optional. The TrustStore can be the same file as the KeyStore. The default method for specifying a JSSE TrustStore is to set the following Java properties:

```
javax.net.ssl.trustStore
javax.net.ssl.trustStorePassword
```

11.4.9 SSL considerations for the WebSphere MQ Explorer

The WebSphere MQ Explorer connects as a client Java application to queue managers.

You can configure the JSSE KeyStore and TrustStore in the SSL Client Certificate Stores section of the WebSphere MQ Preferences window. Figure 11-1 shows this window.

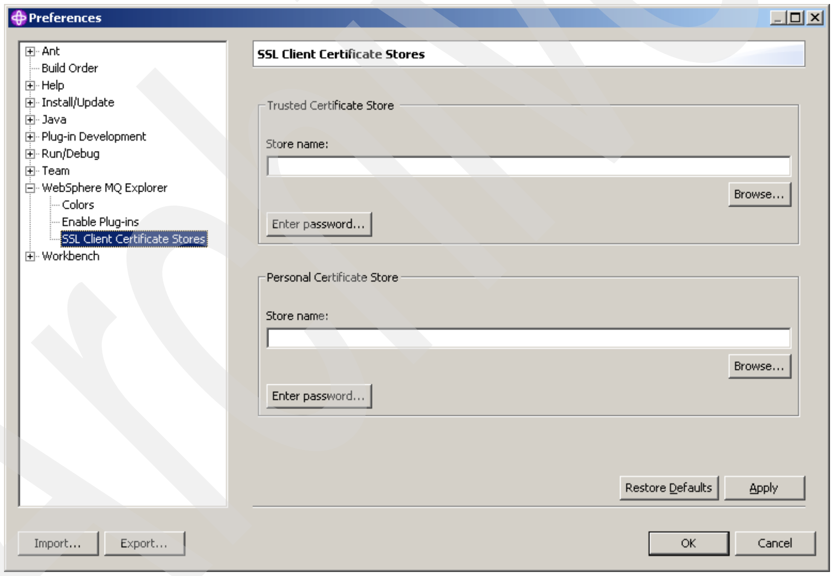


Figure 11-1 Configuring KeyStore and TrustStore in WebSphere MQ Explorer

It is not possible to manually specify the CipherSuite to use on an individual connection to a queue manager. However, the WebSphere MQ Explorer allows connections to be made to queue managers using a CCDT.

Note: Before attempting these steps, review APAR IC47466.

You can review this APAR by visiting the following WebSphere MQ support Web page and performing a search for IC47466:

http://www.ibm.com/software/integration/wmq/support/

The following steps are required to connect the WebSphere MQ Explorer to a queue manager over a SSL secured connection:

1. Create a listener and server connection channel object on the remote queue manager. Specify a value in the SSL Cipher Suite (`SSLCIPH`) attribute on the server connection channel object.
2. Create a client connection channel object of exactly the same name, with an identical SSL Cipher Suite (`SSLCIPH`) attribute. The Queue Manager Name (`QMNAME`) attribute of the client connection channel object must exactly match the name of the target queue manager where the server connection channel was created.

   It does not matter on which queue manager this object is created. However, the CCDT file needs to be transferred to the machine running the WebSphere MQ Explorer, or shared on a network drive. This file is created by a queue manager in the following location on Windows:

   ```
   C:\Program Files\IBM\WebSphere MQ\Qmgrs\QMNAME\@ipcc\AMQCLCHL.TAB
   ```
3. Place the CCDT file in a location accessible to the WebSphere MQ Explorer.
4. Create a Java KeyStore (JKS) file containing the following:
   - A personal certificate and private key to identify the WebSphere MQ Explorer. This is optional if SSL Client Authentication (`SSLCAUTH`) has been specified to be `OPTIONAL` on the server connection channel.
   - The public certificates of all certificate authorities required to authenticate the personal certificate of the queue manager.

   This step can be performed by issuing the **`strmqikm`** command on Windows, and using the IBM Key Management GUI that opens.
5. Open the WebSphere MQ Explorer.
6. Select **Window** → **Preferences** on the menu bar to open the Preferences window.
7. Select **WebSphere MQ Explorer** → **SSL Client Certificate Stores**.
8. Enter the location of the JKS file created in both the Trusted Certificate Store and Personal Certificate Store text boxes.
9. Click **Enter Password** in the Personal Certificate Store section.

10. Enter the password configured for the JKS file.
11. Click **OK**.
12. Create a direct connection to the queue manager using the Add queue manager wizard. Specify the exact name of the queue manager, and specify to use a client channel definition table. Provide the location of the CCDT file as made accessible to the WebSphere MQ Explorer previously.

> **Note:** The Add Queue Manager wizard is accessed by right-clicking the **Queue Managers** folder and selecting **Show/Hide Queue Managers** to display the Show/Hide Queue Managers window. Then, click **Add** on that window.

11.4.10 Certificate revocation lists

It is possible for the integrity of a certificate to be compromised within the validity period of that certificate. For example, the security of a private key associated with that certificate might be compromised. These certificates should no longer be trusted, even though they have been correctly signed.

SSL provides a mechanism to allow a certificate authority to revoke certificates that it has signed. For this, use certificate revocation lists (CRLs), which can be checked by WebSphere MQ queue managers using the Lightweight Directory Access Protocol (LDAP).

For more information about CRLs and how to configure WebSphere MQ queue managers to check CRLs, refer to the "Working with Certificate Revocation Lists and Authority Revocation Lists" section in *WebSphere MQ Security*, SC34-6588.

11.4.11 Choosing a certificate authority

There are three main choices of certificates to use in a WebSphere MQ infrastructure:

- Using self-signed certificates:
  Each certificate is signed by itself. The private part is kept in the certificate repository of the queue manager or client application that is identified by that certificate. The public certificate is extracted from this repository and distributed to all entities that trust that entity.

  This approach can involve a large amount of administration in maintaining the certificates in certificate stores. However, it can provide fine-grain control over the certificates and does not require a certificate authority to be set up.

- Creating a certificate authority:
  The WebSphere MQ product is supplied with tools that can be used to create and sign certificate requests. A self-signed certificate can be created in a key repository designated as a certificate authority. This can be used to sign the certificates of other queue managers and client applications. In addition, many third-party tools exists to perform these functions.

  This approach can simplify administration over using self-signed certificates and provide a more scalable solution. However, you consider the security of the certificate authority. In addition, building the infrastructure and processes for validation and signing and revoking certificates in a large infrastructure can be a significant administrative task.

- Using a third-party certificate authority:
  There are a number of third-party certificate authorities that can perform the administration of signing certificates. Certificate requests can be generated from the key repository of an entity with the correct parameters. Generating a certificate request also causes a private key to be generated, but kept securely within the key repository of that entity. The certificate request can be sent to a certificate authority to sign and return the signed public certificate. The public certificate can then be received back into the key repository of the entity to join the private certificate.

  Although there are costs involved, using a certificate authority in this way can provide significant benefits. These include that the third-party takes responsibility for the security of the certificate authority itself and might provide LDAP server access to query CRLs. This option generally requires the smallest amount of administration of certificates and certificate stores.

11.4.12 Validation of distinguished name using SSL Peer

Often, there is a requirement to provide different access privileges to different entities connecting to a queue manager over a network.

SSL validation ensures that the identity context contained in the certificate is trusted. After this trust has been established, the *distinguished name* (DN) contained in the certificate can be used to identify that entity. The DN might contain a name, e-mail address, company name, server name, country, or other relevant information.

WebSphere MQ allows individual channels to be configured to only accept connections from entities with a correctly matching DN. The DN is matched against the SSL Peer (`SSLPEER`) attribute of the channel object.

This matching is flexible to allow some fields in the DN to require a fixed string, while others contain wildcards. For a list of rules of how this matching is performed, refer to the “WebSphere MQ rules for SSLPEER values” section in *WebSphere MQ Security*, SC34-6588.

An example application of this is to configure different SSL Peer (SSLPEER) attributes on the server connection channels for administrators using the WebSphere MQ Explorer and applications accessing certain queues. Combining this with specifying different MCA user identifier (MCAUSER) attributes on these channels allows the authority provided to those groups to be controlled independently.

11.4.13 Federal Information Processing Standard (FIPS) compliance

WebSphere MQ uses IBM components, common to multiple products, to provide SSL secured communications. This means that WebSphere MQ benefits from the testing and security validation performed on these components.

On many platforms supported by WebSphere MQ, these components have passed validation programs relating to the Federal Information Processing Standard (FIPS).

For further details refer to the “Federal Information Processing Standard (FIPS)” section in *WebSphere MQ Security*, SC34-6588.

11.5 WebSphere MQ internet pass-thru (IPT)

The communication links that interconnect the nodes of a system, or provide external interfaces into that system, are often implemented using varied technologies and with different security requirements.

These links can cover large physical distances, use public networks and infrastructures such as the Internet, or be independently managed with only certain routes in and out.

Technologies such as *firewalls* can protect the external interfaces into networks, only allowing certain forms of network traffic to enter that network from particular destinations or using particular routes.

Other technologies, such as *network address translation (NAT)*, can provide different addresses for a single node when communicating within a particular network in comparison to communicating from outside of that network.

Some interfaces into networks might only allow particular transport mechanisms for data, such as the *HTTP* protocol used to transfer Web pages. These might have additional security restrictions, such as use of the secure *HTTPS* protocol built on SSL technologies.

WebSphere MQ provides the *WebSphere MQ internet pass-thru (IPT)* product in order to flexibly route WebSphere MQ communication through these varied networks.

WebSphere MQ IPT can be used to *tunnel* WebSphere MQ communications through a number of different forms of network built on the *Internet protocol suite* (TCP/IP).

Client connections, distributed message channels, and cluster message channels, including where these channels are secured by SSL, can be routed through WebSphere MQ IPT instances, transparent to the operation of WebSphere MQ.

Multiple instances of the WebSphere MQ IPT product can be connected together to bridge WebSphere MQ communications over multiple nodes in the network. At each stage, a different communications protocol, target address for the destination, or security level can be implemented by WebSphere MQ IPT. WebSphere MQ IPT nodes can communicate with other WebSphere MQ IPT nodes or a final WebSphere MQ destination.

WebSphere MQ IPT provides SSL capabilities that can connect to the SSL capabilities of WebSphere MQ by matching the CipherSuites used by WebSphere MQ IPT to the CipherSpecs used by WebSphere MQ.

A WebSphere MQ server installation is not required on machines where an WebSphere MQ IPT instance is hosted. WebSphere MQ IPT is supplied separately from the WebSphere MQ product through SupportPac MS81. Refer to the following Web page for more information about the WebSphere MQ internet pass-thru product:

http://www.ibm.com/support/docview.wss?rs=203&uid=swg24006386&loc=en_US&cs=utf-8&lang=en

12

Troubleshooting

This chapter outlines the information available for diagnosing and resolving WebSphere MQ problems. We provide specific guidance for some common issues experienced when building, administering, and accessing a WebSphere MQ environment. To aid the efficient resolution of any problems encountered that cannot be resolved using the information resources available, we provide advice about the information to provide to IBM Service.

We discuss the following topics:

- Primary information provided by WebSphere MQ
- Solving known problems
- Common problems building an infrastructure
- Common problems accessing an infrastructure
- Gathering documentation for service

12.1 Primary information provided by WebSphere MQ

Review the following sources of information when you experience any unexpected behavior from WebSphere MQ. This behavior includes:

- An unsuccessful WebSphere MQ control command
- An unsuccessful action in the WebSphere MQ Explorer
- An unsuccessful execution of a WebSphere MQ sample program
- A failure to connect to WebSphere MQ from an application
- A failure to send or receive a message from an application
- A failure in communication between queue managers
- Failures of other actions performed programmatically against WebSphere MQ

12.1.1 AMQXXXX messages

The information displayed to a user *administering* a WebSphere MQ infrastructure in response to an action or to signify an event is tagged with an AMQXXXX message identifier.

This includes success or failure messages for actions performed in the WebSphere MQ Explorer and when using MQSC and WebSphere MQ control commands. This also includes information about the operation of the queue manager as shown in the queue manager error logs.

The format of these message identifiers is `AMQ` followed by four digits. The range of numbers is from `AMQ4000` to `AMQ9999`.

Any time you see information displayed by WebSphere MQ with a message identifier in this format, you can gain more information about that message.

This includes a more detailed explanation of the circumstances in which the message is seen. It might also provide suggested actions to take in response to the message.

Use the following methods to access this additional information:

- By looking up the message identifier in *WebSphere MQ Messages*, GC34-6601
- By executing the **`mqrc`** command, specifying a message identifier, from a machine with a WebSphere MQ server installation. For example:

```
mqrc AMQ4002
```

12.1.2 Reason codes

All actions performed *programmatically* against WebSphere MQ that do not complete successfully, or only complete partially, cause a *reason code* to be returned to the application.

This reason code identifies the reason the action did not complete successfully, or only partially completed. Applications are developed to expect certain reason codes in certain situations. For example, an application waiting for messages to arrive on a queue for a number of seconds expect to receive a reason code and unsuccessful completion of the action if no messages arrive on that queue.

If an action does not complete successfully, and the application is not expecting that failure, the reason code is the first piece of information to inspect when determining the reason for the failure.

Reason codes are four digit decimal numeric values that map to a defined name for that reason code. The **`mqrc`** command provides the mapping between the decimal representation, a hexidecimal representation of the same value, and the definition name of a reason code.

The parameter of this command can be the decimal value of a reason code, `0x` followed by a hexidecimal representation of the reason code, or the defined name of the reason code. All three representations are then returned.

For more information about a particular reason code, look up the reason code by number in *WebSphere MQ Messages*, GC34-6601.

Reason codes using the MQI or object-oriented APIs

Reason codes are returned directly from actions that are performed using the core MQI application programming interface (API), or APIs based on the WebSphere MQ object model, to interact with a queue manager. *WebSphere MQ Application Programming Reference*, SC34-6596, documents the possible reason codes that can be returned by each MQI action.

Reason codes using standardized APIs such as JMS

When using standardized APIs, such as the Java Message Service (JMS), the concept of a WebSphere MQ reason code is not part of the standard for that API. However, the WebSphere MQ reason code might be contained in the information provided to that application. This information documents the failure according to the standards of the API. For example, when using the JMS API to interact with WebSphere MQ, a Java exception might contain a WebSphere MQ reason code.

12.1.3 Queue manager error logs

The queue manager error logs are the primary source of information about the *operation* of the queue manager.

If the reason code or AMQXXXX message is not sufficient to explain an unsuccessful action performed programmatically or using an administration interface, the next place to find information is in the queue manager error logs. However, these error logs do not only contain information about failures.

The queue manager error logs provide very useful information about the operation of that queue manager. We recommend the regular inspection of these logs by a WebSphere MQ administrator in the same way that the operating system logs of a machine are regularly inspected by a system administrator.

12.1.4 WebSphere MQ system error logs

Some interactions with a WebSphere MQ infrastructure cannot be linked to an individual queue manager. For example, when an application attempts to connect to a queue manager as a client, and that connection fails, the information about the failure from the perspective of the application is separate from the queue manager to which it attempted to connect.

This type of information is stored in the WebSphere MQ system error logs. Consult these logs, along with the error logs for the individual queue manager, when diagnosing problems connecting to a queue manager as a client.

These logs are available for a WebSphere MQ client installation and a WebSphere MQ server installation.

12.1.5 Error log locations

The error logs in the following locations:

- Windows:
  - Queue manager error logs:

    C:\Program Files\IBM\WebSphere MQ\Qmgrs\Queue_Manager_Name\errors
  - WebSphere MQ system error logs:

    C:\Program Files\IBM\WebSphere MQ\errors
  - WebSphere MQ system error logs for a client only installation:

    c:\Program Files\IBM\WebSphere MQ Client\errors

  – System queue manager error logs:

    C:\Program Files\IBM\WebSphere MQ\Qmgrs\@SYSTEM\errors

- UNIX:
  – Queue manager error logs:

    /var/mqm/qmgrs/*Queue_Manager_Name*/errors
  – WebSphere MQ system error logs:

    /var/mqm/errors
  – System queue manager error logs:

    /var/mqm/qmgrs/@SYSTEM/errors
- iSeries:
  – Queue manager error logs:

    /QIBM/UserData/mqm/*Queue_Manager_Name*/errors
  – WebSphere MQ system error logs:

    /QIBM/UserData/mqm/errors
  – System queue manager error logs:

    /QIBM/UserData/mqm/&SYSTEM/errors

For more information about these error logs, see 5.3.15, “Error logs” on page 119.

12.1.6 First-failure support technology (FFST)

If an unexpected event has been detected by a WebSphere MQ queue manager, which might affect the ability of that queue manager to perform its function, information is provided in a first-failure support technology (FFST) report.

Some of the information in an FFST report can be read directly by an experienced WebSphere MQ administrator. However, some of the information relates to the internal operation of WebSphere MQ at the time of the failure. This information is extremely useful for an IBM Service representative diagnosing an issue experienced with WebSphere MQ.

As such, keep any FFST files created by WebSphere MQ for a reasonable period of time. Providing a full history of any FFST files that were created by WebSphere MQ can be extremely helpful for IBM Service representatives. FFSTs created some time previous to external symptoms of a problem being observed can be instrumental in understanding the reason for a problem being experienced.

Not all FFST reports represent a failure in the WebSphere MQ product. However, they represent unexpected events. Check for FFST files regularly.

12.1.7 WebSphere MQ documentation

The WebSphere MQ documentation provides reference information about performing administration tasks and developing applications to access WebSphere MQ infrastructures.

Refer to "Related publications" on page 387 and *WebSphere MQ Bibliography and Glossary*, SC34-6603, for an overview of the contents of each publication.

The WebSphere MQ V6.0 documentation is also available in an Information Center, which provides a very useful search facility. This Information Center is supplied on the documentation media for the WebSphere MQ product. It is also available from the following Web page:

`http://publib.boulder.ibm.com/infocenter/wmqv6/v6r0/index.jsp`

12.2 Solving known problems

WebSphere MQ is a mature product, with a wide user base. If you experience an issue when building a WebSphere MQ infrastructure, or attempting to perform an administrative or programmatic action against a WebSphere MQ infrastructure, it is likely that the same issue has been experienced by other WebSphere MQ users.

Your issue may have been resolved through a change to the WebSphere MQ product in an *APAR*. The issue might have been resolved with a change to the configuration of the WebSphere MQ infrastructure, a change to an application accessing that infrastructure, or related to other hardware and software infrastructure components in the system.

12.2.1 The WebSphere MQ support Web site

Access to all IBM support information for the WebSphere MQ product is available through a central Web site:

`http://www.ibm.com/software/integration/wmq/support/`

This includes access to the WebSphere MQ documentation, a searchable interface into all problems previously fixed in the WebSphere MQ product, SupportPacs available for the WebSphere MQ product, and access to WebSphere MQ maintenance downloads.

12.2.2 Applying maintenance

IBM regularly ships packages of fixes for the WebSphere MQ product. It is important to apply these maintenance deliveries regularly to every WebSphere MQ installation in a WebSphere MQ infrastructure.

Applying maintenance deliveries regularly, regardless of whether problems have been experienced, protects your WebSphere MQ infrastructure from being affected by known problems.

Note: We recommend that all changes to a production environment, including changes to applications, changes to infrastructure configuration, and application of maintenance to infrastructure software, are performed in the QA environment prior to performing them in a production environment. This includes applying maintenance deliveries to WebSphere MQ.

Details of the maintenance deliveries available for WebSphere MQ are available on the WebSphere MQ support Web page.

Before applying a maintenance delivery, read the *readme* document associated with that maintenance delivery. This contains details of the problems fixed within that maintenance delivery, as well as a history of problems fixed in previous maintenance deliveries for that version of the WebSphere MQ product.

Updates to the information contained in the WebSphere MQ documentation and special information that might be relevant to your WebSphere MQ installation are also contained in this *readme* document.

Note: You can use the **`dspmqver`** WebSphere MQ control command on Windows and UNIX to determine the current level of maintenance applied to the WebSphere MQ installation.

This is of the form Version.Release.Modification.Fixpack.

On iSeries, you can use the CALL QMQM/DSPMQVER command.

12.2.3 Flashes

Each time a new maintenance delivery is released, or if an important interim fix is released to supplement a maintenance delivery, a flash is sent out to all users registered through the IBM support Web site.

We recommend that all WebSphere MQ administrators subscribe to these flashes. To do this, register through the IBM support Web site and add the WebSphere MQ product to your personalized page.

12.2.4 Searching APARs and Technotes

A knowledge base of solutions to common issues in documents providing advice and guidance is also provided. Each document in this knowledge base is called a Technote.

The search function provided on the WebSphere MQ support Web page enables you to search all APAR descriptions and Technotes.

12.2.5 Further sources of information

There are many sources of information on the Internet about the WebSphere MQ product. These include newsgroups, forums, and even dedicated Web sites.

The IBM developerWorks® Web site contains links to a number of such resources. See the following Web page for more information:

http://www.ibm.com/developerworks/websphere/community

12.2.6 WebSphere MQ Explorer Healthcheck plug-in

The WebSphere MQ Explorer Healthcheck plug-in extends the ability of the WebSphere MQ Explorer. It examines the configuration of all queue managers connected to the WebSphere MQ Explorer and looks for potential problems in the configuration of those queue managers.

The WebSphere MQ Explorer Healthcheck plug-in is supplied in SupportPac MH01, available from the following Web page:

http://www.ibm.com/support/docview.wss?rs=171&uid=swg24010096

12.3 Common problems building an infrastructure

This section suggests steps for diagnosing and resolving problems experienced when establishing communication between queue managers.

12.3.1 Troubleshooting distributed message channels

We recommend following these steps when diagnosing problems related to establishing communication between two queue managers using manually defined distributed message channels:

1. Ensure that the names of the channel objects defined on both sides of the connection match. Channel object names are case-sensitive.
2. Ensure that no channel status record exists on *either* queue manager for the channel name with status `STOPPED`. This disables the channel. To enable the channel, issue a start channel command against the channel object on the queue manager that has a channel status record with status `STOPPED`. To see all status records for a channel in the WebSphere MQ Explorer, right-click the channel and select **Status** → **Current Status**.
3. Ensure that the object type of the sending MCA, which retrieves messages from a transmission queue, and the object type of the receiving MCA, which delivers messages to queues on the target queue manager, represent a valid combination. We describe valid combinations in 7.4.10, "Valid distributed message channel object pairs" on page 174.
4. Ensure that the host name or IP address specified in the connection name (`CONNAME`) attribute of the channel object for the MCA that establishes the connection matches the name of the machine where the partner queue manager is hosted. Use an operating system **`ping`** command to ensure that the machine can be accessed over the network. Both sides of some channel pairs can establish a connection.
5. Ensure that the port number specified in the connection name (`CONNAME`) attribute of the channel object for the MCA that establishes the connection matches the port of the listener running for the partner queue manager. Both sides of some channel pairs can establish a connection.
6. Perform a WebSphere MQ **`ping`** across the channel.
7. If the ping is unsuccessful, check the queue manager error logs of both queue managers.
8. Ensure that the transmission queue (`XMITQ`) attribute of the channel object that defines the sending MCA matches the name of a local queue object defined on the same queue manager. Queue object names are case-sensitive.
9. Ensure that the usage (`USAGE`) attribute of the local queue object is set to transmission (`XMITQ`).
10. Attempt to manually start the channel.
11. If the channel does not start, check the queue manager error logs of both queue managers.

12. Check that a dead letter queue is defined on the queue manager hosting the receiving MCA. Persistent messages stuck on a transmission queue due to a delivery failure where no dead letter queue is defined cause the channel to enter `RETRYING` status, and eventually `STOPPED` status.
13. Check that no two channel objects on a queue manager have the same transmission queue name attribute.
14. If a status entry for the channel exists, ensure that the indoubt (`INDOUBT`) attribute of the channel status record for the channel is not `YES`.

12.3.2 Troubleshooting message channel initiation

We recommend following these steps when diagnosing problems related to initiation of distributed message channels after ensuring that the channel can be established manually:

1. Ensure the trigger control (`TRIGGER`) attribute is specified on the local queue object with usage transmission defined on the same queue manager as the sender channel object. This object is called *the transmission queue* from this point.
2. Ensure that the trigger type (`TRIGTYPE`) attribute of the transmission queue is set to `FIRST` or `DEPTH`.
3. If depth triggering is used, ensure that the trigger depth (`TRIGDPTH`) is set appropriately.
4. Ensure that the trigger data (`TRIGDATA`), or process (`PROCESS`), attribute of the transmission queue is set to the name of the sender channel object. Channel object names are case-sensitive.
5. Ensure that the initiation queue (`INITQ`) attribute of the transmission queue is set to `SYSTEM.CHANNEL.INITQ`.

Note: Custom initiation queues can be used. However, a channel initiator must be started manually against a custom initiation queue.

6. Ensure that the default channel initiator for the queue manager is running. In WebSphere MQ V6.0, use the DISPLAY QMSTATUS CHINIT MQSC command, or right-click the queue manager in the WebSphere MQ Explorer and select **Status**.
7. Check that the disconnect interval (`DISCINT`) attribute of the sender channel object is set appropriately.
8. Start the channel manually to allow all messages to flow from the transmission queue to the remote queue manager.

9. Let the channel reach its disconnect interval. Alternatively, stop the channel, specifying the target status (`STATUS`) as `INACTIVE`.
10. Put a message to a queue hosted on the remote queue manager. This should cause a message to be placed on the transmission queue, and the channel to be started automatically by the queue manager.

12.3.3 Troubleshooting cluster message channels

When joining a queue manager to a queue manager cluster, refer to 8.3.3, “Steps to join a queue manager to a cluster” on page 203.

When removing a queue manager from a queue manager cluster, refer to 8.3.4, “Steps for a queue manager to leave a cluster” on page 206.

This section provides general advice for troubleshooting problems with cluster message channels. Most problems relate to the attributes specified on the cluster sender or cluster receiver channel object definitions.

A common external symptom of a problem is entries of the following form shown in output from DISPLAY CLUSQMGR commands, or under the Clusters folder in the WebSphere MQ Explorer:

```
SYSTEM.TEMPQMGR.hostname(port)
```

These are most commonly seen due to a problem joining a cluster, or rejoining a cluster after issuing a REFRESH CLUSTER command with REPOS(YES) specified.

Note: Alterations to the manually defined cluster *sender* channel object, used to first establish contact with a full repository, do not generally have an effect after the queue manager has joined the queue manager cluster. The attributes of the manually defined cluster sender channel are overridden by an automatically defined cluster sender based on the cluster receiver channel object defined on the full repository queue manager.

One important issue in a queue manager cluster is that all queue managers must be able to establish communication with all other queue managers in the cluster using the connection name specified in the cluster receiver channel object defined on that queue manager.

If a change needs to be made to that connection name, remove that queue manager from all clusters it has joined, using that cluster receiver channel object, before rejoining with the altered connection name. We do not recommend that you change the connection name specified on a cluster receiver channel object while that object is published within any queue manager cluster.

Cluster channels are affected by many of the same considerations as distributed cluster sender channels. Always start by validating the channels between the full repository queue managers for a cluster. Then, validate the channels between a partial repository and both full repositories for the cluster. Only after establishing the health of these channels, attempt to establish the health of channels between partial repositories.

For any two queue managers in a cluster, check the following information:

- For a full repository queue manager, check that the repository (`REPOS`) or repository namelist (`REPOSNL`) attribute queue manager object contains the correct cluster name or namelist object name. Cluster names and namelist object names are case-sensitive. If a namelist is used, check the names (`NAMES`) attribute to that ensure the cluster name is contained.
- Ensure that a listener is created and running on both queue managers.
- Check that the cluster (`CLUSTER`) or cluster namelist (`CLUSNL`) attribute of the cluster sender channel object and cluster receiver channel object contain the correct cluster name or namelist object name. Cluster names and namelist object names are case-sensitive. If a namelist is used, check the names (`NAMES`) attribute to ensure that the cluster name is contained.
- Issue a DISPLAY CLUSQMGR(*) MQSC command on both queue managers. Ensure that no records exist with a `SYSTEM.TEMPQMGR.`*`hostname(port)`* name. If one does, this signifies that the queue manager has not correctly joined the cluster. This might be for two reasons:
  - The manually defined cluster sender channel is incorrect:

    This prevents a channel being established *to* the full repository queue manager. Check that the name of the cluster sender channel object matches the name of the cluster receiver channel object defined on the full repository queue manager. Check that the host name or IP address and port in the connection name is valid to contact the full repository queue manager.

    Note: It does not matter which full repository is contacted. However, the connection name *must* match the same full repository as the channel name; otherwise, the channel cannot start.

  - The cluster receiver channel object is incorrect:

    This prevents the channel being established *from* the full repository. Check the connection name specified in the cluster receiver channel object. We recommend that you set the `CLUSTER` or `CLUSNL` attribute to blank before making any changes to the connection name.

- Ensure that the host name or IP address specified in the connection name (`CONNAME`) attribute of the cluster receiver channel object for both queue managers can be accessed from the partner queue manager. You can use an operating system **`ping`** command to do this. Check that the port numbers are correct for the listeners defined.
- Check the queue manager error logs for both queue managers.
- Issue a DISPLAY CHSTATUS(*) MQSC command on both queue managers. Check for any channels which have a channel status record, but are not in `RUNNING` status, for example, records in `RETRYING` status. Also check for an `INDOUBT` attribute of `YES` on any channel status records.
- If the two queue managers are partial repositories, and both have knowledge of the full repositories for the cluster, but not of each other, this can be normal. To add knowledge of a remote partial repository, define a local queue object on that remote partial repository. Share this queue object in the cluster by specifying the cluster (`CLUSTER`) attribute to the name of the cluster. Use the **`amqsput`** sample, connected to the local partial repository, to put a message to this queue. If this completes successfully, a DISPLAY CLUSQMGR(*) command on the local partial repository shows an entry for the remote partial repository.

12.4 Common problems accessing an infrastructure

This section suggests steps for diagnosing and resolving problems experienced by applications accessing a WebSphere MQ infrastructure.

12.4.1 Troubleshooting connection failures to a queue manager

We recommend using the following steps when diagnosing problems related to an application connecting to a queue manager:

1. Ensure that the queue manager is running. For example:
   – Using the **`dspmq`** command on Windows and UNIX, or the WebSphere MQ Explorer.
   – Using the WRKMQM command on iSeries.
2. View the information about the return code from the connection action.

3. For applications connecting as clients, ensure that a listener is running for the queue manager. Ensure that the transport (usually TCP) and connection name are correct for the queue manager. If a client channel definition table (CCDT) is being used, ensure that the location specified for this file is correct. For JMS applications, these are specified on the connection factory object in the directory being accessed through JNDI, which must be accessible from the application.
4. For applications connecting as clients, ensure that the channel name being used matches a server connection channel on the queue manager, or that the channel auto-definition (`CHAD`) has been enabled on the queue manager. Channel names are case-sensitive and must match.
5. Ensure that the queue manager name specified by the application is correct, including the case. For applications connecting as clients using a CCDT, ensure that the client connection channel object, defined on the queue manager that created the CCDT, has the correct queue manager name (`QMNAME`) attribute.
6. View the WebSphere MQ system error logs.
7. View the queue manager error logs for the queue manager to which the connection is failing.
8. Ensure that the user identifier under which the application is connecting has authority to connect to the queue manager.

Note: For applications connecting as clients, it is advisable to consider specifying a message channel agent (MCA) user identifier (`MCAUSER`) attribute on the server connection channel object. This causes any application connecting using that channel name, regardless of their local user identifier, to have authority on the queue manager based on the user identifier specified in the server connection channel object.

12.4.2 Troubleshooting failures sending messages

We recommend using the following steps when diagnosing problems related to a failure to open a queue to put a message or when putting that message:

1. View the information about the return code from the open or put action.
2. Ensure that the application is specifying the correct object name, and optionally the object queue manager name, when opening the queue. Object names and object queue manager names are case-sensitive.
3. Ensure that the destination queue is known to the queue manager. For example, a valid transmission queue and local definition of the remote queue are defined.

4. If the queue is hosted by a queue manager in the same queue manager cluster, ensure that the object has been shared in the cluster by the remote queue manager. Also, ensure that both queue managers are able to establish communication with a full repository for the cluster.
5. Ensure that the application is specifying the correct options when opening the queue. Specifically, that the queue is being opened for output.
6. Review table Table 6-1 on page 148 to determine the object that is used for authority checks with the combination of object name and object queue manager name specified by the application. Ensure that the entity under which the application is connected to the queue manager has put authority for that queue. Displaying connection information for the queue manager while the application is connected might help confirm the identity context of the application, for example using the DISPLAY CONN(*) ALL command.
7. Ensure that the put message options are valid, including whether syncpoint is specified to include the put in a unit of work.
8. If syncpoint is specified, ensure that the unit of work is being committed.
9. Ensure that all queues on route to the destination are put enabled.
10. Ensure that the maximum message length (`MAXMSGL`) attributes of the locally hosted queue, and any transmission queue, are large enough for the message. Also, ensure that the maximum message length (`MAXMSGL`) attribute of the queue manager is large enough for the message.
11. View the queue manager error logs for the queue manager to which the application is connected.

12.4.3 Troubleshooting failures getting messages

We recommend using the following steps when diagnosing problems related to a failure to open a queue to get a message or when getting that message:

1. View the information about the return code from the open or get action.
2. Ensure that either a valid local queue object, or model queue object, has been defined on the queue manager to which the application is connected.
3. Ensure that the application is specifying the correct object name when opening the queue. Object names are case-sensitive.
4. Ensure that the application is specifying the correct options when opening the queue. Specifically, ensure that the queue is being opened for input, input exclusive, or browse.

5. Review Table 6-1 on page 148 to determine the object that is used for authority checks with the combination of object name and object queue manager name specified by the application. Ensure that the entity under which the application is connected to the queue manager has get authority for that queue. Displaying connection information for the queue manager while the application is connected might help confirm the identity context of the application, for example using the DISPLAY CONN(*) ALL command.
6. Ensure that the get message options are valid, including whether syncpoint is specified to include the get in a unit of work.
7. If syncpoint is specified, ensure that the unit of work is being committed.
8. Ensure that the buffer provided by the application is large enough to contain the message, unless the get message options specify that truncated messages should be accepted.

   Note: If a get fails because the buffer is not large enough, the size of the message is returned to the application. Then, the application can retry the action with a larger buffer.

9. Ensure that messages are available on the queue. This includes ensuring that other applications have not already retrieved the message from the queue and that the match options specified match the required messages.

   Note: Unless the version field in the get message options structure is set to version 2 or 3 with no match options, the message identifier and correlation identifier in the message descriptor passed to the get action should be cleared before each get.

10. Ensure that the required messages have been committed to the queue. Possible reasons for uncommitted messages on a queue are an application that did not commit a unit of work after putting a message under syncpoint, an indoubt channel to the queue manager, or another application that got the message under syncpoint, but not yet committed the unit of work.

    Note: You can check if there are uncommitted messages on a queue using the DISPLAY QSTATUS MQSC command, or by right-clicking the queue in the WebSphere MQ Explorer and selecting **Status**.

11. View the queue manager error logs for the queue manager to which the application is connected.

12.4.4 Troubleshooting common triggering problems

There are a number of triggering rules that must be satisfied in order for a trigger message to be generated on an initiation queue when a message arrives on the queue configured to be initiated. Refer to the “Starting WebSphere MQ applications using triggers” section in *WebSphere MQ Application Programming Reference*, SC34-6596, for a full list.

12.4.5 Finding a message put into the infrastructure

WebSphere MQ V6.0 queue managers provide a facility called *trace-route*. This allows routes through infrastructures of queue managers to be tested with information provided back to the trace-route application from WebSphere MQ V6.0 queue managers through which the test messages pass.

Trace-route functionality builds on functionality called *activity reporting* provided by WebSphere MQ V6.0 queue managers. This causes the components of the queue manager, such as the message channel agents (MCAs), to generate activity reports every time an action is performed on a message.

For information about trace-route and activity reporting, refer to the “Message monitoring” section in *Monitoring WebSphere MQ*, SC34-6593.

General steps for locating messages

We recommend using the following general steps when attempting to locate a message within infrastructures of interconnected queue managers:

1. Note that nonpersistent messages might be lost. Factors that affect this are restarting of queue managers, queue managers without dead letter queues defined, nonpersistent messages too large to be delivered, and network failures for channels with a fast nonpersistent message speed.
2. Ensure that all queue managers have a dead letter queue configured.
3. Browse the dead letter queue on all queue managers.
4. Check for any status other than `RUNNING` or no status (called inactive) on all channels over which the message might have been routed.
5. Browse the transmission queue on the queue manager to which the application was connected when it generated the message, and then browse subsequent transmission queues on the possible routes to the destination. This includes the SYSTEM.CLUSTER.TRANSMIT.QUEUE for queue managers within a queue manager cluster.
6. Ensure that each queue manager, transmission queue, channel, and the destination queue has a large enough maximum message length specified, especially if the message is larger than 4 MB.

7. Check for uncommitted messages on all transmission queues and the destination queue. If found, ensure that no channels are marked indoubt. Also ensure that no application has an uncommitted unit of work containing a get of that message.

How messages are routed through infrastructures

The route a message can take through an infrastructure, or interconnected infrastructures, can be complex. Queue name resolution is performed independently at each queue manager to determine the next step on route to the final destination.

For the first queue manager to which the application that sent the message is connected, this is based on the object name and object queue manager name specified when the application performed an open action in order to put messages.

For subsequent queue managers on route to the destination, the queue name resolution is performed based on queue name and queue manager name resolved by the previous queue manager. These are passed across the channel within the transmission queue header. The receiving message channel agent (MCA) performs an open action, based on this information in the transmission queue header, in the same way that an application would.

Queue name resolution, at each queue manager, is based on the queue objects residing on that queue manager, such as local queues with a usage attribute of transmission (transmission queues), queue alias objects, and queue remote objects. Queue remote objects have three separate uses in queue name resolution: to locally define a remote queue, as a queue manager alias, and as a reply-to queue alias.

The membership of a queue manager cluster also affects the queue name resolution of all queue managers that are members of that cluster. This is due to the queue managers that are members of that cluster and the queue objects they share in the cluster. Workload balancing can also occur when multiple queue objects of the same name are shared within the queue manager cluster.

12.5 Gathering documentation for service

If the particular behavior being experienced does not match the information gained by inspecting the WebSphere MQ documentation, the AMQXXXX messages returned during administration tasks, reason codes returned to an application, and the queue manager or WebSphere MQ system error logs, you might need to contact IBM Service.

This might be due to problems experienced while building a WebSphere MQ infrastructure, developing an application to access that infrastructure, or diagnosing unexpected behavior of an application in a quality assurance (QA) or production environment.

In any of these cases, providing a comprehensive set of documentation describing the issue being experienced is key to resolution of that issue by an IBM Service representative.

12.5.1 Providing a description of the observed issue

Writing a technical description, summarizing an issue observed, reduces the chance of misinterpretation of information. After contacting IBM Service, you might communicate with a number of different IBM Service representatives.

Your description needs to fully describe the external symptoms of the problem being observed. Include technical details of the symptoms, and if an individual action can be identified that is resulting in these symptoms, describe that action.

A clear technical summary of an issue can be made available to all IBM Service representatives, as well as representatives from your business, through the Problem Management Record (PMR) raised when contacting IBM Service. Raising this PMR electronically enables you to provide this description directly into the PMR, rather than through verbal communication with IBM representatives.

Visit the following Web page for information about the electronic submission of PMRs:

http://www.ibm.com/software/support/probsub.html

After submitting the record, the PMR becomes the primary source of information for the problem. IBM Service representatives update this PMR with the results of all investigations performed.

12.5.2 Environment details

Describe the environment on which the problem occurs in detail. Always include the following information:

- The hardware platform of all machines involved
- The operating system of all machines, including the maintenance level
- The WebSphere MQ product version number
- Details of all maintenance applied to the WebSphere MQ installation

12.5.3 Describing the use of WebSphere MQ

A context for the issue being observed is extremely useful. Describe how the product is being used in the circumstances in which the problem is observed. Provide as much detail as you are able. The more information is provided, the more efficient the investigation can be. Examples of such details include:

- How many machines, queue managers, and applications are involved?
- Are applications involved connecting directly to queue managers using a bindings connection or through a client connection? If through a client connection, is the hardware platform and operating system on which applications are running the same as the platform on which the queue manager is running?
- In which programming language are the applications developed?
- Are the applications using the MQI API directly using an API conforming to the WebSphere MQ object model, or using a standardized API such as JMS?
- Are the applications hosted directly by the operating system, or running within an application server, or other environment?
- What are the details of the objective of the application action, or administrative task, attempted when the issue is observed? Include technical details wherever possible, such as MQI actions, MQSC commands, or WebSphere MQ control commands.

12.5.4 Collecting failure documentation to send to IBM Service

The Technote "MustGather: Read first for all WebSphere MQ v5.3, v5.3.1, and v6.0 products" describes specific documentation to gather relevant to the issue being experienced. Refer to the following Web page for this Technote:

`http://www.ibm.com/support/docview.wss?rs=171&uid=swg21177923`

12.5.5 Re-creating the issue

The occurrence of a problem might be related to attempting to perform a specific action and thus can hence be re-created by repeating the attempt to perform that action.

If it is possible to re-create the issue being experienced by creating a simplified WebSphere MQ infrastructure including one or more queue managers, providing this information to IBM Service is useful.

However, this is not always the case. If a system is operational in a production environment and unexpected behavior occurs, re-creating the issue, or determining the original cause of that issue, can be challenging.

These circumstances can be the most important, because the operation of the system might be affected. Attempting to reproduce the problem in the production environment is not always possible.

A QA environment, which closely matches the production environment, can be extremely useful in these circumstances. You can simulate similar conditions in the QA environment and attempt to re-create the problem.

The documentation produced in the FFSTs and error logs by WebSphere MQ provides a high level of detail. This data from the original time of a failure significantly improves the ability of IBM Service representatives to identify and resolve an issue experienced in a production environment where a re-creation might not be possible.

12.5.6 WebSphere MQ trace

WebSphere MQ infrastructures, and the use of those infrastructures, can be complex and specific to the individual requirements of a business. Re-creating an issue on IBM machines is not always efficient, due to the individual nature of a business's use of the WebSphere MQ product.

Trace is a powerful feature of the WebSphere MQ, in which all internal operations of WebSphere MQ are logged to files on the file system. These files can then be sent to IBM Service for analysis.

This enables IBM Service to thoroughly investigate an issue, without requiring access to machines or making any modifications to the environment.

Trace can be started and stopped while queue managers are running on a machine.

If a problem can be re-created in a QA environment, start the trace before re-creating the problem, then re-create the problem, and then stop the trace. Send all trace information produced, along with any FFSTs, all queue manager error logs, and all WebSphere MQ system error logs for the affected machines and queue managers to IBM support for analysis.

Tracing every operation performed by WebSphere MQ can affect the performance of a WebSphere MQ infrastructure. Therefore, it is preferable to enable trace and re-create the problem in a QA environment rather than a production environment. If such a QA environment is not available, or the problem can only be reproduced in the production environment, enabling trace in the production environment might be required to resolve an issue.

If the specific action causing unexplained behavior cannot be identified, tracing the operation of WebSphere MQ during the time the external symptoms are observed aids the IBM Service investigation into the issue.

Gathering trace on Windows

On the Windows platform, start trace for all WebSphere MQ operations occurring on a system by issuing the following command:

```
strmqtrc -t detail -t all
```

Stop trace using the following command:

```
endmqtrc
```

In WebSphere MQ V6.0, the trace files are in C:\Program Files\IBM\WebSphere MQ\Trace.

In WebSphere MQ V5.3, the trace files are in C:\Program Files\IBM\WebSphere MQ\Errors.

Refer to the “Problem determination” section in *WebSphere MQ Application Programming Guide*, SC34-6595, for information about configuring options to limit the size of the trace files produced.

Gathering trace on UNIX

On UNIX platforms, start trace for all WebSphere MQ operations occurring on a system by issuing the following command:

```
strmqtrc -e -t detail -t all
```

You can start trace for one queue manager using the following command:

```
strmqtrc -m Queue_Manager_Name -t detail -t all
```

Note: If a problem occurs while starting a queue manager, or involves an application connecting to a queue manager, use the `-e` option rather than specifying an individual queue manager.

Trace files are created in the /var/mqm/trace directory.

You need to format these files to make them human readable. To do this, issue the following command from the directory containing the .TRC files:

```
dspmqtrc *.TRC
```

Refer to the “Problem determination” section in *WebSphere MQ Application Programming Guide*, SC34-6595, for information about configuring options to limit the size of the trace files produced.

Gathering trace with WebSphere MQ for AIX 5L V5.3

WebSphere MQ for AIX 5L V5.3 uses the AIX 5L operating system trace functionality. This functionality remains available on WebSphere MQ for AIX 5L V6.0, but the WebSphere MQ trace facilities described earlier are preferred.

Configure WebSphere MQ trace as follows:

```
MQS_TRACE_OPTIONS=4194303
export MQS_TRACE_OPTIONS
```

Start trace with a maximum 50 MB size non-wrapping trace file:

```
trace -a -j30D,30E -o wmq_trace.trc -s -L 52428800
```

Note: See the man page for the AIX 5L trace command for options that you can specify on the **`trace`** command.

Stop trace as follows:

```
trcstop
```

Format the single trace file produced as follows:

```
trcrpt -t /usr/mqm/lib/amqtrc.fmt wmq_trace.unf > wmq_trace.fmt
```

Gathering trace for iSeries

On the iSeries platform, trace can be started for all WebSphere MQ operations occurring on a system, by issuing the following command:

```
TRCMQM TRCEARLY(*YES) SET(*ON) TRCLEVEL(*DETAIL) MAXSTG(8)
```

Start trace for one queue manager using the following command:

```
TRCMQM SET(*ON) TRCLEVEL(*DETAIL) MAXSTG(8) MQMNAME(QMGR_NAME)
```

Stop trace using the following command:

```
TRCMQM SET(*END)
```

Trace files are stored in the /QIBM/UserData/mqm/trace/ directory on the integrated file system (IFS).

These files need to be formatted in order to be human readable. To do this, issue the following command from QShell after changing to the directory containing the .TRC files:

```
dspmqtrc *.TRC
```

Refer to the “Analyzing problems” section in *WebSphere MQ for iSeries V6.0 System Administration Guide*, SC34-6586, for information about configuring options to limit the size of the trace files produced.

z/OS

For information about WebSphere MQ trace facilities on the z/OS platform, refer to *WebSphere MQ for z/OS V6.0 Problem Determination Guide*, GC34-6600.

A

Functionality new to WebSphere MQ V6.0

This appendix provides a summary of the new functionality in Version 6.0 of IBM WebSphere MQ. This is in comparison to the previous Version 5.3 release of the WebSphere MQ product.

The WebSphere MQ Explorer

The WebSphere MQ Explorer is a graphical user interface (GUI) for monitoring and administrating a WebSphere MQ infrastructure from a desktop workstation.

The WebSphere MQ Explorer is supplied with WebSphere MQ V6.0 installations for desktop platforms, such as Microsoft Windows and Linux (x86). However, it can be used to administer queue managers hosted on machines of all platforms, including where the WebSphere MQ installation is a version previous to WebSphere MQ V6.0.

WebSphere MQ for z/OS queue managers can be administered using the WebSphere MQ Explorer, including the administration of queue sharing groups.

Note: Only WebSphere MQ for z/OS V6.0 queue managers can be remotely administered from the WebSphere MQ Explorer. Queue managers of previous versions of WebSphere MQ for z/OS cannot be administered in this way.

The WebSphere MQ Explorer can connect to queue managers for remote administration using SSL secured client connections.

For more information about the WebSphere MQ Explorer and the Eclipse technology on which it is based, see 5.2.1, “WebSphere MQ Explorer” on page 85.

PCF commands on WebSphere MQ for z/OS V6.0

The command server of WebSphere MQ for z/OS V6.0 queue managers can process programmable command format (PCF) commands, as well as the MQSC format commands previously supported by the WebSphere MQ for z/OS command server.

In addition to enabling remote administration using the WebSphere MQ Explorer, this functionality allows PCF commands to be used as a consistent, programmable administration interface into all queue managers within a WebSphere MQ infrastructure.

64-bit queue managers

The most significant functional difference between WebSphere MQ V5.3 and WebSphere MQ V6.0 on AIX 5L, Solaris™, and HP-UX is that queue managers on these platforms are now 64 bit.

This allows both 32-bit and 64-bit applications to connect to these queue managers using bindings connections. The 64-bit applications can then also connect to other 64-bit resources hosted by the machine, such as 64-bit database products.

The 64-bit queue managers also benefit from access to more memory than 32-bit queue managers on a machine with a large amount of memory resources. On such machines, a 64-bit queue manager can scale to accommodate a larger capacity than a 32-bit queue manager.

At the time of writing of this book, the WebSphere MQ V6.0 product does not provide 64-bit queue managers across all platforms. See the following Web page for details about the 64-bit platforms supported by WebSphere MQ V6.0:

`http://www.ibm.com/software/integration/websphere/mqplatforms/supported.html`

Internet Protocol Version 6 (IPv6)

In WebSphere MQ Version 6.0, queue managers can communicate using the IPv6 protocol, in addition to the existing IPv4 protocol.

For more information about the IPv6 protocol, and for information regarding the migration of existing WebSphere MQ infrastructures to IPv6, refer to *WebSphere MQ Migration Information*, SC34-6604.

Changes to SSL on Windows

WebSphere MQ for Windows V5.3 uses functionality provided by the Windows platform in order to provide Secure Sockets Layer (SSL) functionality. On other platforms, this functionality was provided using the IBM Global Security Toolkit (GSKit) component.

WebSphere MQ for Windows V6.0 uses GSKit to provide SSL functionality, inline with other platforms. This provides additional consistency across platforms and allows WebSphere MQ for Windows to benefit from additional functionality provided by GSKit.

There are migration considerations for customers using SSL and migrating queue managers or WebSphere MQ client applications from WebSphere MQ for Windows V5.3 to WebSphere MQ for Windows V6.0.

Refer to *WebSphere MQ Migration Information*, SC34-6604 for more information about the migration of WebSphere MQ for Windows V5.3 SSL certificate repositories to WebSphere MQ for Windows V6.0.

SSL and TLS improvements and FIPS certification

WebSphere MQ V6.0 can secure communication over channels using both the Secure Sockets Layer (SSL) and Transport Layer Security (TLS) protocols. TLS is used for all channels where the SSL Cipher Specification (`SSLCIPH`) attribute of a channel begins with `TLS_`.

Note: The CipherSpecs available vary by platform.

WebSphere V6.0 allows the static key, used to encrypt information that flows across a channel after an SSL handshake has been performed, to be renegotiated at regular intervals without stopping and restarting the channel. This behavior is controlled using the SSL reset key count (`SSLRKEYC`) attribute on the queue manager object.

The ability to refresh the SSL environment of a queue manager is also introduced. This is performed using the REFRESH SECURITY TYPE(SSL) MQSC command. Issuing this command causes all running SSL channels to be restarted and all cached SSL information to be discarded. Issue this command after changing any certificates in the key repository for a queue manager or to cause a queue manager to discard all cached certificate revocation lists (CRLs) previously retrieved from a Lightweight Directory Access Protocol (LDAP) server.

On the Windows and UNIX platforms, the cryptography modules used by WebSphere MQ to provide SSL and TLS functionality have passed the Federal Information Processing Standard (FIPS) Cryptomodule Validation Program of the U.S. National Institute of Standards and Technology, at level 140-2.

This FIPS compliance only applies to the cryptography modules used for certain CipherSpecs. A queue manager can be configured to only allow channels to start that use a CipherSpec that has been certified as FIPS compliant using the FIPS required (`SSLFIPS`) attribute on the queue manager object.

Refer to *WebSphere MQ Security*, SC34-6588, for more information about SSL, TLS, FIPS, CipherSpecs, and the new MQSC commands and configuration attributes.

Built-in publish/subscribe broker

The WebSphere MQ publish/subscribe broker was previously supplied separately to the WebSphere MQ product in SupportPac MA0C.

In WebSphere MQ V6.0, the publish/subscribe broker is supplied with the product and can be automatically started and stopped with a queue manager. See “Custom services started and stopped with a queue manager” on page 351.

The number of platforms supported by the publish/subscribe broker has been increased to include all platforms except WebSphere MQ for z/OS.

Note: On many platforms, the publish/subscribe broker was integrated into the product in WebSphere MQ V5.3 Fix Pack 8 (CSD8).

WebSphere MQ as a transport for Web services

WebSphere MQ V6.0 is supplied with the functionality required to allow a WebSphere MQ infrastructure to be used as a transport for Web services.

Queue sharing group enhancements on z/OS

You can use queue sharing groups in WebSphere MQ for z/OS, as described in 5.3.3, “Queue sharing groups on WebSphere MQ for z/OS” on page 100, c to increase both service and message availability.

WebSphere MQ for z/OS V6.0 provides the following enhanced queue sharing group functionality:

- Messages larger than 63 KB are supported:
  Previously, the largest message that could be placed on a shared queue within a queue sharing group was 63 KB. WebSphere MQ for z/OS V6.0 removes this limitation, and shared queues can hold messages up to the full100 MB supported by other WebSphere MQ queues.

  Note: The message body of messages larger than 63 KB is held within DB2 rather than within the coupling facility.

- Toleration of failure of the administration coupling facility structure:
  If queue sharing groups are used, and the administration coupling facility structure fails, WebSphere MQ for z/OS V6.0 queue managers within that queue sharing group do not terminate. Instead, work related to the queue sharing group is suspended and the administration structure is automatically reallocated and rebuilt. Work then continues.

Queue manager cluster workload balancing

The workload balancing algorithm, used by queue managers to route and workload balance messages to the queues shared within a queue manager cluster, has been significantly enhanced in WebSphere MQ V6.0.

This allows customization of the algorithm to account for the different capacities provided by the queue managers within a queue manager cluster, to force messages to travel along a certain route between clusters, to mark queue managers as primary or secondary locations for messages, and to include disable prioritization of local instances of a queue.

To facilitate these enhancements, WebSphere MQ V6.0 queue managers have a number of extra attributes that can be specified on the cluster receiver definition a queue manager publishes within a cluster, the queues shared within a cluster, and the queue manager object.

Refer to 8.4, “Workload balancing” on page 209 for details of the facilities provided by the WebSphere MQ V6.0 workload balancing algorithm.

Administering connections to a queue manager

WebSphere MQ V5.3 provided the ability to identify applications that had a particular queue open using the DISPLAY QSTATUS MQSC command.

WebSphere MQ V6.0 provides new functionality to identify all applications connected to a queue manager, see details of how that application has connected to the queue manager, and to see a complete list of the queues that are open for each one of those applications. It is also possible for an administrator to disconnect individual applications from the queue manager.

This functionality is available from the Application Connections window in the WebSphere MQ Explorer. It is also available using the DISPLAY CONN and STOP CONN MQSC commands.

Consistent method for starting and stopping listeners

In previous versions of WebSphere MQ, on all platforms except Windows and z/OS, it was necessary to start and stop listeners using WebSphere MQ control commands or operating system facilities.

In WebSphere MQ for Windows V5.3, listeners can be defined, started, and stopped graphically using the WebSphere MQ Services snap-in. They could also be started automatically with a queue manager.

In WebSphere MQ V6.0, on all platforms except z/OS where the behavior is unchanged, listeners can be administered as WebSphere MQ objects in the same way as any other object defined on a queue manager.

This provides the same functionality as was previously available only on Windows for all of these platforms. Because listeners are WebSphere MQ objects, the administration is consistent across all platforms and can be graphical using the WebSphere MQ Explorer or be performed using the DEFINE/START/STOP/DISPLAY LISTENER and DISPLAY LSSTATUS MQSC commands.

On Windows, listeners defined using the WebSphere MQ Services snap-in are automatically converted to WebSphere MQ objects during migration of the queue manager.

Custom services started and stopped with a queue manager

In addition to allowing listeners to be automatically started and stopped with a queue manager, with WebSphere MQ V6.0, applications specified by an administrator can be automatically started and stopped with a queue manager on all platforms except WebSphere MQ for z/OS.

This is performed by defining Service WebSphere MQ objects. These specify the details of how the application is started and stopped and at which points the queue manager should start and stop the application. It is also possible to use WebSphere MQ to check whether an application started using a Service object is still reported as running by the operating system.

These objects can be administered graphically using the WebSphere MQ Explorer, or using the DEFINE/START/STOP/DISPLAY SERVICE and DISPLAY SVSTATUS MQSC commands.

Each WebSphere MQ V6.0 queue manager has a Service object automatically defined for the WebSphere MQ publish/subscribe broker. However, this is not configured by default to start with the queue manager. To cause the publish/subscribe broker to be automatically started with a queue manager, change the service control (`CONTROL`) attribute of the `SYSTEM.BROKER` Service object to queue manager (`QMGR`).

Filtering of information about a queue manager

Previous versions of WebSphere MQ allowed wildcards to be used at the end of an certain attributes on DISPLAY MQSC commands and PCF display commands to limit the number of objects displayed.

WebSphere MQ V6.0 significantly extends this functionality to allow a more sophisticated filter to be applied in combination with the existing wildcards. This is based on an attribute name, an operator, and a value. A number of operators are available, such as equal to, less than, and greater than.

For example, you can use a filter to display queues containing more than a specified number of messages, or to display channel status records that have a particular overall status or an indoubt status.

This filtering is available from the WebSphere MQ Explorer using the Filter drop-down list shown above the majority of tables showing objects. The WebSphere MQ Explorer allows filters to be saved for future reuse. The MQSC DISPLAY commands can also use filtering by adding the `FILTER` keyword to the command, and PCF display commands can also use this filtering.

Improved real-time monitoring information

WebSphere MQ V6.0 significantly improves the real-time monitoring information available regarding the usage of a WebSphere MQ infrastructure and the performance of delivery and processing of messages that flow through that infrastructure.

Queue status records can provide information about the average time a message spends on a queue before being processed, as well the maximum age of any message on a queue. Monitoring this information for significant changes can allow early identification of problems processing messages and might help when provisioning resources for applications accessing a WebSphere MQ infrastructure.

Channel status records can provide information about the average number of messages within each batch flowing across that channel to aid in the tuning of the batch size and help you gather information about the usage of that channel. More detailed status information about a channel is also available, as well as some information about the performance of the network over which the channel is operating. The information in queue status records for transmission queues can also be used to infer information about the flow of information across channels.

Accounting information

WebSphere MQ V6.0 can generate report messages containing information about the use of a queue manager by each application that connects to that queue manager. These event messages are generated each time an application disconnects from a queue manager, or at regular intervals for longer running applications.

When enabled, these report messages are written in a PCF format to a particular queue. A sample application is provided that processes these messages and provides a textual summary. The output from this sample can be used directly, or the sample code can be used as a basis for a custom application processing the report messages.

Refer to *Monitoring WebSphere MQ*, SC34-6593, for more information about gathering accounting information.

Statistics information

Similar to accounting messages, WebSphere MQ V6.0 can generate report messages containing information about the usage of particular resources of a queue manager.

Granularity is provided to allow statistics messages to be enabled and disabled on a per-resource basis, or for all resources of a queue manager. When enabled, they are generated at regular intervals, summarizing usage within that interval.

Statistics messages fall into three categories:

- MQI statistics messages:
  These contain information about all MQI commands of each type executed by applications connected to the queue manager. All actions that involve sending or receiving messages in WebSphere MQ resolve to MQI commands, regardless of the actual application programming interface (API) used to interact with the infrastructure.
- Queue statistics messages:
  These contain information about the usage of a particular queue, for example, the number of messages put or got to a particular queue within the preceding interval.
- Channel statistics messages:
  These contain information about the usage of a channel within a particular interval, for example, the number of messages that flowed across that channel within the preceding interval.

Refer to *Monitoring WebSphere MQ*, SC34-6593, for more information about gathering statistics information.

Trace-route for WebSphere MQ infrastructures

WebSphere MQ V6.0 queue managers are able to generate activity messages every time an action is performed on a specially marked trace-route message that passes through that queue manager. Trace-route messages can be discarded automatically by a WebSphere MQ V6.0 queue manager when they reach their destination queue.

Examples of activities for which activity messages are generated are when a message is retrieved from a queue, sent across a channel, put to a destination queue, or placed on a dead letter queue.

This functionality provides the basis for tracing the full route of a message through a WebSphere MQ infrastructure by sending a trace-route message to a destination and collecting activity messages from all queue managers on route to that destination.

WebSphere MQ V6.0 provides the *display route* application to generate trace-route messages, place them into a WebSphere MQ infrastructure, collect activity reports, and summarize the contents of these activity reports in a human-readable form.

Refer to *Monitoring WebSphere MQ*, SC34-6593, for more information about activity reports, trace-route messages, and the display route application.

Logging enhancements on distributed platforms

On platforms other than z/OS, WebSphere MQ V6.0 introduces some enhancements to the logging performed by a queue manager.

An overview of this functionality is as follows:

- Increased maximum active log size:
  The maximum size of the active portion of the log has been significantly increased. This increase affects both the maximum size of log extents and the maximum number of primary and secondary log extents within the active portion. Therefore, the size of the active log of migrated queue managers, which previously had the maximum number of log extents configured, can be increased without re-creating the queue manager.
- Facilities to aid the administration of linear logs:
  For the administration of linear logs, previous versions wrote details of the oldest log extents required for queue manager restart and media recover of all queue manager objects to the queue manager error logs. WebSphere MQ V6.0 also allows information about required log extents to be queried dynamically and more granularly. This information is available graphically in the WebSphere MQ Explorer, or using the DISPLAY QMSTATUS and DISPLAY QSTATUS MQSC commands.
- Command to force advance to a new log extent:
  WebSphere MQ V6.0 allows the current log extent in use by a queue manager to be moved forward. This can be used to back up the queue manager logs, which is more consistent with the current state of a queue manager. This is performed with the RESET QMGR TYPE(ADVANCELOG) MQSC command.
- Ability to replay log records on a physically remote backup queue manager:
  WebSphere MQ V6.0 provides facilities to allow a backup queue manager to be maintained at a physically remote location by replaying backups of log extents transferred from a primary location. This can provide a solution for disaster recovery. This mechanism cannot be used to make the backup queue manager an exact copy of the primary, because the synchronous replication of data over long distances has significant performance implications. However, it can allow a queue manager, with reasonably up-to-date information, to become available quickly after a disastrous outage.

Dynamic configuration of queue managers on z/OS

WebSphere MQ for z/OS V6.0 introduces functionality to allow more queue manager configuration to be performed dynamically while a queue manager is active. An overview of this functionality is as follows:

- Many channel initiator configuration parameters can be modified while a queue manager is running using the ALTER QMGR MQSC command, instead of being set within `CSQXPARM`.

- Page sets and buffer pools can be dynamically added and removed, and buffers can be dynamically added and removed from buffer pools.
- There is an automatic preemptive expansion of page sets.
- Page sets can grow to 64 GB, allowing queues to contain larger quantities of data.
- Log data sets can be dynamically added, for example, to temporarily add log data sets if the current log fills and archiving is temporarily unavailable.

Log shunt on WebSphere MQ for z/OS

Log shunting causes the log records for some units of work to be written further down the log. This reduces the amount of log data that must be read at the queue manager restart, or backout, for long-running or long-term indoubt units of work.

For more information, refer to the “Managing the logs” section in *WebSphere MQ for z/OS V6.0 System Administration Guide*, SC34-6585.

B

Quick reference

This appendix provides a quick reference of the WebSphere MQ commands, object types, and data structures described in this book.

We discuss the following topics:

- WebSphere MQ control commands
- WebSphere MQ for iSeries CL commands
- WebSphere MQ message descriptor (MQMD) fields
- Message queue interface (MQI) verbs
- The WebSphere MQ Script (MQSC) command interface
- The queue manager object
- Listener objects
- Service objects
- Namelist objects
- Queue objects
- Cluster queue records
- Cluster queue manager records
- Channels and channel objects
- Channel status records

WebSphere MQ control commands

Use the following WebSphere MQ control commands on Microsoft Windows and UNIX to perform the administration of queue managers:

- **`dspmq`**
  Display a list of queue managers that exist on the machine and their status.
- **`crtmqm`**
  Create a queue manager.
- **`amqmdain`**
  Start or end a queue manager on Windows. Perform WebSphere MQ configuration on Windows.
- **`strmqm`**
  Start a queue manager on UNIX.
- **`endmqm`**
  End (stop) a queue manager.
- **`runmqsc`**
  Interactive console for MQSC commands.
- **`runmqsc <input.txt`**
  Execute MQSC commands contained in a script.
- **`strmqcsv`**
  Start the command server for a queue manager.
- **`setmqaut`**
  Configure Object Authority Manager (OAM) authorities for the objects of queue manager.
- **`dspmqaut`**
  Display OAM authorities for the objects of a queue manager.
- **`dmpmqaut`**
  Display detailed information regarding the OAM authority records held by a queue manager.
- **`runmqlsr`**
  WebSphere MQ V5.3 command to start a listener. For WebSphere MQ V6.0, use listener objects.
- **`mqrc`**
  Display details and numeric values for WebSphere MQ reason codes and AMQXXXX message numbers.
- **`dspmqver`**
  Report information about the current maintenance level of the WebSphere MQ installation. The version reported is in the form Version.Release.Modification.Fixpack, and changes each time a maintenance delivery is applied to the installation.

WebSphere MQ for iSeries CL commands

Use the following WebSphere MQ CL commands on iSeries to perform the administration of queue managers:

- WRKMQM
  Access all WebSphere MQ CL command panels.
- CRTMQM
  Create a queue manager.
- STRMQM
  Start a queue manager.
- ENDMQM
  End (stop) a queue manager.
- RUNMQSC
  Interactive console for MQSC commands.
- STRMQMMQSC
  Execute MQSC commands contained in a script.
- STRMQMCSVR
  Start the command server for a queue manager.
- STRMQMLSR
  WebSphere MQ V5.3 command to start a listener. For WebSphere MQ V6.0, use listener objects.
- CALL QMQM/DSPMQVER
  Report information about the current maintenance level of the WebSphere MQ installation. The version reported is in the form Version.Release.Modification.Fixpack, and changes each time a maintenance delivery is applied to the installation.

WebSphere MQ message descriptor (MQMD) fields

The WebSphere MQ message descriptor (MQMD) associated with each message that passes through a WebSphere MQ infrastructure contains the following fields:

- Message Type (`MsgType`)
  The type of the message: datagram, request, reply, or report.
- Report (`Report`)
  The circumstances in which reports are generated when delivering or processing this message.

- Feedback (`Feedback`)
  The reason a report message was generated.
- Reply-to queue (`ReplyToQ`)
  The queue name to which report or reply messages created in response to this message are sent.
- Reply-to queue manager (`ReplyToQMgr`)
  The queue manager that hosts the reply-to queue. Usually, automatically filled by WebSphere MQ.
- Message identifier (`MsgID`)
  A unique identifier for the message. Usually, automatically created by WebSphere MQ.
- Correlation identifier (`CorrelID`)
  An identifier to correlate report and reply messages with the original request or datagram message.
- Persistence (`Persistence`)
  Whether the message is persistent (business critical) or nonpersistent (query data).
- Coded Character Set Identifier (`CodedCharSetId`)
  The way character data is stored as binary data in the message.
- Encoding (`Encoding`)
  The way numeric data is stored as binary data in the message. Often messages contain only character data.
- Put time (`PutTime`)
  The time the message arrived on the queue on which it is currently stored.
- Put date (`PutDate`)
  The date the message arrived on the queue on which it is currently stored.
- Expiry (`Expiry`)
  The amount of time before the message can be discarded by the WebSphere MQ infrastructure.

Message queue interface (MQI) verbs

The following verbs make up the message queue interface (MQI), which is the core WebSphere MQ application programming interface (API):

- MQCONN
  Connect to a queue manager.
- MQCONNX
  Connect to a queue manager, specifying additional options.

- MQDISC
  Disconnect from a queue manager previously connected using MQCONN or MQCONNX.
- MQOPEN
  Open a queue, specifying which actions are going to be performed on that queue. Some other WebSphere MQ object types can be opened to inquire or set attributes.
- MQCLOSE
  Close a queue, or other WebSphere MQ object, previously opened with MQOPEN.
- MQPUT
  Put a message to a queue previously opened for output with MQOPEN.
- MQPUT1
  Wraps MQOPEN, MQPUT, and MQCLOSE into a single verb.
- MQGET
  Get, or browse, a message from a queue previously opened for input, or browse, with MQOPEN.
- MQCMIT
  Commit the current unit of work.
- MQBACK
  Back out (roll back) the current unit of work.
- MQBEGIN
  Begin a global unit of work coordinated by WebSphere MQ, including external participants such as databases.
- MQINQ
  Inquire attributes from a WebSphere MQ object previously opened for inquire with MQOPEN.
- MQSET
  Set attributes of a WebSphere MQ object previously opened for *set* with MQOPEN.

The WebSphere MQ Script (MQSC) command interface

WebSphere MQ Script (MQSC) is an administration interface that can be used to create and administer the WebSphere MQ objects within an individual queue manager. MQSC commands can be run interactively, or placed within a script.

The general format of an MQSC command is:

```
COMMAND OBJTYPE('object.name') ATTR1(VALUE) ATTR2('value') ATTR3
```

Lowercase names and values, or values that contain characters that are not alphanumeric, must be contained in single quotation marks.

For commands that display the attributes of existing WebSphere MQ objects, a *generic name* can be used. A generic name can end in an asterisk (*) character to match any objects that begin with the specified string.

We provide MQSC commands and commonly used attributes for a range of WebSphere MQ object types in this quick reference. Command keywords and object types often have short versions, which are also listed for each command.

Example MQSC commands

The following example MQSC commands demonstrate the MQSC syntax and some features of the MQSC interface:

- Define a local queue with a lowercase name:

  ```
  DEFINE QLOCAL('payroll.queue') DESCR('Payroll Queue')
  ```

- Replace an existing local queue using the `REPLACE` keyword:

  ```
  DEFINE QLOCAL('payroll.queue') DESCR('Payroll Queue NEW') REPLACE
  ```

- Create a namelist with an uppercase name and multiple values in the `NAMES` attribute, including a lowercase value:

  ```
  DEFINE NAMELIST(CLUSTER.NAMELIST) NAMES(CLUSTER1.UCASE,'cluster2.lcase')
  ```

- Display all attributes of local queues that have names starting with `payroll.` followed by any other string:

  ```
  DISPLAY QLOCAL('payroll.*') ALL
  ```

- Display only the current depth (`CURDEPTH`) and description (`DESCR`) attributes of all local queues defined on a queue manager:

  ```
  DISPLAY QLOCAL(*) CURDEPTH DESCR
  ```

- Display the current depth (`CURDEPTH`) and description (`DESCR`) attributes of queues with names starting with `payroll.` and a depth of more than 10 messages (new functionality in WebSphere MQ V6.0):

  ```
  DISPLAY QLOCAL('payroll.*') CURDEPTH DESCR WHERE(CURDEPTH GT 10)
  ```

- An MQSC command that defines a sender channel and, for readability in a script, spans multiple lines and is preceded by a comment:

  ```
  * Define a channel to queue manager QM_REMOTE
  DEFINE CHANNEL(TO.QM_REMOTE) +
  CHLTYPE(SDR) +
  CONNAME('remotehost.domain.com(1414)') +
  XMITQ(QM_REMOTE) +
  DESCR('Channel to qmgr QM_REMOTE')
  ```

The queue manager object

Each queue manager has a single queue manager object. By changing the attributes of this object, you can preform configuration of the queue manager.

You can change many of these attributes while the queue manager is running.

MQSC command for the queue manager object

The MQSC command for the queue manager is:

- ALTER QMGR or ALT QMGR
  Alter the attributes of the queue manager object.

Queue manager object attributes

The queue manager object attributes are:

- Dead letter queue (`DEADQ`)
  The name of a local queue defined on the queue manager, which is designated to receive messages that cannot be delivered.
- Queue manager identifier (`QMID`)
  Read-only attribute. Unique identifier for the queue manager to distinguish this queue manager instance from other queue managers that might have previously existed in a queue manager cluster with the same name.
- Channel auto-definition (`CHAD`)
  Whether channel auto-definition is enabled for the queue manager.
- Trigger interval (`TRIGINT`)
  The interval after which to generate an additional trigger events for queues with trigger type `FIRST`.
- Maximum message length (`MAXMSGL`)
  The maximum length of any message that can be stored on any queue hosted by this queue manager.
- Start command server (`SCMDSERV`)
  Whether to automatically start the command server with the queue manager.
- Start channel initiator (`SCHINIT`)
  Whether to automatically start the channel initiator with the queue manager on Windows, UNIX, and iSeries.
- SSL key repository (`SSLKEYR`)
  The location of the SSL key repository for the queue manager.
- Repository (`REPOS`)The name of a single cluster for which this queue manager should hold a full repository.

- Repository namelist (`REPOSNL`)
  The name of a namelist object, contains the names of multiple clusters for which this queue manager should hold a full repository.
- Cluster workload use queue (`CLWLUSEQ`)
  Whether to by default perform workload balancing across remote cluster instances of a queue when a local queue with the same name exists on the queue manager.
- Cluster workload recently used channel (`CWMRUC`)
  The maximum number of instances of a queue in a cluster across which to workload balance.

Listener objects

Listeners give a queue manager an identity within a network. Listener objects define the attributes of a listener that can be started for a queue manager.

MQSC commands for listener objects

The MQSC commands for listener objects are:

- DEFINE LISTENER(*NAME*) TRPTYPE(*TCP*) or DEF LSTR(*NAME*) TRPTYPE(*TCP*)
  Create a new listener object.
- ALTER LISTENER(*NAME*) TRPTYPE(*TCP*) or ALT LSTR(*NAME*) TRPTYPE(*TCP*)
  Alter an existing listener object.
- DELETE LISTENER(*NAME*) or DELETE LSTR(*NAME*)
  Delete an existing listener object.
- DISPLAY LISTENER(*GENERIC_NAME*) or DIS LSTR(*GENERIC_NAME*)
  Display the attributes of existing listener objects.
- DISPLAY LSSTATUS(*GENERIC_NAME*) or DIS LSSTATUS(*GENERIC_NAME*)
  Display the status of running listeners.
- START LISTENER(*NAME*) or STA LSTR(*NAME*)
  Start the listener associated with a listener object.
- STOP LISTENER(*NAME*) or STOP LSTR(*NAME*)
  Stop the listener associated with a listener object.

Attributes for listener objects

The attributes for listener objects are:

- Port (`PORT`)
  The port on which a TCP/IP listener listens for connections.
- Control (`CONTROL`)
  Whether the listener is started manually (`MANUAL`), or automatically when the queue manager starts (`QMGR`).

Service objects

Service objects allow custom applications to be started and stopped by a queue manager. This includes automatically starting and stopping an application when a queue manager is started and ended.

The WebSphere MQ publish/subscribe broker is an application that can be started using a service object. A service object called `SYSTEM.BROKER` is created with a WebSphere MQ V6.0 queue manager for the publish/subscribe broker.

MQSC commands for service objects

The MQSC commands for service objects are:

- DEFINE SERVICE(*NAME*) or DEF SERVICE(*NAME*)
  Create a new service object.
- ALTER SERVICE(*NAME*) or ALT SERVICE(*NAME*)
  Alter an existing service object.
- DELETE SERVICE(*NAME*)
  Delete an existing service object.
- DISPLAY SERVICE(*GENERIC_NAME*) or DIS SERVICE(*GENERIC_NAME*)
  Display the attributes of existing service objects.
- DISPLAY SVSTATUS(*GENERIC_NAME*) or
  DIS SVSTATUS(*GENERIC_NAME*)
  Display the status of running services.
- START SERVICE(*NAME*) or STA SERVICE(*NAME*)
  Start the application associated with a service object.
- STOP SERVICE(*NAME*)
  Stop the application associated with a service object.

Attributes for service objects

The attributes are service objects are:

- Service type (SERVTYPE)
  Whether only a single instance of the service can be started at a time (SERVER), or whether multiple instances can be started (COMMAND).
- Control (CONTROL)
  Whether the service is started manually (MANUAL), started and stopped with the queue manager (QMGR), or only started with the queue manager (STARTONLY).
- Start command (STARTCMD)
  The path of the executable used to start the service.
- Start arguments (STARTARG)
  Arguments to pass to the command on startup.
- Stop command (STOPCMD)
  The path of the executable used to stop the service.
- Stop arguments (STOPARG)
  Arguments to pass to the command when stopped.
- Standard out (STDOUT)
  A file name to which the *standard output* from the service is redirected while it is running.
- Standard error (STDERR)
  A file name to which the *standard error* from the service is redirected while it is running.

Namelist objects

Namelist objects contain a set of names. Namelists objects are most commonly used to contain a list of clusters, which can then be specified in the cluster namelist (CLUSNL) or repository namelist (REPOSNL) attributes of other objects.

MQSC commands for namelist objects

The MQSC commands for namelist objects are:

- DEFINE NAMELIST(*NAME*) or DEF NL(*NAME*)
  Create a new namelist object.
- ALTER NAMELIST(*NAME*) or ALT NL(*NAME*)
  Alter an existing namelist object.

- DELETE NAMELIST(*NAME*) or DELETE NL(*NAME*)
  Delete an existing namelist object.
- DISPLAY NAMELIST(*GENERIC_NAME*) or DIS NL(*GENERIC_NAME*)
  Display the attributes of existing namelist objects.

Attribute for namelist objects

The attribute for namelist objects is:

- Names (`NAMES`)
  A list of names. Specify in MQSC by separating each name with a comma.

Queue objects

Queue objects define the queues hosted by a queue manager and control how a queue manager routes message to queues hosted by other queue managers within a WebSphere MQ infrastructure.

Types of queue objects

The types of queue objects are:

- Local queue (`QLOCAL` or `QL`)
  Local queues are the only type of queue object that represent a queue within a queue manager that can hold messages.
  A local queues can be designated as *transmission queues* to provide a temporary location for message designated for another queue manager within the WebSphere MQ infrastructure. Messages sent to a remote queue manager with the same name as a transmission queue are placed on that transmission queue.
- Alias queue object (`QALIAS` or `QA`)
  Alias queue objects provide a reference to another queue object with a different name. The target queue object can be a local queue, a remote queue object, or a queue shared within the queue manager cluster.
- Model queue object (`QMODEL` or `QM`)
  Model queue objects provide a mechanism for local queues to be dynamically created by applications. This can be used to give an application a temporary identity within the WebSphere MQ infrastructure. The attributes of the model queue object determine the attributes of the *dynamic queue* created.

- Remote queue object (`QREMOTE` or `QR`)
  Remote queue objects are used to explicitly define and control routes between queue managers. Remote queue objects have the following uses, depending on the attributes specified:
  - Queue manager alias
    A queue manager alias defines a route to a queue manager, where the name of the remote queue manager does not match the name of the transmission queue. Queue manager aliases can also provide a reference to a queue manager with a different name.
  - Local definition of a remote queue
    A local definition of a remote queue defines an explicit route through a transmission queue to a queue of a particular name hosted on a remote queue manager.
  - Reply-to queue alias
    A reply-to queue alias causes a queue manager to replace reply-to information specified within the MQMD of a message with the information contained in the reply-to queue alias at the time a message is sent.

MQSC commands for queue objects

The following commands are common to all queue objects. Replace `QLOCAL` or `QL` with the required queue object type listed earlier.

- DEFINE QLOCAL(*NAME*) or DEF QL(*NAME*)
  Create a new queue object.
- ALTER QLOCAL(*NAME*) or ALT QL(*NAME*)
  Alter an existing queue object.
- DELETE QLOCAL(*NAME*) or DELETE QL(*NAME*)
  Delete an existing queue object.
- DISPLAY QLOCAL(*GENERIC_NAME*) or DIS QL(*GENERIC_NAME*)
  Display the attributes of existing queue objects.

The following command can be used to display the attributes of any locally defined queue object, regardless of its type:

```
DISPLAY QUEUE(GENERIC_NAME) or DIS Q(GENERIC_NAME)
```

The following MQSC command displays status records associated with local queues. This command applies only to local queues.

```
DISPLAY QSTATUS(GENERIC_NAME) or DIS QS(GENERIC_NAME)
```

Attributes for all queue objects

The following attributes apply to all queue object types:

- Put (`PUT`)
  Whether it is possible to put messages onto this local queue, or send messages through this queue object. Put disabling a queue shared in a cluster prevents messages from being routed to it.
- Default persistence (`DEFPSIST`)
  The default persistence of messages sent (with `MQPUT`), after opening this queue object (with `MQOPEN`).

Queue attributes for workload balancing within clusters

The following attributes apply to all queue object types, with the exception of model queue objects. Dynamic queues, created from model queue objects, cannot be shared within a cluster.

> **Note:** Sharing a queue object in a cluster allows the queue managers in that cluster to include this queue object as a destination when workload balancing messages addressed to the name of the queue object.

- Cluster (`CLUSTER`)
  The name of a single cluster in which the queue object is shared.
- Cluster namelist (`CLUSNL`)
  The name of a namelist object, containing multiple cluster names in which the queue object is shared.
- Default bind type (`DEFBIND`)
  The default bind type when opening this queue object. If *bind on open* (`OPEN`) is specified, multiple messages addressed to the name of this queue object sent using the same object handle (a single `MQOPEN`) are delivered to the same queue manager within a cluster. Otherwise, if *bind not fixed* (`NOTFIXED`) is specified, workload balancing occurs independently for each message sent.
- Cluster workload rank (`CLWLRANK`)
  A queue instance shared in a cluster with a higher rank is chosen in preference to queues with a lower rank, even if the queue manager hosting it is unavailable.
- Cluster workload priority (`CLWLPRTY`)
  A queue instance shared in a cluster with a higher priority is chosen in preference to queues with a lower priority, as long as the queue manager hosting it is available.

- Cluster workload use queue (CLWLUSEQ)
  Whether to perform workload balancing across remote cluster instances of a queue when a local queue with the same name exists on the queue manager. This attribute applies only to manually created local queues.

Attributes for local queues, including dynamic queues

The following attributes apply to local queues (QLOCAL). These attributes can be specified on a model queue object (QMODEL) to control the attributes of the dynamic queue created.

- Definition type (DEFTYPE)
  Whether a queue was created manually (PREDEFINED) or dynamically from a model queue object. Dynamic queues can be permanent (PERMDYN) or temporary (TEMPDYN). Temporary dynamic queues cannot hold persistent messages, because they are automatically deleted when an application disconnects or a queue manager ends.
- Get (GET)
  Whether it is possible to open this local queue to get or browse messages. This attribute also applies to alias queue objects that alias local queues.
- Usage (USAGE)
  A usage of transmission (XMITQ) marks a local queue as a transmission queue, which provides a route to a single remote queue manager. In order for messages to flow to the remote queue manager, a channel is required to process messages from the transmission queue.

Triggering is a mechanism that can be used to start applications when messages arrive on a queue. On Windows and UNIX, triggering can be enabled on a transmission queue to automatically start channels when messages are available to send. This is called *channel initiation*. The following attributes configure triggering on a queue:

- Trigger control (TRIGGER)
  Whether triggering is enabled for this queue. Triggering should be enabled for channel initiation.
- Trigger type (TRIGTYPE)
  The type of triggering enabled for this queue. For channel initiation, this should be FIRST.
- Trigger depth (TRIGDPTH)
  The threshold on which to generate a trigger event when the trigger type is DEPTH.
- Trigger data (TRIGDATA)
  Custom data to add to the trigger message. For channel initiation, this is set to the name of the channel to initiate.

- Initiation queue (INITQ)
  The queue on which to generate trigger messages when a trigger event occurs. For channel initiation, this is SYSTEM.CHANNEL.INITQ.

Attribute for alias queue objects

The following attribute applies to alias queue objects (QALIAS):

- Target queue (TARGQ) or base queue
  The name of the queue being aliased.

Attributes for local definitions of remote queues

The following attributes of a remote queue object (QREMOTE) are used when defining a local definition of a remote queue:

- Remote name (RNAME)
  The name of the queue on the remote queue manager.
- Remote queue manager name (RQMNAME)
  The name of the remote queue manager.
- Transmission queue (XMITQ)
  The name of the transmission queue where messages are placed to be transferred to the remote queue manager. This can be blank if the same as the remote queue manager name.

Attributes for queue manager aliases

The following attributes of a remote queue object (QREMOTE) are used when defining a queue manager alias:

- Remote name (RNAME)
  Blank for queue manager aliases.
- Remote queue manager name (RQMNAME)
  The name of the queue manager being aliased.
- Transmission queue (XMITQ)
  The name of the transmission queue where messages are placed to be transferred to the remote queue manager. This can be blank if the same as the remote queue manager name, or if the local queue manager is being aliased.

Attributes for reply-to queue aliases

The following attributes of a remote queue object (`QREMOTE`) are used when defining a reply-to queue alias:

- Remote name (`RNAME`)
  The name to place in the reply-to queue field.
- Remote queue manager name (`RQMNAME`)
  The name to place in the reply-to queue manager field.
- Transmission queue (XMITQ)
  Blank for reply-to queue aliases.

Cluster queue records

A cluster queue record represents a queue object that has been shared by a queue manager within a cluster. Each cluster queue record is a destination that can be sent messages by the workload balancing algorithm.

If a queue manager holds a full repository for a cluster, it has a cluster queue record for every queue shared in that cluster. If a queue manager holds a partial repository for a cluster, it only has cluster queue records for queue names of which it shares an instance, or queue applications connect to the queue manager have accessed.

MQSC command for cluster queue records

The MQSC command for cluster queue records is:

- DISPLAY QCLUSTER(*GENERIC_NAME*) or DIS QC(*GENERIC_NAME*)
  Display the attributes of cluster queue records known to this queue manager.

Attributes of cluster queue records

The attributes of cluster queue records are:

- Cluster (`CLUSTER`)
  The cluster in which the queue object is shared.
- Cluster queue manager (`CLUSQMGR`)
  The name of queue manager within the cluster that has shared this object.
- Queue manager identifier (`QMID`)
  A unique identifier for the queue manager within the cluster that has shared this object.

- Cluster queue type (`CLUSQT`)
  The type of queue object shared in the cluster.

Cluster queue manager records

A cluster queue manager record represents a queue manager and how it is contacted through a cluster of which it is a member.

If a queue manager holds a full repository for a cluster, it has a cluster queue manager record for every queue manager that is a member of that cluster. If a queue manager holds a partial repository for a cluster, it only has cluster queue manager records for queue managers that are full repositories for the cluster, or host queue objects known by the local queue manager.

MQSC command for cluster queue records

The MQSC command for cluster queue records is:

- DISPLAY CLUSQMGR(*GENERIC_NAME*) or
  DIS CLUSQMGR(*GENERIC_NAME*)
  Display the attributes of cluster queue manager records known to this queue manager.

Attributes of cluster queue manager records

The attributes of cluster queue manager records are:

- Cluster (`CLUSTER`)
  The cluster of which the queue manager is a member.
- Cluster queue manager (`CLUSQMGR`)
  The name of queue manager.
- Queue manager identifier (`QMID`)
  A unique identifier for the queue manager.
- Queue manager type (`QMTYPE`)
  Whether this queue manager holds a full or partial repository for the cluster.
- Channel (`CHANNEL`)
  The name of the channel used to establish communication with this queue manager.
- Definition type (`DEFTYPE`)
  The type of channel used to establish communication with this queue manager.

- Connection name (CONNAME)
  The host name or IP address and port number used to establish communication with this queue manager.
- Status (STATUS)
  The current status of the channel used to communicate with this queue manager.

Channels and channel objects

Channel objects configure the message channel agents (MCAs) that establish and receive connections to and from a queue manager over a network.

MCAs always work in pairs. The names of the two MCAs, thus the name of the channel objects, must match. An established connection between two MCAs is called a channel.

Applications connecting as clients to a queue manager often programatically specify the attributes of their MCA, instead of using channel objects.

Channel objects are used to join a queue manager to a queue manager cluster.

Types of channel objects

The following types of channel objects can be defined on a queue manager:

- Distributed message channel types:
  - Sender (SDR)
    Sends all messages that arrive on a specified transmission queue. The partner can be a receiver or a requester. Always establishes the connection, although it can receive requests to establish a connection from a requester.
  - Receiver (RCVR)
    Receives messages and routes them to queues. The partner should be a sender. Never establishes the connection.
  - Server (SVR)
    Send all messages that arrive on a specified transmission queue. The partner should be a requester. Can receive connections. Can also establish connections if a connection name has been specified in the definition.
  - Requester (RQSTR)
    Receives messages and routes them to queues. The partner can be a sender or server. Can receive, establish, or request connections.

- Cluster message channel types:
  - Cluster sender (`CLUSSDR`)
    Sends messages from the cluster transmission queue to other queue managers within a cluster. When explicitly defined, these establish an initial connection to a full repository queue manager in order to begin joining the specified clusters. Cluster sender channels are automatically established by WebSphere MQ between queue managers in a cluster. The attributes of automatically defined cluster sender channels are based on the cluster receiver channel objects published by the queue managers in a cluster.
  - Cluster receiver (`CLUSRCVR`)
    Defines how all queue managers within the specified clusters should connect to this queue manager. The partner can be a manually defined cluster sender, or an automatically defined cluster sender.
- Message queue interface (MQI) channel types:
  - Server connection (`SVRCONN`)
    Defines how a client application can connect to a queue manager.
  - Client connection (`CLNTCONN`)
    This is different to all other channel types, because it is never used by the queue manager itself. Instead, an entry is added to a client channel definition table (CCDT) file, which can be distributed to other machines and used by client applications to configure their MCAs.

MQSC commands for channel objects

The MQSC commands for channel objects are:

- DEFINE CHANNEL(*NAME*) CHLTYPE(*TYPE*) or DEF CHL(*NAME*) CHLTYPE(*TYPE*)
  Create a new channel object.
- ALTER CHANNEL(*NAME*) CHLTYPE(*TYPE*) or ALT CHL(*NAME*) CHLTYPE(*TYPE*)
  Alter an existing channel object.
- DELETE CHANNEL(*NAME*) or DELETE CHL(*NAME*)
  Delete an existing channel object.
- DISPLAY CHANNEL(*GENERIC_NAME*) or DIS CHL(*GENERIC_NAME*)
  Display the attributes of existing channel objects.

Attributes of channel objects

The attributes of channel object are:

- Connection name (CONNAME)
  For channels that can be used to establish a connection, this is the network connection information for the target queue manager. For cluster receiver (CLUSRCVR) channels, this is connection information for the local queue manager. For TCP/IP channels, the format is hostnameoripaddress(port).
- Transmission queue (XMITQ)
  For sender (SDR) and server (SVR) channels, this is the name of the transmission queue from which they get messages and send them over the channel.
- Short retry (SHORTRTY)
  For message channels that establish connections, this is the number of times to attempt to establish a connection at intervals specified by the short retry timer (SHORTTMR).
- Short retry timer (SHORTTMR)
  The number of milliseconds to wait between short retry (SHORTRTY) attempts. While waiting between retry attempts, the channel is in RETRYING status.
- Long retry (LONGRTY)
  If the number of short retry attempts has been reached on a message channel attempting to establish a connection, the channel continues to attempt to establish a connection at intervals specified by the long retry timer (LONGTMR). If the number of long retry attempts is reached, the channel enters STOPPED status and must be manually restarted.
- Long retry timer (LONGTMR)
  The number of milliseconds to wait between long retry (LONGRTY) attempts. While waiting between retry attempts, the channel is in RETRYING status.
- Message retry (MRRTY)
  For message channels that receive messages, this is the number of times to attempt to deliver the message to the destination queue before placing the message on the dead letter queue.
- Message retry timer (MRTMR)
  The number of milliseconds to wait between message retry attempts.
- MCA user identifier (MCAUSER)
  For server connection (SVRCONN) channels, this forces the actions of clients connecting using that channel to be validated using the specified user identifier.
- Nonpersistent message speed (NPMSPEED)
  For message channels, this specifies whether to use units of work when transferring nonpersistent messages.

- Batch size (`BATCHSZ`)
  For message channels, this is the maximum number of message to transfer before confirming delivery and committing the unit of work.
- Batch interval (`BATCHINT`)
  For message channels, this is the maximum amount of time to wait for a batch to fill before confirming delivery and committing the unit of work.
- Disconnect interval (`DISCINT`)
  For message channels that send messages, this is the amount to time to leave a channel active while no messages are available on the transmission queue.
- SSL cipher specification (`SSLCIPH`)
  The Secure Sockets Layer (SSL) or Transport Layer Security (TLS) cipher specification to use to secure the channel. This must match on both sides of the channel.
- SSL client authentication (`SSLCAUTH`)
  For channels that act as an SSL server receiving connections, this determines whether the SSL client is required to provide a certificate.
- SSL peer (`SSLPEER`)
  A string specifying the distinguished names (DN) that are allowed to connect to this channel. This is checked after performing the authentication of the partner's certificate.
- Cluster (`CLUSTER`)
  For cluster message channels, this is the name of a single cluster to which the channel object applies.
- Cluster namelist (`CLUSNL`)
  For cluster channels, this is the name of a namelist object, containing multiple cluster names to which the channel object applies.

MQSC commands for controlling channels

The following MQSC commands are used to manually establish connections to remote queue managers. These commands also control whether channels can be started automatically by WebSphere MQ, in response to connections from remote queue managers and applications, or by a channel initiator.

- START CHANNEL(*NAME*) or STA CHL(*NAME*)
  Establish a connection to a remote queue manager, using the specified channel object.
  This command can also be used to enable channels previously disabled with a STOP CHANNEL command.
- STOP CHANNEL(*NAME*) or STOP CHL(*NAME*)
  Stop all channels currently established with the specified name.

By default STATUS(STOPPED) is specified on the command. This disables any channels of that name from starting automatically. This behavior can be overridden using STATUS(INACTIVE) on the command.

Channel status records

A channel status record is held for each message channel agent (MCA) that is active for a queue manager. A channel status record is also created whenever a channel is disabled by issuing a STOP CHANNEL command or because a channel reaches its number of long retry (`LONGRTY`) attempts.

If a channel is in `INACTIVE` status, no channel status record exists.

Some of the attributes specified on channel objects are negotiated between the two MCAs that form the channel. A channel status record shows the negotiated value for these attributes.

MQSC command for channel status records

The MQSC command for channel status records is:

- DISPLAY CHSTATUS(*GENERIC_NAME*) or DIS CHS(*GENERIC_NAME*)
  Display the attributes of channel status records.

Attributes of channel status records

The attributes of channel status records are:

- Status (`STATUS`)
  The overall status that this channel status record represents: `RUNNING` for active channels, `RETRYING` for channels that have failed to connect and are retrying, or `STOPPED` for channels that are disabled. Other values, such as `BINDING` or `STOPPING`, are transitory.
- In doubt status (`INDOUBT`)
  Whether a message channel that sends messages is currently indoubt. A channel is indoubt while waiting for acknowledgment that a batch of messages has been received and can remain indoubt if the connections breaks during that time. Indoubt status is automatically resolved by restarting a channel.
- Remote queue manager name (`RQMNAME`)
  For message channels, this is the name of the remote queue manager.

Glossary

Application server. A managed environment within which applications are deployed and run with access to a defined set of functionality that might include a messaging facilities, such as WebSphere MQ.

Broker. In a publish/subscribe messaging model, a broker maintains information about topics and the subscribers on those topics. When a publisher publishes information about a topic to the broker, the broker distributes that information to all registered subscribers.

Business-critical data. Data that is not stored elsewhere in the system. If this data is lost, important information, or a change in state within the system, is lost.

Capacity. The number of requests for the service that can be processed by the system as a whole within a given time interval.

Certificate Authority (CA). A entity, identified by their public certificate, that is trusted to sign the certificates of others.

Certificate repository. A password-protected store containing the public certificates of trusted certificate authority (CA) and also personal certificates, including their private keys.

Certificate Revocation List (CRL). A list of revoked certificates previously signed by a certificate authority that have been compromised and should not be trusted. Usually queried from a Lightweight Directory Access Protocol (LDAP) server.

Channel. A network communications link between two queue managers over which messages flow, or a network communications link between an application and a queue manager over which message queue interface (MQI) commands flow.

CipherSpec. A method for specifying a CipherSuite that assumes RSA is used as the key exchange protocol.

CipherSuite. A method for specifying the key exchange, encryption, and message authentication code (MAC) algorithms to use for securing a channel with Secure Sockets Layer (SSL) or Transport Layer Security (TSL).

Client application. An application connecting to a WebSphere MQ queue manager over a network.

Cluster. See Queue manager cluster.

Cluster message channel. A message channel between two queue managers within the same queue manager cluster.

Data conversion. The process of converting the binary representation of characters and numbers from that of one environment to that of another.

Disaster recovery. The process of recovering data and restoring access services after a significant event that impacts multiple nodes in a system.

Distributed message channel. A message channel between two queue managers where all messages are transferred from a single transmission queue on one queue manager to destination queues on the remote queue manager.

Event information. Information that describes an event that has occurred and might require action.

Exactly once delivery. An assurance provided by a messaging infrastructure that a message arrives at its destination, that it arrives once, and that it arrives once only.

Failover. The process of making the data held by a node and the services provided by that node available on a different node.

Fix pack. A maintenance delivery that increments the fourth, Fixpack, digit in the WebSphere MQ version. The format of the WebSphere MQ V6.0 version, as displayed with the **dspmqver** command, is Version.Release.Modification.Fixpack.

Global unit of work. A unit of work that includes actions on multiple different resources, which might include WebSphere MQ and database products. This unit of work is coordinated by a transaction manager.

High availability. High availability encompasses the concepts of service availability and message availability in a message queuing environment.

High availability cluster. A mechanism to automatically failover the data held by a node and the services provided by that node in the event that that node experiences a planned or unplanned outage.

Hub and spoke architecture. A WebSphere MQ infrastructure architecture in which services are provided by a small number of hub queue managers, and access to those services is extended through a larger number of intermediate spoke queue managers interconnected with those hub queue managers.

IBM Message Service Client (XMS). An application programming interface for the C and C++ programming languages and the .NET environment, which is consistent with the Java Message Service application programming interface.

Java Message Service (JMS). An industry standardized application programming interface for the Java programming language, which is part of the Java 2 Platform, Enterprise Edition standard.

Load. The number of attempts made to request a service within a given time interval.

Maintenance delivery. A package of fixes that can be applied to a WebSphere MQ installation. These are released at regular intervals through the WebSphere MQ support Web site.

Maintenance pack. A maintenance delivery that increments the third, Modification, digit in the WebSphere MQ version. The format of the WebSphere MQ V6.0 version, as displayed with the **dspmqver** command, is Version.Release.Modification.Fixpack. See also Fix pack.

Message. A piece of information, with addressing or other meta information associated, that can be passed between software components.

Message availability. If a failure occurs on a node through which messages flow, whether those messages be recovered if that node fails and how quickly they become available.

Message channel. A network communications link between two queue managers over which messages flow.

Message channel agent (MCA). A component of a WebSphere MQ queue manager, or a WebSphere MQ client product, that forms one half of a channel, establishing network communications with, or responding to network communications from, a partner MCA.

Message descriptor. See WebSphere MQ message descriptor (MQMD)

Message queuing. A middleware technique that allows unlike software components to interact asynchronously through a queue.

Message queue interface (MQI) channel. A network communications link between an application and a queue manager over which message queue interface (MQI) commands flow.

Middleware. A software infrastructure layer between applications and the infrastructure components with which they interact, which is common to multiple nodes in a system, and simplifies interaction between the unlike software and hardware components that reside on those nodes.

Object Authority Manager (OAM). A component of a WebSphere MQ queue manager that performs authority checking.

Outage. A period of time when a service or services provided by a system are unavailable.

Performance. The time taken between submitting a request for a service and completion of that service. How the start and end points of a service are determined are specific to the function being performed by the service.

Personal certificate. A public certificate that can be used to identify an entity combined with the private key for that certificate.

Planned outage. Periods when a service or services are unavailable in order to perform planned work on those services.

Point to point messaging. The sending of messages from one location to a single destination that is determined based on addressing information provided by the sender of the message.

Polling. Repeatedly requesting a piece of information at regular intervals in order to detect changes in that information.

Production environment. An environment through which real services are made available within or outside of the business, or both.

Proxy. An interface between an existing service, usually with a proprietary interface, and a middleware layer that is used by other nodes in the system to access that service.

Publish/subscribe messaging. A model of messaging in which the producers of information do not have direct knowledge of the consumers of that information, which can be zero or many.

Publisher. In a publish/subscribe messaging model, a publisher produces information on a particular topic which is distributed to registered subscribers on that topic by a broker.

Quality assurance environment. An environment created to simulate a production environment for testing and development of application and infrastructure changes.

Query data. Transient data being sent through a system derived from data that is stored safely within the system.

Queue. A container for messages from which messages are usually retrieved in first-in first-out order, which can be used as an asynchronous buffer between two software components.

Queue manager. Queue managers are the interconnected nodes within a WebSphere MQ infrastructure that maintain the messages and queues, provide data integrity, and provide applications with access to the infrastructure to send and receive messages.

Queue manager cluster. A mechanism provided by WebSphere MQ to interconnect queue managers in a flexible way, which simplifies administration and provides workload balancing facilities for scalability and service availability.

Queue name resolution. The action performed by a queue manager whenever an application or channel attempts to open a queue in order to put a message on a queue hosted by that queue manager or to send a message through that queue manager.

Queue sharing group. A feature of WebSphere MQ for z/OS that allows applications connected to multiple queue managers, running on different z/OS systems within a sysplex, to get and put messages to the same queue.

Request/reply messaging. Asynchronous communication between two software components in which a request message is sent and a reply message is returned following processing of the request.

Resource manager. A component that, under the control of a transaction manager, manages an individual resource that is participating in a global unit of work.

Scalability. How easily the capacity of the system can be increased to cope with increased load and how this affects performance.

Secure Sockets Layer (SSL). An industry standardized technology to provide authentication and secure communication.

Security of access. Ensuring that services are only accessible by those entities authorized to do so.

Security of communications. Ensuring that sensitive information cannot be intercepted or tampered with during communication.

Send and forget messaging. Sending messages without requiring a reply upon the processing of those messages, thus relying on the exactly once delivery assurance of the message queuing infrastructure to deliver the message.

Service availability. If a planned or unplanned outage affects nodes in the system that provide a service, whether the system as a whole can continue to provide that service.

State information. Information that changes over time, but only has one value at any point in time.

Statement of environment (SOE). Details of the supported versions of operating systems, compilers, and other software components that interact with the WebSphere MQ product. SOEs are available through the WebSphere MQ support Web page.

Subscriber. In a publish/subscribe messaging model, a subscriber registers with a broker to receive all information published about a particular topic.

SupportPac. A package of additional functionality or documentation for the WebSphere MQ product, distributed through the IBM SupportPacs Web page. SupportPacs are not related to fix pack maintenance deliveries.

Topic. In a publish/subscribe messaging model, a topic uses group information so that publishers that produce information about a topic can be loosely coupled with the subscribers that consume the information about that topic.

Transaction. The mechanism by which multiple actions, possibly on multiple resources, can be grouped together in a unit of work.

Transaction manager. The component that manages the resources participating in a global unit of work.

Transport Layer Security (TLS). An industry standardized technology to provide authentication and secure communication.

Unit of work. A logical grouping of actions that must either all succeed or all fail.

Unplanned outage. Periods when a service or services become unavailable unexpectedly.

Web services. A standardized way to describe and invoke services.

WebSphere MQ message descriptor (MQMD). A data structure, associated with each WebSphere MQ message, that contains meta information associated with that message, such as identifying and type information.

WebSphere MQ object model. A defined set of classes, methods, and properties to interact with WebSphere MQ that are implemented for multiple object-oriented programming languages, including Java and C++, building on the facilities provided by the message queue interface (MQI).

Abbreviations and acronyms

API	Application Programming Interface
ARM	Automatic Restart Manager
CCDT	Client Channel Definition Table
CF	Coupling Facility
CL	Control Language
COA	Confirm on Arrival
COD	Confirm on Delivery
COM	Component Object Model
CRL	Certificate Revocation List
DN	Distinguished Name
ESM	External Security Manager
FFST	First-Failure Support Technology
FIPS	Federal Information Processing Standard
FIFO	First-In First-Out
GUI	Graphical User Interface
GSKit	IBM Global Security Toolkit
IBM	International Business Machines Corporation
IDE	Integrated Development Environment
IFS	Integrated File System
IPL	Initial Program Load
IPT	WebSphere MQ internet pass-thru
ISPF	Interactive System Productivity Facility
IT	Information Technology
ITSO	International Technical Support Organization
IPv6	Internet Protocol Version 6
J2EE	Java 2 Platform, Enterprise Edition
JDK	Java Development Kit
JKS	Java KeyStore
JMS	Java Message Service
JNDI	Java Naming and Directory Interface
JSSE	Java Secure Sockets Extension
JVM	Java Virtual Machine
LDAP	Lightweight Directory Access Protocol
LPAR	Logical Partition
MAC	Message Authentication Code
MCA	Message Channel Agent
MMC	Microsoft Management Console
MQAX	WebSphere MQ Automation Classes for ActiveX
MQI	Message Queue Interface
MQOD	WebSphere MQ Object Descriptor
MQSC	WebSphere MQ Script Command
MQMD	Message Descriptor Structure
MQPMO	Message Options Structure
NAT	Network Address Translation
OAM	Object Authority Manager
PCF	Programmable Command Formats
PMR	Problem Management Record
QA	Quality Assurance
QMID	Queue Manager Identifier
QSG	Queue Sharing Group

RACF	Resource Access Control Facility
SDSF	System Display and Search Facility
SOAP	Simple Object Access Protocol
SOE	Statement of Environment
SSL	Secure Sockets Layer
TSL	Transport Layer Security
TCF	Topic Connection Factory
TCP/IP	Transmission Control Protocol/Internet Protocol
TLS	Transport Layer Security
TSO	Time Sharing Option
WSDL	Web Services Description Language
XML	Extensible Markup Language
XMS	IBM Message Service Client

Related publications

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

IBM Redbooks

For information about ordering these publications, see "How to get IBM Redbooks" on page 390. Note that some of the documents referenced here may be available in softcopy only.

- *WebSphere MQ Solutions in a Microsoft .NET Environment*, SG24-7012
- *WebSphere Business Integration Pub/Sub Solutions*, SG24-6088
- *MQSeries Publish/Subscribe Applications*, SG24-6282

Other publications

These publications are also relevant as further information sources.

Multiplatform publications:

- *WebSphere MQ Application Programming Guide*, SC34-6595
- *WebSphere MQ Application Programming Reference*, SC34-6596
- *WebSphere MQ Bibliography and Glossary*, SC34-6603
- *WebSphere MQ Clients*, GC34-6590
- *WebSphere MQ Constants*, SC34-6607
- *WebSphere MQ Intercommunication*, SC34-6587
- *WebSphere MQ Messages*, GC34-6601
- *WebSphere MQ Migration Information*, SC34-6604
- *Monitoring WebSphere MQ*, SC34-6593
- *WebSphere MQ Programmable Command Formats and Administration Interface*, SC34-6598
- *WebSphere MQ Publish/Subscribe User's Guide*, SC34-6606
- *WebSphere MQ Queue Manager Clusters*, SC34-6589

- *WebSphere MQ Script (MQSC) Command Reference*, SC34-6597
- *WebSphere MQ Security*, SC34-6588
- *WebSphere MQ System Administration Guide*, SC34-6584
- *WebSphere MQ Transport for SOAP*, SC34-6651
- *WebSphere MQ Using C++*, SC34-6592
- *WebSphere MQ Using Java*, SC34-6591
- *WebSphere MQ Using .Net*, GC34-6605

Platform-specific publications:

- *WebSphere MQ for AIX V6.0 Quick Beginnings*, GC34-6478
- *WebSphere MQ for HP-UX V6.0 Quick Beginnings*, GC34-6479
- *WebSphere MQ for iSeries V6.0 Application Programming Reference (ILE RPG)*, SC34-6599
- *WebSphere MQ for iSeries V6.0 Quick Beginnings*, GC34-6481
- *WebSphere MQ for iSeries V6.0 System Administration Guide*, SC34-6586
- *WebSphere MQ for Linux V6.0 Quick Beginnings*, GC34-6480
- *WebSphere MQ for Solaris V6.0 Quick Beginnings*, GC34-6477
- *WebSphere MQ for Windows V6.0 Quick Beginnings*, GC34-6476
- *WebSphere MQ for Windows V6.0, Using the Component Object Model Interface*, SC34-6594
- *WebSphere MQ for z/OS V6.0 Concepts and Planning Guide*, GC34-6582
- *WebSphere MQ for z/OS V6.0 System Setup Guide*, SC34-6583
- *WebSphere MQ for z/OS V6.0 System Administration Guide*, SC34-6585
- *WebSphere MQ for z/OS V6.0 Messages and Codes*, GC34-6602
- *WebSphere MQ for z/OS V6.0 Problem Determination Guide*, GC34-6600

Online resources

These Web sites and URLs are also relevant as further information sources:

- All WebSphere MQ guides

  http://www.ibm.com/software/integration/wmq/library/

- WebSphere MQ multiplatform guides

  http://www.ibm.com/software/integration/mqfamily/library/manualsa/manuals/crosslatest.html
- WebSphere MQ platform-specific guides

  http://www.ibm.com/software/integration/mqfamily/library/manualsa/manuals/platspecific.html
- IBM WebSphere MQ Information Center

  http://publib.boulder.ibm.com/infocenter/wmqv6/v6r0/index.jsp
- WebSphere MQ support site

  http://www.ibm.com/software/integration/wmq/support/
- IBM developerWorks WebSphere community

  http://www.ibm.com/developerworks/websphere/community
- *Understanding high availability with WebSphere MQ* white paper

  http://www.ibm.com/developerworks/websphere/library/techarticles/0505_hiscock/0505_hiscock.html
- Platforms supported by IBM for hosting queue managers (statements of environment)

  http://www.ibm.com/software/integration/websphere/mqplatforms/supported.html
- SupportPac IA94: IBM Message Service Client for C/C++

  http://www.ibm.com/support/docview.wss?rs=171&uid=swg24007092&loc=en_US&cs=utf-8&lang=en
- SupportPac MA89: Perl language support for MQSeries

  http://www.ibm.com/support/docview.wss?rs=171&uid=swg24000208&loc=en_US&cs=utf-8&lang=en
- SupportPac MH01: WebSphere MQ Explorer Healthcheck Plug-in

  http://www.ibm.com/support/docview.wss?rs=171&uid=swg24010096
- SupportPac MS0B: WebSphere MQ Java classes for PCF

  http://www.ibm.com/support/docview.wss?rs=171&uid=swg24000668&loc=en_US&cs=utf-8&lang=en
- SupportPac MS81: WebSphere MQ internet pass-thru

  http://www.ibm.com/support/docview.wss?rs=203&uid=swg24006386&loc=en_US&cs=utf-8&lang=en
- The XA Specification from The Open Group

  http://www.opengroup.org/bookstore/catalog/c193.htm

- WebSphere MQ Extended Transactional Client

  http://www.opengroup.org/bookstore/catalog/c193.htm
- IBM KeyMan utility

  http://www.alphaworks.ibm.com/tech/keyman
- Software support: Submit/track problems

  http://www.ibm.com/software/support/probsub.html
- Technote “MustGather: Read first for all WebSphere MQ v5.3, v5.3.1, and v6.0 products”

  http://www.ibm.com/support/docview.wss?rs=171&uid=swg21177923

How to get IBM Redbooks

You can search for, view, or download Redbooks, Redpapers, Hints and Tips, draft publications and Additional materials, as well as order hardcopy Redbooks or CD-ROMs, at this Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Index

WebSphere MQ V6 Fundamentals

(0.5” spine)
0.475”<->0.875”
250 <-> 459 pages

WebSphere MQ V6 Fundamentals

Overview of message queuing and WebSphere MQ V6.0

Broad technical introduction to the Websphere MQ product

Hands-on guide to the first steps of building a WebSphere MQ infrastructure

This IBM Redbook describes the fundamental concepts and benefits of message queuing technology. This book is an update of a very popular Redpaper (REDP-0021) based on IBM WebSphere MQ Versions 5.0 to 5.2.

This publication provides a design-level overview and technical introduction for the established and reliable WebSphere MQ product.

A broad technical understanding of the WebSphere MQ product can improve design and implementation decisions for WebSphere MQ infrastructures and applications. To reduce the time required to gain this understanding, this IBM Redbook summarizes relevant information from across the WebSphere MQ product documentation.

We also include hands-on security and troubleshooting sections to aid understanding and provide a reference for common administrative actions performed when building and maintaining WebSphere MQ infrastructures.

In the appendix, we provide a summary of the new features in WebSphere MQ Version 6.0.

SG24-7128-00 ISBN 073849299X