Sun Java™ System

Message Queue 3 Administration Guide

2005Q4

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054
U.S.A.

Part No: 819-2571-10

Contents

List of Figures

List of Tables

List of Procedures

Preface

This Sun Java™ System *Message Queue Administration Guide* describes Sun Java System Message Queue 3 2005Q4 (Message Queue 3.6), providing the information you need in order to administer a Message Queue messaging system.

This preface contains the following sections:

Who Should Use This Book

This manual is intended for administrators and application developers who need to perform Message Queue administrative tasks. A Message Queue *administrator* is responsible for setting up and managing a Message Queue messaging system, especially the message server at the heart of the system.

Before You Read This Book

Before reading this manual, you should read the *Message Queue Technical Overview* to become familiar with the Message Queue implementation of the Java Message Specification, with the components of the Message Queue service, and with the basic process of developing, deploying, and administering a Message Queue application.

How This Book Is Organized

Table 1 briefly describes the contents of this manual.

Table 1 Contents of This Manual

Part/Chapter	Description
Part I, "Introduction to Message Queue Administration"	
Chapter 1, "Administrative Tasks and Tools"	Introduces Message Queue administrative tasks and tools.
Chapter 2, "Quick-Start Tutorial"	Provides a hands-on tutorial to acquaint you with the Message Queue Administration Console.
Part II, "Administration Tasks"	
Chapter 3, "Starting Brokers and Clients"	Describes how to start the Message Queue broker and clients.
Chapter 4, "Configuring a Broker"	Describes how configuration properties are set and read, and gives an introduction to the configurable aspects of the broker. Also describes how to set up a file or database to perform persistence functions.

Table 1 Contents of This Manual *(Continued)*

Part/Chapter	Description
Chapter 5, "Managing a Broker"	Describes broker management tasks.
Chapter 6, "Managing Physical Destinations"	Describes management tasks relating to physical destinations.
Chapter 7, "Managing Security"	Describes security-related tasks, such as managing password files, authentication, authorization, and encryption.
Chapter 8, "Managing Administered Objects"	Describes the object store and shows how to perform tasks related to administered objects (connection factories and destinations).
Chapter 9, "Working With Broker Clusters"	Describes how to set up and manage a cluster of Message Queue brokers.
Chapter 10, "Monitoring a Message Server"	Describes how to set up and use Message Queue monitoring facilities.
Chapter 11, "Analyzing and Tuning a Message Service"	Describes techniques for analyzing and optimizing message server performance.
Chapter 12, "Troubleshooting Problems"	Provides suggestions for determining the cause of common Message Queue problems and the actions you can take to resolve them.
Part III, "Reference"	
Chapter 13, "Command Line Reference"	Provides syntax and descriptions for Message Queue command utilities.
Chapter 14, "Broker Properties Reference"	Lists and describes the properties you can use to configure a broker.
Chapter 15, "Physical Destination Property Reference"	Lists and describes the properties you can use to configure physical destinations.
Chapter 16, "Administered Object Attribute Reference"	Lists and describes the properties you can use to configure administered objects (connection factories and destinations).
Chapter 17, "JMS Resource Adapter Property Reference"	Lists and describes the properties you can use to configure the Message Queue Resource Adapter for use with an application server.
Chapter 18, "Metrics Reference"	Lists and describes the metrics produced by a Message Queue broker.

Table 1 Contents of This Manual *(Continued)*

Part/Chapter	Description
Part IV, “Appendixes”	
Appendix A, “Platform-Specific Locations of Message Queue Data”	Lists the locations of Message Queue files on each supported platform.
Appendix B, “Stability of Message Queue Interfaces”	Describes the stability of various Message Queue interfaces.
Appendix C, “HTTP/HTTPS Support”	Describes how to set up and use the Hypertext Transaction protocol (HTTP) for Message Queue communication.
Appendix D, “Frequently Used Command Utility Commands”	Lsts some frequently used Message Queue Command utility (imqcmd) commands.

Conventions Used In This Book

This section describes the conventions used in this manual.

Text Conventions

Table 2 summarizes the text conventions used in this manual.

Table 2 Text Conventions

Format	Description
italics	Italicized text represents a placeholder to be replaced with an appropriate clause or value. It is also used for document titles, for emphasis, and for key words or phrases being introduced.
`monospace`	Monospace text represents example code; commands that you enter on the command line; directory, file, or path names; error message text; class names; method names (including all elements in the signature); package names; reserved words; and URLs.
`[]`	Square brackets indicate optional values in a command line syntax statement.
ALL CAPS	Text in all capitals represents environment variables (such as `IMQ_HOME`) or acronyms (such as JMS, GIF, or HTML).
Key+Key	Simultaneous keystrokes are joined with a plus sign: Ctrl+`A` means press the Ctrl and `A` keys simultaneously.

Table 2 Text Conventions *(Continued)*

Format	Description
Key-Key	Consecutive keystrokes are joined with a hyphen: Esc-`S` means press the Esc key, release it, and then press the `S` key.

Directory Variable Conventions

Message Queue makes use of three directory variables; how they are set varies from platform to platform. Table 3 describes these variables and how they are used on the Solaris™, Linux, and Windows platforms.

NOTE In this manual, directory variables are shown without the usual platform-specific syntax (such as `$IMQ_HOME` on UNIX). Path names generally use UNIX directory separator notation (`/`).

Table 3 Message Queue Directory Variables

Variable	Description
`IMQ_HOME`	Refers to the Message Queue base directory (root installation directory): • Unused on Solaris and Linux; there is no Message Queue base directory. • On Windows, set by the Message Queue installer (by default, to `C:\Program Files\Sun\MessageQueue3`). • For Sun Java System Application Server on Solaris and Windows, set to `/imq` under the Application Server base directory.
`IMQ_VARHOME`	The directory in which Message Queue temporary or dynamically created configuration and data files are stored; can be set as an environment variable to point to any directory. • On Solaris, defaults to `/var/imq`. • On Linux, defaults to `/var/opt/sun/mq` directory. • On Windows, defaults to `IMQ_HOME\var`. • For Sun Java System Application Server, Evaluation Edition, on Solaris, defaults to `IMQ_HOME/var`. • For Sun Java System Application Server on Windows, defaults to `IMQ_HOME\var`.

Table 3 Message Queue Directory Variables *(Continued)*

Variable	Description

`IMQ_JAVAHOME`	The location of the Java™ runtime (JRE) required by Message Queue executables. Set by default to look in the following locations in the order shown, but can optionally be set to wherever the required JRE resides.
	• On Solaris 8 or 9:
	`/usr/jdk/entsys-j2se`
	`/usr/jdk/jdk1.5.*`
	`/usr/jdk/j2sdk1.5.*`
	`/usr/j2se`
	• On Solaris 10:
	`/usr/jdk/entsys-j2se`
	`/usr/java`
	`/usr/j2se`
	• On Linux:
	`/usr/jdk/entsys-j2se`
	`/usr/java/jre1.5.*`
	`/usr/java/jdk1.5.*`
	`/usr/java/jre1.4.2*`
	`/usr/java/j2sdk1.4.2*`
	• On Windows:
	`IMQ_HOME\jre*`

Related Documentation

The information resources listed in this section provide further information about Message Queue in addition to that provided in this manual.

Message Queue Documentation Set

The Message Queue documentation set comprises the documents listed in Table 4.

Table 4 Message Queue Documentation Set

Document	Audience	Description
Message Queue Installation Guide	Developers and administrators	Explains how to install Message Queue software on Solaris, Linux, and Windows platforms.
Message Queue Release Notes	Developers and administrators	Includes descriptions of new features, limitations, and known bugs, as well as technical notes.
Message Queue Technical Overview	Developers and administrators	Introduces Message Queue concepts, features, and components.
Message Queue Administration Guide	Administrators and developers	Provides background and information needed to perform administrative tasks using Message Queue administration tools.
Message Queue Developer's Guide for Java Clients	Developers	Provides information on developing Java client programs using the Message Queue implementation of the JMS and SOAP/JAXM specifications.
Message Queue Developer's Guide for C Clients	Developers	Provides information on developing C client programs using the C application programming interface (C-API) to the Message Queue message service.

Java Message Service Specification

The Message Queue message service conforms to the Java Message Service (JMS) application programming interface, described in the *Java Message Service Specification*. This document can be found at the URL

`http://java.sun.com/products/jms/docs.html`

Online Help

Online help is available for the Message Queue command line utilities; see Chapter 13, "Command Line Reference," for details. The Message Queue graphical user interface (GUI) administration tool, the Administration Console, also includes a context-sensitive online help facility; see "Administration Console Online Help" on page 40.

JavaDoc

JMS and Message Queue API documentation in JavaDoc format is provided at the locations shown in Table 5. This documentation can be viewed in any HTML browser, such as Netscape or Internet Explorer. It includes standard JMS API documentation as well as Message Queue–specific APIs.

Table 5 JavaDoc Locations

Platform	Location
Solaris	`/usr/share/javadoc/imq/index.html`
Linux	`/opt/sun/mq/javadoc/index.html/`
Windows	`IMQ_HOME/javadoc/index.html`

Example Client Applications

The Message Queue installation includes a directory containing several example client applications. See Appendix A, "Platform-Specific Locations of Message Queue Data," for exact locations depending on the particular platform you are using. The `README` files located in that directory and in each of its subdirectories provide descriptive information about the example applications.

Related Third-Party Web Sites

Where relevant, this manual refers to third-party URLs that provide additional, related information.

NOTE Sun is not responsible for the availability of third-party Web sites mentioned in this manual. Sun does not endorse and is not responsible or liable for any content, advertising, products, or other materials available on or through such sites or resources. Sun will not be responsible or liable for any actual or alleged damage or loss caused by or in connection with the use of or reliance on any such content, goods, or services available on or through such sites or resources.

Sun Welcomes Your Comments

Sun is always interested in improving its documentation and welcomes your comments and suggestions. To share your comments, go to

`http://docs.sun.com`

and click **Send comments**. In the resulting online form, provide the document title and part numberd along with your comment. (The part number is a seven-digit or nine-digit number that can be found on the title page of the book or at the top of the document.)

Part I

Introduction to Message Queue Administration

Chapter 1

Administrative Tasks and Tools

This chapter provides an overview of Sun Java™ System Message Queue administrative tasks and the tools for performing them, focusing on common features of the command line administration utilities. It consists of the following sections:

- “Administrative Tasks” on page 31
- “Administration Tools” on page 34

Administrative Tasks

The typical administrative tasks to be performed depend on the nature of the environment in which you are running Message Queue. The demands of a software development environment in which Message Queue applications are being developed and tested are different from those of a production environment in which such applications are deployed to accomplish useful work. The following sections summarize the typical administrative requirements of these two different types of environment.

Administration in a Development Environment

In a development environment, the emphasis is on flexibility. The Message Queue message server is needed principally for testing applications under development. Administration is generally minimal, with programmers often administering their own systems. Such environments are typically distinguished by the following characteristics:

- Simple startup of brokers for use in testing

- Administered objects instantiated in client code rather than created administratively
- Auto-created destinations
- File-system object store
- File-based persistence
- File-based user repository
- No master broker in multiple-broker clusters

Administration in a Production Environment

In a production environment in which applications must be reliably deployed and run, administration is more important. Administrative tasks to be performed depend on the complexity of the messaging system and of the applications it must support. Such tasks can be classified into two general categories: setup operations and maintenance operations.

Setup Operations

Administrative setup operations in a production environment typically include some or all of the following:

Administrator security

- Setting the password for the default administrative user (`admin`) ("Changing the Default Administrator Password" on page 138)
- Controlling individual or group access to the administrative connection service ("Access Control for Connection Services" on page 145) and the dead message queue ("Access Control for Physical Destinations" on page 146)
- Regulating administrative group access to a file-based or Lightweight Directory Access Protocol (LDAP) user repository ("Groups" on page 135, "Setting Up Access Control for Administrators" on page 141)

General security

- Managing the contents of a file-based user repository ("Populating and Managing a User Repository" on page 137) or configuring the broker to use an existing LDAP user repository ("Editing the Instance Configuration File" on page 139)

- Controlling the operations that individual users or groups are authorized to perform ("Authorizing Users: The Access Control Properties File" on page 142)
- Setting up encryption services using the Secure Socket Layer (SSL) ("Working With an SSL-Based Service" on page 148)

Administered objects

- Setting up and configuring an LDAP object store ("LDAP Server Object Stores" on page 162)
- Creating connection factories and destinations ("Adding Administered Objects" on page 173)

Broker clusters

- Creating a cluster configuration file ("Using a Cluster Configuration File" on page 182)
- Designating a master broker ("Master Broker" on page 186)

Persistence

- Configuring a broker to use a persistent store ("Configuring a Persistent Data Store" on page 93).

Memory management

- Setting a destination's configuration properties to optimize its memory usage ("Updating Physical Destination Properties" on page 123, Chapter 15, "Physical Destination Property Reference")

Maintenance Operations

Because application performance, reliability, and security are at a premium in production environments, message server resources must be tightly monitored and controlled through ongoing administrative maintenance operations, including the following:

Broker administration and tuning

- Using broker metrics to tune and reconfigure a broker (Chapter 11, "Analyzing and Tuning a Message Service")
- Managing broker memory resources ("Routing Services" on page 78)

- Creating and managing broker clusters to balance message load (Chapter 9, “Working With Broker Clusters”)
- Recovering failed brokers (“Starting Brokers” on page 66).

Administered objects

- Adjusting connection factory attributes to ensure the correct behavior of client applications (“Connection Factory Attributes” on page 165)
- Monitoring and managing physical destinations (Chapter 6, “Managing Physical Destinations”)
- Controlling user access to destinations (“Access Control for Physical Destinations” on page 146)

Client management

- Monitoring and managing durable subscriptions (see “Managing Durable Subscriptions” on page 112).
- Monitoring and managing transactions (see “Managing Transactions” on page 113).

Administration Tools

Message Queue administration tools fall into two categories:

- Command line utilities
- The graphical Administration Console

Command Line Utilities

All Message Queue utilities are accessible via a command line interface. Utility commands share common formats, syntax conventions, and options. They include the following:

- The *Broker utility* (`imqbrokerd`) starts up brokers and specifies their configuration properties, including connecting them together into a *cluster*.
- The *Command utility* (`imqcmd`) controls brokers and their resources and manages physical destinations.

- The *Object Manager utility* (`imqobjmgr`) manages provider-independent *administered objects* in an object store accessible via the Java Naming and Directory Interface (JNDI).
- The *Database Manager utility* (`imqdbmgr`) creates and manages databases for persistent storage that conform to the Java Database Connectivity (JDBC) standard.
- The *User Manager utility* (`imqusermgr`) populates a file-based user repository for user authentication and authorization.
- The *Service Administrator utility* (`imqsvcadmin`) installs and manages a broker as a Windows service.
- The *Key Tool utility* (`imqkeytool`) generates self-signed certificates for Secure Socket Layer (SSL) authentication.

See Chapter 13, "Command Line Reference," for detailed information on the use of these utilities.

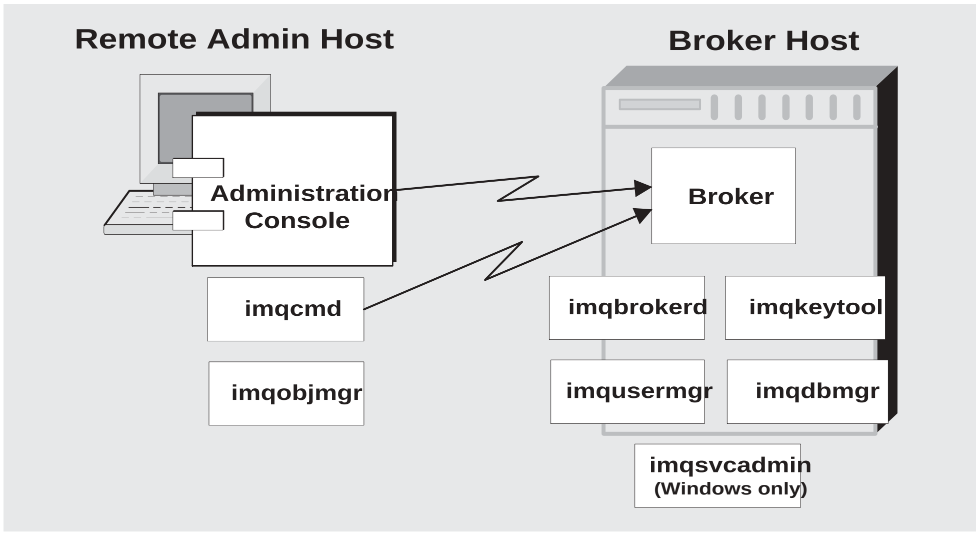
Administration Console

The Message Queue *Administration Console* combines some of the capabilities of the Command and Object Manager utilities. You can use it to perform the following tasks:

- Connect to and control a broker remotely
- Create and manage physical destinations
- Create and manage administered objects in a JNDI object store

However, you cannot use the Administration Console to perform such tasks as starting up a broker, creating broker clusters, managing a JDBC database or a user repository, installing a broker as a Windows service, or generating SSL certificates. For these, you need the other command line utilities (Broker, Database Manager, User Manager, Service Administrator, and Key Tool), which cannot operate remotely and must be run on the same host as the broker they manage (see Figure 1-1).

Figure 1-1 Local and Remote Administration Utilities



See Chapter 2, "Quick-Start Tutorial," for a brief, hands-on introduction to the Administration Console. More detailed information on its use is available through its own help facility.

Chapter 2

Quick-Start Tutorial

This quick-start tutorial provides a brief introduction to Message Queue administration by guiding you through some basic administrative tasks using the Message Queue Administration Console, a graphical interface for administering a message broker and object store. The chapter consists of the following sections:

The tutorial sets up the physical destinations and administered objects needed to run a simple JMS-compliant application, `HelloWorldMessageJNDI`. The application is available in the `helloworld` subdirectory of the example applications directory (`demo` on the Solaris and Windows platforms or `examples` on Linux; see Appendix A, "Platform-Specific Locations of Message Queue Data"). In the last part of the tutorial, you will run this application.

NOTE You must have the Message Queue product installed in order to follow the tutorial. If necessary, see the *Message Queue Installation Guide* for instructions.

The tutorial is only a basic introduction; it is not a substitute for reading the documentation. By following the steps described in the tutorial, you will learn how to

- Start a message broker
- Connect to a broker and use the Administration Console to manage it
- Create physical destinations on the broker
- Create an object store and use the Administration Console to connect to it
- Add administered objects to the object store and view their properties

---

NOTE The instructions given in this tutorial are specific to the Windows platform. Where necessary, supplemental notes are added for users of other platforms.

Some administrative tasks cannot be accomplished using the Administration Console. You must use command line utilities to perform such tasks as the following:

- Start up a broker
- Create a broker cluster
- Configure certain physical destination properties
- Manage a JDBC database for persistent storage
- Manage a user repository
- Install a broker as a Windows service
- Generate SSL certificates

All of these tasks are covered in later chapters of this manual.

Starting the Administration Console

To start the Administration Console, use one of the following methods:

- On Solaris, enter the command

  ```
  /usr/bin/imqadmin
  ```

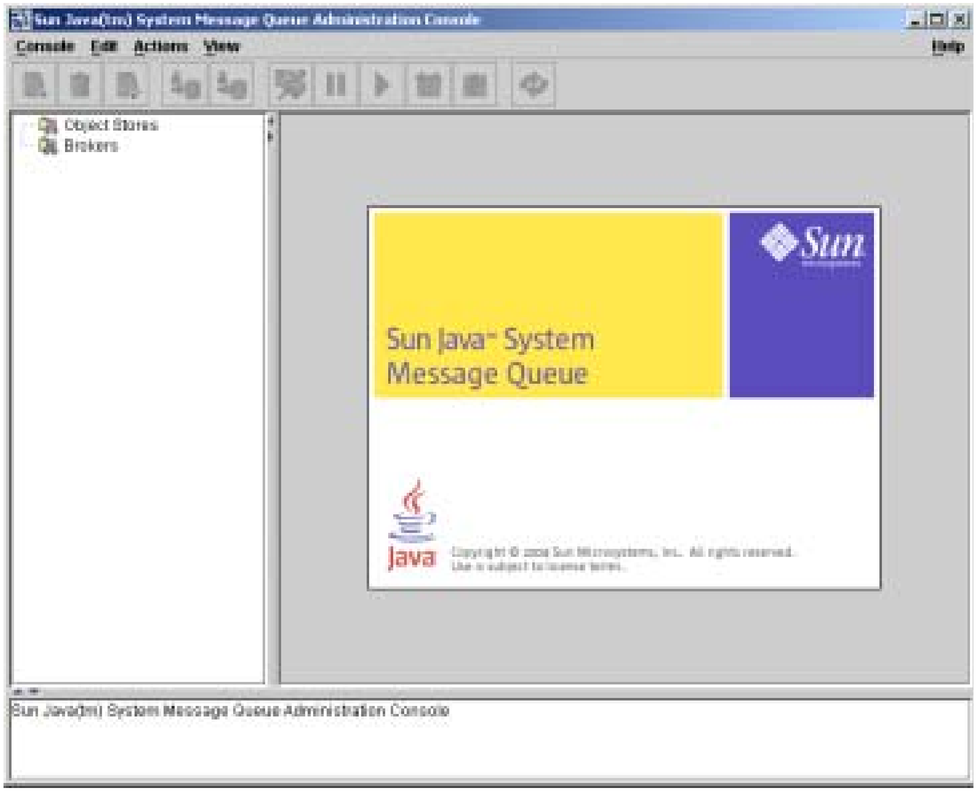
- On Linux, enter the command

  ```
  /opt/sun/mq/bin/imqadmin
  ```

- On Windows, choose Start > Programs > Sun Microsystems > Sun Java System Message Queue 3.6 > Administration.

You may need to wait a few seconds before the Administration Console window is displayed (see Figure 2-1).

Figure 2-1 Administration Console Window



Sun Java(tm) System Message Queue Administration Console

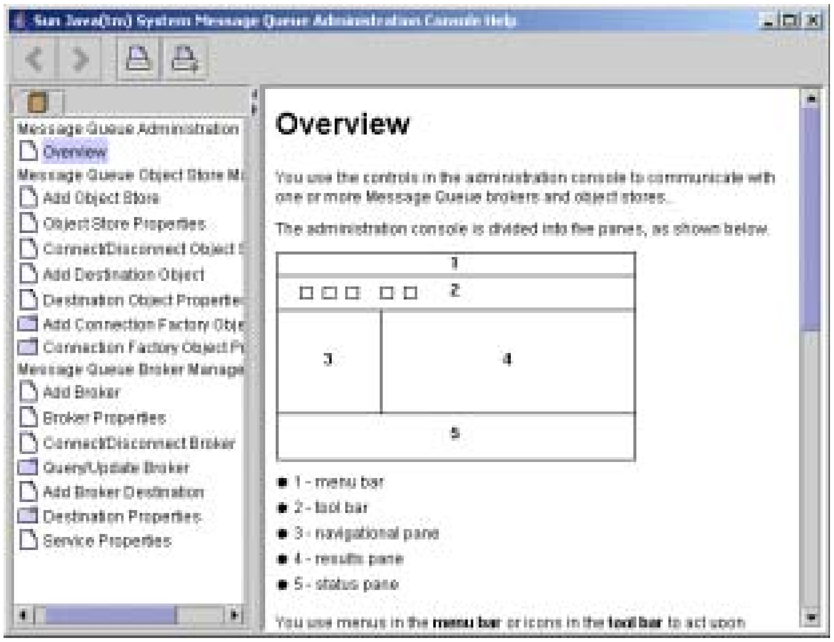
Take a few seconds to examine the Administration Console window. It has a menu bar at the top, a tool bar just below it, a navigation pane to the left, a result pane to the right (now displaying graphics identifying the Sun Java System Message Queue product), and a status pane at the bottom.

NOTE As you work with the Administration Console, you can use the Refresh command on the View menu to update the visual display of any element or group of elements, such as a list of brokers or object stores.

Administration Console Online Help

The Administration Console provides a help facility containing complete information about how to use the Console to perform administrative tasks. To use the help facility, pull down the Help menu at the right end of the menu bar and choose Overview. The Administration Console's Help window (Figure 2-2) will be displayed.

Figure 2-2 Administration Console Help Window


Sun Java(tm) System Message Queue Administration Console Help
Message Queue Administration
Overview
Message Queue Object Store M
Add Object Store
Object Store Properties
Connect/Disconnect Object
Add Destination Object
Destination Object Properti
Add Connection Factory Obje
Connection Factory Object P
Message Queue Broker Manage
Add Broker
Broker Properties
Connect/Disconnect Broker
Query/Update Broker
Add Broker Destination
Destination Properties
Service Properties
1
2
3
4
5

Overview

You use the controls in the administration console to communicate with one or more Message Queue brokers and object stores.

The administration console is divided into five panes, as shown below.

- 1 - menu bar
- 2 - tool bar
- 3 - navigational pane
- 4 - results pane
- 5 - status pane

You use menus in the **menu bar** or icons in the **tool bar** to act upon

The Help window's navigation pane, on the left, organizes topics into three areas: Message Queue Administration Console, Message Queue Object Store Management, and Message Queue Broker Management. Within each area are files and folders. The folders provide help for dialog boxes containing multiple tabs, the files for simple dialog boxes or individual tabs. When you select an item in the navigation pane, the result pane to the right shows the contents of that item. With the Overview item chosen, the result pane displays a skeletal view of the Administration Console window identifying each of the window's panes, as shown in the figure.

Your first task with the Administration Console will be to create a reference to a broker. Before you start, however, check the Help window for information. Click the Add Broker item in the Help window's navigation pane; the contents of the result pane will change to show text explaining what it means to add a broker and describing the use of each field in the Add Broker dialog box. Read through the help text, then close the Help window.

Working With Brokers

This section describes how to use the Administration Console to connect to and manage message brokers.

Starting a Broker

You cannot start a broker using the Administration Console. Instead, use one of the following methods:

- On Solaris, enter this command:

  ```
  /usr/bin/imqbrokerd
  ```

- On Linux, enter this command:

  ```
  /opt/sun/mq/bin/imqbrokerd
  ```

- On Windows, choose Start > Programs > Sun Microsystems > Sun Java System Message Queue 3.6 > Message Broker.

If you used the Windows Start menu, the command window will appear, indicating that the broker is ready by displaying lines like the following:

```
Loading persistent data...
Broker “imqbroker@stan:7676 ready.
```

Reactivate the Administration Console window. You are now ready to add the broker to the Console and connect to it. You do not have to start the broker before adding a reference to it in the Administration Console, but you must start it before you can connect to it.

Adding a Broker to the Administration Console

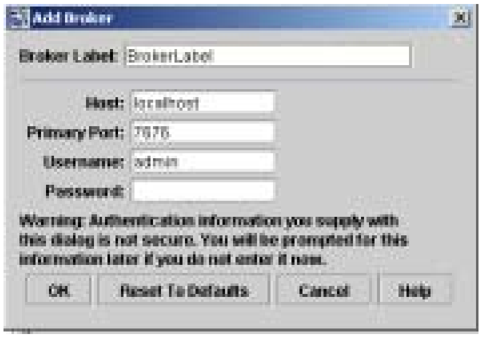
Adding a broker creates a reference to that broker in the Administration Console. After adding the broker, you can connect to it.

➤ To Add a Broker to the Administration Console

1. Click on the Brokers item in the Administration Console window's navigation pane and choose Add Broker from the Actions menu.

   Alternatively, you can right-click on Brokers and choose Add Broker from the pop-up context menu. In either case, the Add Broker dialog box (Figure 2-3) will appear.

Figure 2-3 Add Broker Dialog Box



Broker Label: BrokerLabel
Host: localhost
Primary Port: 7676
Username: admin
Password:

Warning: Authentication information you supply with this dialog is not secure. You will be prompted for this information later if you do not enter it now.

2. Enter a name for the broker in the Broker Label field.

   This provides a label that identifies the broker in the Administration Console.

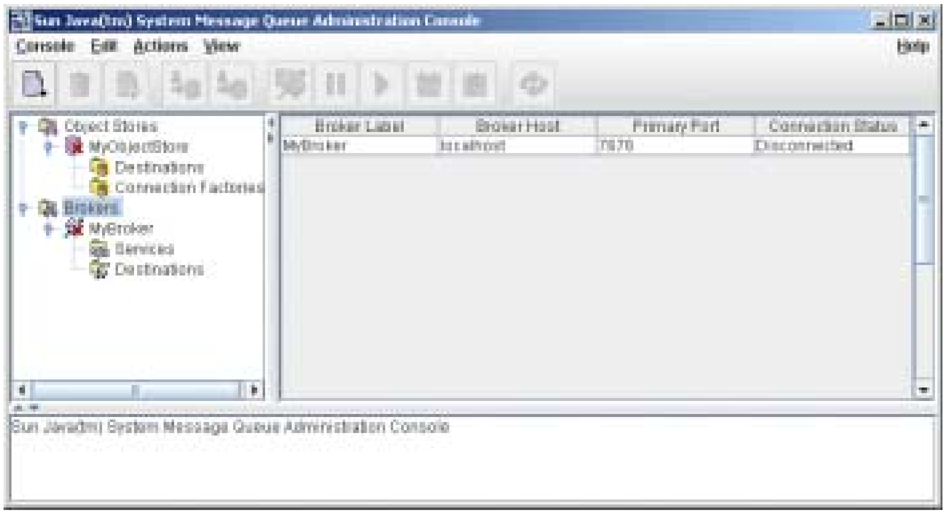
   Note the default host name (`localhost`) and primary port (7676) specified in the dialog box. These are the values you must specify later, when you configure the connection factory that the client will use to create connections to this broker.

   For this exercise, type the name `MyBroker` into the Broker Label field. Leave the Password field blank; your password will be more secure if you specify it at connection time.

3. Click OK to add the broker and dismiss the dialog box.

   The new broker will appear under Brokers in the navigation pane, as shown in Figure 2-4. The red X over the broker's icon indicates that it is not currently connected to the Administration Console.

Figure 2-4 Broker Displayed in Administration Console Window



Broker Label	Broker Host	Primary Port	Connection Status
MyBroker	localhost	7676	Disconnected

Sun Java(tm) System Message Queue Administration Console

Once you have added a broker, you can use the Properties command on the Actions menu (or the pop-up context menu) to display a Broker Properties dialog box, similar to the Add Broker dialog shown in Figure 2-3, to view or modify any of its properties.

Connecting to a Broker

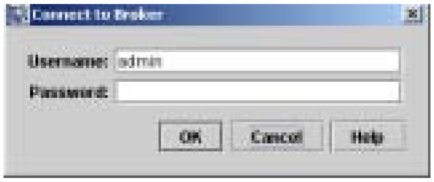
Now that you have added a broker to the Administration Console, you can proceed to connect to it.

➤ To Connect to a Broker

1. Click on the broker's name in the Administration Console window's navigation pane and choose Connect to Broker from the Actions menu.

   Alternatively, you can right-click on the broker's name and choose Connect to Broker from the pop-up context menu. In either case, the Connect to Broker dialog box (Figure 2-5) will appear.

Figure 2-5 Connect to Broker Dialog Box



2. Enter the user name and password with which to connect to the broker.

   The dialog box initially displays the default user name, `admin`. In a real-world environment, you should establish secure user names and passwords as soon as possible (see "Authenticating Users" on page 132); for this exercise, simply use the default value.

   The password associated with the default user name is also `admin`; type it into the Password field in the dialog box. This will connect you to the broker with administrative privileges.

3. Click OK to connect to the broker and dismiss the dialog box.

Once you have connected to the broker, you can use the commands on the Actions menu (or the context menu) to perform the following operations on a selected broker:

- Pause Broker temporarily suspends the operation of a running broker.
- Resume Broker resumes the operation of a paused broker.
- Restart Broker reinitializes and restarts a broker.
- Shut Down Broker terminates the operation of a broker.
- Query/Update Broker displays or modifies a broker's configuration properties.
- Disconnect from Broker terminates the connection between a broker and the Administration Console.

Viewing Connection Services

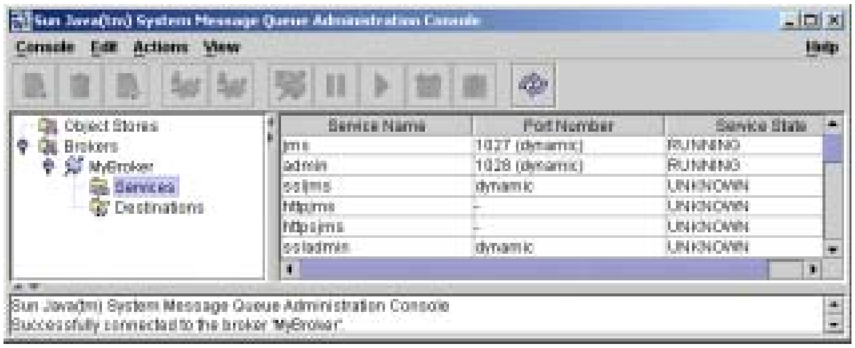
A broker is distinguished by the connection services it provides and the physical destinations it supports.

➤ To View Available Connection Services

1. Select Services under the broker's name in the Administration Console window's navigation pane.

   A list of the available services will appear in the result pane (see Figure 2-6), showing the name, port number, and current state of each service.

   Figure 2-6 Viewing Connection Services

Service Name	Port Number	Service State
jms	1027 (dynamic)	RUNNING
admin	1028 (dynamic)	RUNNING
ssljms	dynamic	UNKNOWN
httpjms	-	UNKNOWN
httpsjms	-	UNKNOWN
ssladmin	dynamic	UNKNOWN

   ```
   Sun Java(tm) System Message Queue Administration Console
   Successfully connected to the broker "MyBroker".
   ```

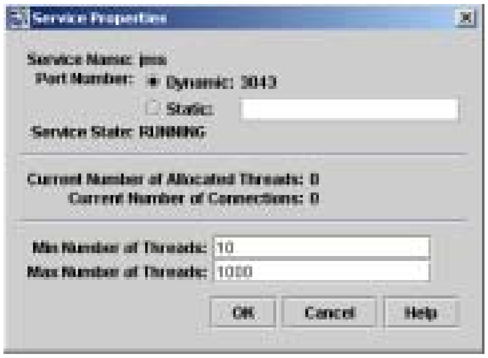
2. Select a service by clicking on its name in the result pane.

   For this exercise, select the name `jms`.

3. Choose Properties from the Actions menu.

   The Service Properties dialog box (Figure 2-7) will appear. You can use this dialog box to assign the service a static port number and to change the minimum and maximum number of threads allocated for it.

   Figure 2-7 Service Properties Dialog Box

   Service Name: jms
   Port Number: [✓ Dynamic: 3043] Static:
   Service State: RUNNING

   Current Number of Allocated Threads: 0
   Current Number of Connections: 0

   Min Number of Threads: 10
   Max Number of Threads: 1000

For this exercise, do not change any of the connection service’s properties.

4. Click OK to accept the new property values and dismiss the dialog box.

The Actions menu also contains commands for pausing and resuming a service. If you select the admin service and pull down the Actions menu, however, you will see that the Pause Service command is disabled. This is because the admin service is the Administration Console’s link to the broker: if you paused it, you would no longer be able to access the broker.

Working With Physical Destinations

A *physical destination* is a location on a message broker where messages received from a message producer are held for later delivery to one or more message consumers. Destinations are of two kinds, depending on the *messaging domain* in use: *queues* (point-to-point domain) and *topics* (publish / subscribe domain). See the *Message Queue Technical Overview* for further discussion of messaging domains and the destinations associated with them.

Creating a Physical Destination

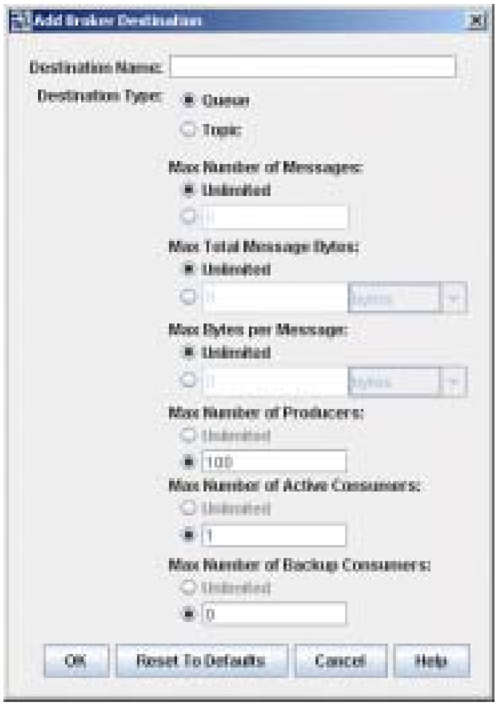
By default, message brokers are configured to create new physical destinations automatically whenever a message producer or consumer attempts to access a nonexistent destination. Such *auto-created destinations* are convenient to use while testing client code in a software development environment. In a production setting, however, it is advisable to disable the automatic creation of destinations and instead require all destinations to be created explicitly by an administrator. The following procedure shows how to add such an *admin-created destination* to a broker.

➤ **To Add a Physical Destination to a Broker**

1. Click on the Destinations item under the broker’s name in the Administration Console window’s navigation pane and choose Add Broker Destination from the Actions menu.

   Alternatively, you can right-click on Destinations and choose Add Broker Destination from the pop-up context menu. In either case, the Add Broker Destination dialog box (Figure 2-8) will appear.

Figure 2-8 Add Broker Destination Dialog Box



Add Broker Destination

Destination Name:

Destination Type: [✓ Queue] Topic

Max Number of Messages: [✓ Unlimited] 0

Max Total Message Bytes: [✓ Unlimited] 0 bytes

Max Bytes per Message: [✓ Unlimited] 0 bytes

Max Number of Producers: Unlimited [✓ 100]

Max Number of Active Consumers: Unlimited [✓ 1]

Max Number of Backup Consumers: Unlimited [✓ 0]

2. Enter a name for the physical destination in the Destination Name field.

Note the name that you assign to the destination; you will need it later when you create an administered object corresponding to this physical destination.

For this exercise, type in the name `MyQueueDest`.

3. Select the Queue or Topic radio button to specify the type of destination to create.

For this exercise, select Queue if it is not already selected.

4. Click OK to add the physical destination and dismiss the dialog box.

The new destination will appear in the result pane.

Viewing Physical Destination Properties

You can use the Properties command on the Administration Console's Actions menu to view or modify the properties of a physical destination.

➤ To View or Modify the Properties of a Physical Destination

1. Select Destinations under the broker's name in the Administration Console window's navigation pane.

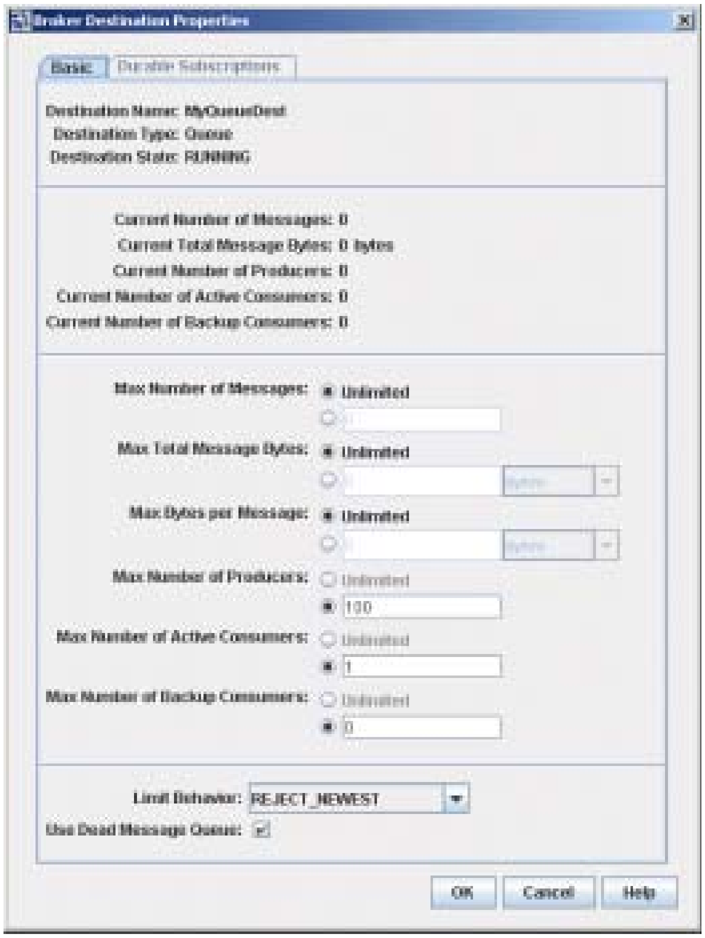
   A list of the available physical destinations will appear in the result pane, showing the name, type, and current state of each destination.

2. Select a physical destination by clicking on its name in the result pane.

3. Choose Properties from the Actions menu.

   The Broker Destination Properties dialog box (Figure 2-9) will appear, showing current status and configuration information about the selected physical destination. You can use this dialog box to change various configuration properties, such as the maximum number of messages, producers, and consumers that the destination can accommodate.

Figure 2-9 Broker Destination Properties Dialog Box



Destination Name: MyQueueDest
Destination Type: Queue
Destination State: RUNNING

Current Number of Messages: 0
Current Total Message Bytes: 0 bytes
Current Number of Producers: 0
Current Number of Active Consumers: 0
Current Number of Backup Consumers: 0

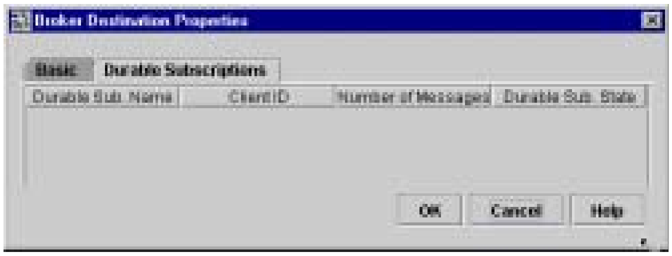
Max Number of Messages: [✓ Unlimited]
Max Total Message Bytes: [✓ Unlimited]
Max Bytes per Message: [✓ Unlimited]
Max Number of Producers: Unlimited [✓ 100]
Max Number of Active Consumers: Unlimited [✓ 1]
Max Number of Backup Consumers: Unlimited [✓ 0]

Limit Behavior: REJECT_NEWEST
Use Dead Message Queue: ✓

For this exercise, do not change any of the destination's properties.

For topic destinations, the Broker Destination Properties dialog box contains an additional tab, Durable Subscriptions. Clicking on this tab displays the Durable Subscriptions panel (Figure 2-10), listing information about all durable subscriptions currently associated with the given topic.

Figure 2-10 Durable Subscriptions Panel



You can use the Durable Subscriptions panel's Purge and Delete buttons to

- Purge all pending messages associated with a durable subscription
- Remove a durable subscription from the topic

The Durable Subscriptions tab is disabled for queue destinations.

4. Click OK to accept the new property values and dismiss the dialog box.

Purging Messages From a Physical Destination

Purging messages from a physical destination removes all pending messages associated with the destination, leaving the destination empty.

➤ To Purge Messages From a Physical Destination

1. Select Destinations under the broker's name in the Administration Console window's navigation pane.

   A list of the available physical destinations will appear in the result pane, showing the name, type, and current state of each destination.

2. Select a destination by clicking on its name in the result pane.
3. Choose Purge Messages from the Actions menu.

   A confirmation dialog box will appear, asking you to confirm that you wish to proceed with the operation.

4. Click Yes to confirm the operation and dismiss the confirmation dialog.

Deleting a Physical Destination

Deleting a destination purges all of its messages and then destroys the destination itself, removing it permanently from the broker to which it belongs.

➤ **To Delete a Physical Destination**

1. Select Destinations under the broker's name in the Administration Console window's navigation pane.

   A list of the available destinations will appear in the result pane, showing the name, type, and current state of each destination.

2. Select a destination by clicking on its name in the result pane.

3. Choose Delete from the Edit menu.

   A confirmation dialog box will appear, asking you to confirm that you wish to proceed with the operation.

4. Click Yes to confirm the operation and dismiss the confirmation dialog.

   For this exercise, do not delete the destination `MyQueueDest` that you created earlier; instead, click No to dismiss the confirmation dialog without performing the delete operation.

Working With Object Stores

An *object store* is used to store Message Queue *administered objects*, which encapsulate implementation and configuration information specific to a particular Message Queue provider. An object store can be either a Lightweight Directory Access Protocol (LDAP) directory server or a directory in the local file system.

Although it is possible to instantiate and configure administered objects directly from within a client application's code, it is generally preferable to have an administrator create and configure these objects and store them in an object store, where client applications can access them using the Java Naming and Directory Interface (JNDI). This allows the client code itself to remain provider-independent.

Adding an Object Store

Although the Administration Console allows you to *manage* an object store, you cannot use it to *create* one; the LDAP server or file-system directory that will serve as the object store must already exist ahead of time. You can then add this existing object store to the Administration Console, creating a reference to it that you can use to operate on it from within the Console.

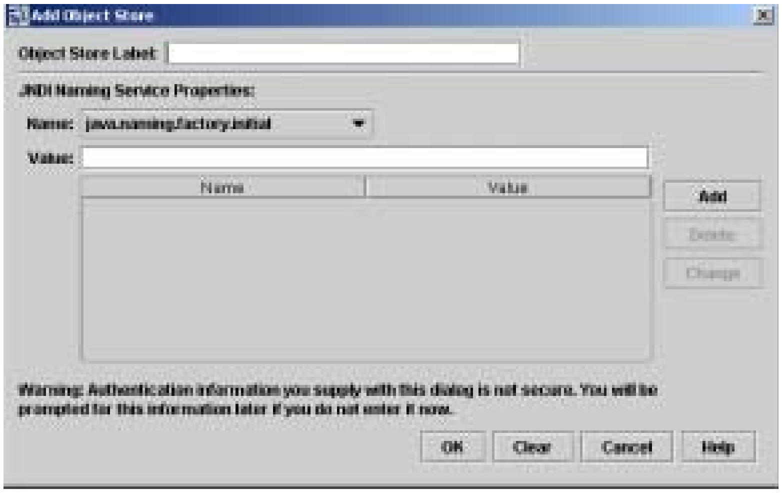
NOTE The sample application used in this chapter assumes that the object store is held in a directory named `Temp` on the C drive. If you do not already have a folder named `Temp` on your C drive, create one before proceeding with the following exercise. (On non-Windows platforms, you can use the `/tmp` directory, which should already exist.)

➤ To Add an Object Store to the Administration Console

1. Click on the Object Stores item in the Administration Console window's navigation pane and choose Add Object Store from the Actions menu.

   Alternatively, you can right-click on Object Stores and choose Add Object Store from the pop-up context menu. In either case, the Add Object Store dialog box (Figure 2-11) will appear.

Figure 2-11 Add Object Store Dialog Box



Warning: Authentication information you supply with this dialog is not secure. You will be prompted for this information later if you do not enter it now.

2. Enter a name for the object store in the Object Store Label field.

   This provides a label that identifies the object store in the Administration Console.

   For this exercise, type in the name `MyObjectStore`.

3. Enter the JNDI attribute values to be used for looking up administered objects:

   a. Select the name of the attribute you wish to specify from the Name pull-down menu.

   b. Type the value of the attribute into the Value field.

   c. Click the Add button to add the specified attribute value.

      The property and its value will appear in the property summary pane.

   Repeat steps a to c for as many attributes as you need to set.

   For this exercise, set the `java.naming.factory.initial` attribute to

   ```
   com.sun.jndi.fscontext.RefFSContextFactory
   ```

   and the `java.naming.provider.url` attribute to

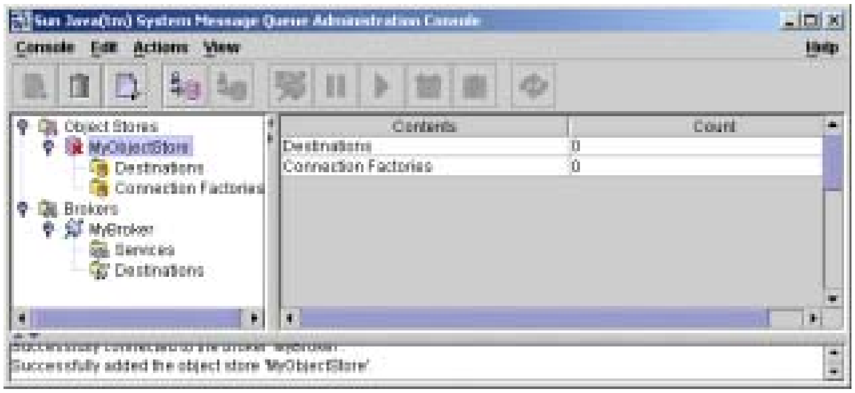
   ```
   file:///C:/Temp
   ```

   (or `file:///tmp` on the Solaris or Linux platforms). These are the only attributes you need to set for a file-system object store; see "LDAP Server Object Stores" on page 162 for information on the attribute values needed for an LDAP store.

4. Click OK to add the object store and dismiss the dialog box.

   The new object store will appear under Object Stores in the navigation pane, as shown in Figure 2-12. The red X over the object store's icon indicates that it is not currently connected to the Administration Console.

Figure 2-12 Object Store Displayed in Administration Console Window



Contents	Count
Destinations	0
Connection Factories	0

```
Successfully connected to the broker 'MyBroker'
Successfully added the object store 'MyObjectStore'
```

When you click on the object store in the navigation pane, its contents are listed in the result pane. Since you have not yet added any administered objects to the object store, the Count column shows 0 for both destinations and connection factories.

Once you have added an object store, you can use the Properties command on the Actions menu (or the pop-up context menu) to display an Object Store Properties dialog box, similar to the Add Object Store dialog shown in Figure 2-11, to view or modify any of its properties.

Connecting to an Object Store

Now that you have added an object store to the Administration Console, you must connect to it in order to add administered objects to it.

➤ To Connect to an Object Store

1. Click on the object store's name in the Administration Console window's navigation pane and choose Connect to Object Store from the Actions menu.

   Alternatively, you can right-click on the object store's name and choose Connect to Object Store from the pop-up context menu. In either case, the red X will disappear from the object store's icon, indicating that it is now connected to the Administration Console.

Working With Administered Objects

Once you have connected an object store to the Administration Console, you can proceed to add administered objects (connection factories and destinations) to it. This section describes how.

NOTE The Administration Console displays only Message Queue administered objects. If an object store contains a non–Message Queue object with the same lookup name as an administered object that you want to add, you will receive an error when you attempt the add operation.

Adding a Connection Factory

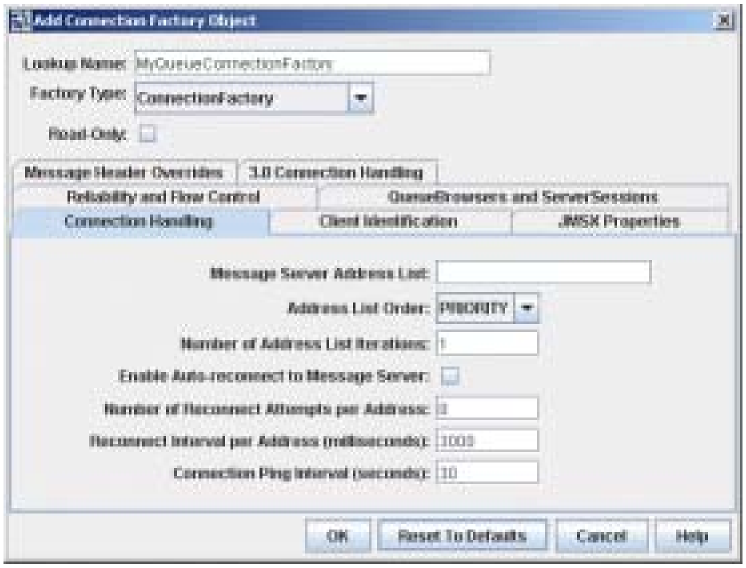
Connection factories are used by client applications to cretae connections to a broker. By configuring a connection factory, you can control the properties of the connections it creates.

➤ **To Add a Connection Factory to an Object Store**

1. Make sure the object store is connected to the Administration Console (see "Connecting to an Object Store" on page 54).
2. Click on the Connection Factories item under the object store's name in the Administration Console window's navigation pane and choose Add Connection Factory Object from the Actions menu.

   Alternatively, you can right-click on Connection Factories and choose Add Connection Factory Object from the pop-up context menu. In either case, the Add Connection Factory Object dialog box (Figure 2-13) will appear.

Figure 2-13 Add Connection Factory Object Dialog Box



Lookup Name: MyQueueConnectionFactory
Factory Type: ConnectionFactory
Read-Only: ☐

Message Server Address List:
Address List Order: PRIORITY
Number of Address List Iterations: 1
Enable Auto-reconnect to Message Server: ☐
Number of Reconnect Attempts per Address: 3
Reconnect Interval per Address (milliseconds): 3000
Connection Ping Interval (seconds): 10

3. Enter a name for the connection factory in the Lookup Name field.

   This is the name that client applications will use when looking up the connection factory with JNDI.

   For this exercise, type in the name `MyQueueConnectionFactory`.

4. Choose the type of connection factory you wish to create from the Factory Type pull-down menu.

   For this exercise, choose QueueConnectionFactory.

5. Click the Connection Handling tab.

   The Connection Handling panel will appear, as shown in Figure 2-13.

6. Fill in the Message Server Address List field with the address(es) of the broker(s) to which this connection factory will create connections.

   The address list may consist of a single broker or (in the case of a broker cluster) multiple brokers. For each broker, it specifies information such as the broker's connection service, host name, and port number. The exact nature and syntax of the information to be specified varies, depending on the connection service to be used; see "Connection Handling" on page 318 for specifics.

   For this exercise, there is no need to type anything into the Message Server Address List field, since the sample application `HelloWorldMessageJNDI` expects the connection factory to use the standard address list attributes to which it is automatically configured by default (connection service `jms`, host name `localhost`, and port number `7676`).

7. Configure any other attributes of the connection factory as needed.

   The Add Connection Factory Object dialog box contains a number of other panels besides Connection Handling, which can be used to configure various attributes for a connection factory.

   For this exercise, do not change any of the other attribute settings. You may find it instructive, however, to click through the other tabs to get an idea of the kinds of configuration information that can be specified. Use the Help button to learn more about the contents of these other configuration panels.

8. If appropriate, click the Read-Only checkbox.

   This locks the connection factory object's configuration attributes to the values they were given at creation time. A read-only administered object's attributes cannot be overridden, whether programmatically from client code or administratively from the command line.

   For this exercise, do not check Read-Only.

9. Click OK to create the connection factory, add it to the object store, and dismiss the dialog box.

   The new connection factory will appear in the result pane.

Adding a Destination

A *destination* administered object represents a physical destination on a broker, enabling clients to send messages to that physical destination independently of provider-specific configurations and naming syntax. When a client sends a message addressed via the administered object, the broker will deliver the message to the corresponding physical destination, if it exists. If no such physical destination exists, the broker will create one automatically if auto-creation is enabled, as described under "Creating a Physical Destination" on page 46, and deliver the message to it; otherwise, it will generate an error signaling that the message cannot be delivered.

The following procedure describes how to add a destination administered object to the object store corresponding to an existing physical destination:

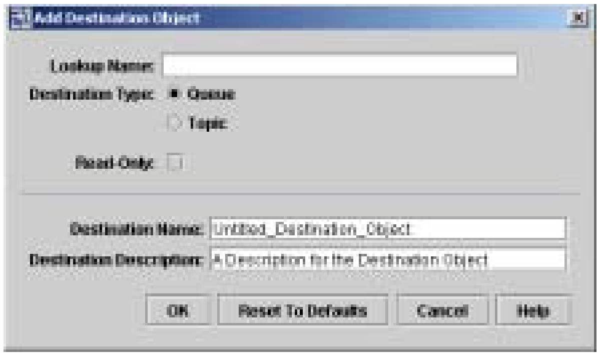
➤ To Add a Destination to an Object Store

1. Make sure the object store is connected to the Administration Console (see "Connecting to an Object Store" on page 54).

2. Click on the Destinations item under the object store's name in the Administration Console window's navigation pane and choose Add Destination Object from the Actions menu.

   Alternatively, you can right-click on Destinations and choose Add Destination Object from the pop-up context menu. In either case, the Add Destination Object dialog box (Figure 2-14) will appear.

Figure 2-14 Add Destination Object Dialog Box


Add Destination Object

Lookup Name:

Destination Type: ● Queue ○ Topic

Read-Only: ☐

Destination Name: Untitled_Destination_Object

Destination Description: A Description for the Destination Object

3. Enter a name for the destination administered object in the Lookup Name field.

   This is the name that client applications will use when looking up the destination with JNDI.

   For this exercise, type in the name `MyQueue`.

4. Select the Queue or Topic radio button to specify the type of destination object to create.

   For this exercise, select Queue if it is not already selected.

5. Enter the name of the corresponding physical destination in the Destination Name field.

   This is the name you specified when you added the physical destination to the broker (see "Working With Physical Destinations" on page 46).

   For this exercise, type in the name `MyQueueDest`.

6. Optionally, enter a brief description of the destination in the Destination Description field.

   The contents of this field are intended strictly for human consumption and have no effect on client operations.

   For this exercise, you can either delete the contents of the Destination Description field or type in some descriptive text such as

   ```
   Example destination for MQ Admin Guide tutorial
   ```

7. If appropriate, click the Read-Only checkbox.

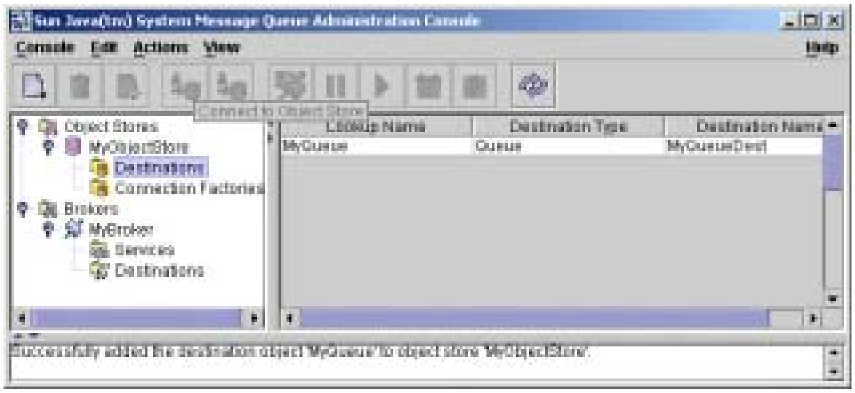
   This locks the destination object's configuration attributes to the values they were given at creation time. A read-only administered object's attributes cannot be overridden, whether programmatically from client code or administratively from the command line.

   For this exercise, do not check Read-Only.

8. Click OK to create the destination object, add it to the object store, and dismiss the dialog box.

   The new destination object will appear in the result pane, as shown in Figure 2-15.

Figure 2-15 Destination Object Displayed in Administration Console Window



Lookup Name	Destination Type	Destination Name
MyQueue	Queue	MyQueueDest

Successfully added the destination object 'MyQueue' to object store 'MyObjectStore'.

Viewing Administered Object Properties

You can use the Properties command on the Administration Console's Actions menu to view or modify the properties of an administered object.

➤ To View or Modify the Properties of an Administered Object

1. Select Connection Factories or Destinations under the object store's name in the Administration Console window's navigation pane.

   A list of the available connection factory or destination administered objects will appear in the result pane, showing the lookup name and type of each (as well as the destination name in the case of destination administered objects).

2. Select an administered object by clicking on its name in the result pane.

3. Choose Properties from the Actions menu.

   The Connection Factory Object Properties or Destination Object Properties dialog box will appear, similar to the Add Connection Factory Object (Figure 2-13 on page 56) or Add Destination Object (Figure 2-14 on page 58) dialog. You can use this dialog box to change the selected object's configuration attributes. Note, however, that you cannot change the object's lookup name; the only way to do this is the delete the object and then add a new administered object with the desired lookup name.

4. Click OK to accept the new attribute values and dismiss the dialog box.

Deleting an Administered Object

Deleting an administered object removes it permanently from the object store to which it belongs.

➤ To Delete an Administered Object

1. Select Connection Factories or Destinations under the object store's name in the Administration Console window's navigation pane.

   A list of the available connection factory or destination administered objects will appear in the result pane, showing the lookup name and type of each (as well as the destination name in the case of destination administered objects).

2. Select an administered object by clicking on its name in the result pane.

3. Choose Delete from the Edit menu.

   A confirmation dialog box will appear, asking you to confirm that you wish to proceed with the operation.

4. Click Yes to confirm the operation and dismiss the confirmation dialog.

   For this exercise, do not delete the administered objects `MyQueue` or `MyQueueConnectionFactory` that you created earlier; instead, click No to dismiss the confirmation dialog without performing the delete operation.

Running the Sample Application

The sample application `HelloWorldMessageJNDI` is provided for use with this tutorial. It uses the physical destination and administered objects that you created:

- A queue physical destination named `MyQueueDest`
- A queue connection factory administered object with JNDI lookup name `MyQueueConnectionFactory`
- A queue administered object with JNDI lookup name `MyQueue`

The code creates a simple queue sender and receiver, and sends and receives a `Hello World` message.

Before running the application, open the source file `HelloWorldMessageJNDI.java` and read through the code. The program is short and amply documented; you should have little trouble understanding how it works.

➤ To Run the Sample Application

1. Make the directory containing the `HelloWorldmessageJNDI` application your current directory, using one of the following commands (depending on the platform you're using):

   - On Solaris:

     ```
     cd /usr/demo/imq/helloworld/helloworldmessagejndi
     ```

   - On Linux:

     ```
     cd /opt/sun/mq/examples/helloworld/helloworldmessagejndi
     ```

- On Windows:

```
cd IMQ_HOME\demo\helloworld\helloworldmessagejndi
```

You should find the file `HelloWorldMessageJNDI.class` present. (If you make changes to the application, you must recompile it using the procedure for compiling a client application given in the *Message Queue Developer's Guide for Java Clients*.)

2. Set the `CLASSPATH` variable to include the current directory containing the file `HelloWorldMessageJNDI.class`, as well as the following `.jar` files that are included in the Message Queue product:

```
jms.jar
imq.jar
jndi.jar
fscontext.jar
```

See the *Message Queue Developer's Guide for Java Clients* for information on setting the `CLASSPATH` variable.

NOTE The file `jndi.jar` is bundled with JDK 1.4. You need not add this file to your `CLASSPATH` unless you are using an earlier version of the JDK.

3. Run the `HelloWorldMessageJNDI` application by executing one of the following commands (depending on the platform you're using):

- On Solaris or Linux:

```
% java HelloWorldMessageJNDI file:///tmp
```

- On Windows:

```
java HelloWorldMessageJNDI
```

If the application runs successfully, you should see the output shown in Code Example 2-1.

Code Example 2-1 Output From Sample Application

```
java HelloWorldMessageJNDI
Using file:///C:/Temp for Context.PROVIDER_URL


Looking up Queue Connection Factory object with lookup name: MyQueueConnectionFactory
Queue Connection Factory object found.
Looking up Queue object with lookup name: MyQueue
Queue object found.


Creating connection to broker.
Connection to broker created.

Publishing a message to Queue: MyQueueDest
Received the following message: Hello World
```

Part II

Administration Tasks

Chapter 3

Starting Brokers and Clients

After installing Sun Java System Message Queue and performing some preparatory steps, you can begin starting brokers and clients. A broker's configuration is governed by a set of configuration files, which can be overridden by command line options passed to the Broker utility (`imqbrokerd`); see Chapter 4, "Configuring a Broker," for more information.

This chapter contains the following sections:

Preparing System Resources

Before starting a broker, there are two preliminary system-level tasks to perform: synchronizing system clocks and (on the Solaris or Linux platform) setting the file descriptor limit. The following sections describe these tasks.

Synchronizing System Clocks

Before starting any brokers or clients, it is important to synchronize the clocks on all hosts that will interact with the Message Queue system. Synchronization is particularly crucial if you are using message expiration (time-to-live). Time stamps from clocks that are not synchronized could prevent message expiration from working as expected and prevent the delivery of messages. Synchronization is also crucial for broker clusters.

Configure your systems to run a time synchronization protocol, such as Simple Network Time Protocol (SNTP). Time synchronization is generally supported by the `xntpd` daemon on Solaris and Linux, and by the `W32Time` service on Windows. (See your operating system documentation for information about configuring this service.) After the broker is running, avoid setting the system clock backward.

Setting the File Descriptor Limit

On the Solaris and Linux platforms, the shell in which a client or broker is running places a soft limit on the number of file descriptors that a process can use. In Message Queue, each connection a client makes, or a broker accepts, uses one of these file descriptors. Each physical destination that has persistent messages also uses a file descriptor.

As a result, the file descriptor limit constrains the number of connections a broker or client can have. By default, the maximum is 256 connections on Solaris or 1024 on Linux. (In practice, the connection limit is actually lower than this because of the use of file descriptors for persistence.) If you need more connections than this, you must raise the file descriptor limit in each shell in which a client or broker will be executing. For information on how to do this, see the `ulimit` man page.

Starting Brokers

You can start a broker either interactively, using the Message Queue command line utilities or the Windows Start menu, or by arranging for it to start automatically at system startup. The following sections describe how.

Starting Brokers Interactively

You can start a broker interactively from the command line, using the Broker utility (`imqbrokerd`). (Alternatively, on Windows, you can start a broker from the Start menu.) You cannot use the Administration Console (`imqadmin`) or the Command utility (`imqcmd`) to start a broker; the broker must already be running before you can use these tools.

On the Solaris and Linux platforms, a broker instance must always be started by the same user who initially started it. Each broker instance has its own set of configuration properties and file-based message store. When the broker instance first starts, Message Queue uses the user's file creation mode mask (`umask`) to set permissions on directories containing the configuration information and persistent data for that broker instance.

A broker instance has the instance name `imqbroker` by default. To start a broker from the command line with this name and the default configuration, simply use the command

```
imqbrokerd
```

This starts a broker instance named `imqbroker` on the local machine, with the Port Mapper at the default port of `7676` (see "Port Mapper" on page 77).

To specify an instance name other than the default, use the `-name` option to the `imqbrokerd` command. The following command starts a broker with the instance name `myBroker`:

```
imqbrokerd -name myBroker
```

Other options are available on the `imqbrokerd` command line to control various aspects of the broker's operation. The following example uses the `-tty` option to send errors and warnings to the command window (standard output):

```
imqbrokerd -name myBroker -tty
```

You can also use the `-D` option on the command line to override the values of properties specified in the broker's instance configuration file (`config.properties`). This example sets the `imq.jms.max_threads` property, raising the maximum number of threads available to the `jms` connection service to 2000:

```
imqbrokerd -name myBroker -Dimq.jms.max_threads=2000
```

See "Broker Utility" on page 266 for complete information on the syntax, subcommands, and options of the `imqbrokerd` command. For a quick summary of this information, enter the command

```
imqbrokerd -help
```

NOTE If you have a Sun Java System Message Queue Platform Edition license, you can use the `imqbrokerd` command's `-license` option to activate a trial Enterprise Edition license, allowing you to try Enterprise Edition features for 90 days. Specify `try` as the license name:

```
imqbrokerd -license try
```

You must use this option each time you start a broker; otherwise the broker will default to the standard Platform Edition license.

Starting Brokers Automatically

Instead of starting a broker explicitly from the command line, you can set it up to start automatically at system startup. How you do this depends on the platform (Solaris, Linux, or Windows) on which you are running the broker.

Automatic Startup on Solaris and Linux

On Solaris and Linux systems, scripts that enable automatic startup are placed in the `/etc/rc*` directory tree during Message Queue installation. To enable the use of these scripts, you must edit the configuration file `/etc/imq/imqbrokerd.conf` (Solaris) or `/etc/opt/sun/mq/imqbrokerd.conf` (Linux) as follows:

- To start the broker automatically at system startup, set the `AUTOSTART` property to `YES`.
- To have the broker restart automatically after an abnormal exit, set the `RESTART` property to `YES`.
- To set startup command line arguments for the broker, specify one or more values for the `ARGS` property.

Automatic Startup on Windows

To start a broker automatically at Windows system startup, you must define the broker as a Windows service. The broker will then start at system startup time and run in the background until system shutdown. Consequently, you do not use the `imqbrokerd` command to start the broker unless you want to start an additional instance.

A system can have no more than one broker running as a Windows service. Task Manager lists such a broker as two executable processes:

- The native Windows service wrapper, `imqbrokersvc.exe`
- The Java runtime that is running the broker

You can install a broker as a service when you install Message Queue on a Windows system. After installation, you can use the Service Administrator utility (`imqsvcadmin`) to perform the following operations:

- Add a broker as a Windows service
- Determine the startup options for the broker service
- Remove a broker that is running as a Windows service

To pass startup options to the broker, use the `-args` argument to the `imqsvcadmin` command. This works the same way as the `imqbrokerd` command's `-D` option, as described under "Starting Brokers" on page 66. Use the Command utility (`imqcmd`) to control broker operations as usual.

See "Service Administrator Utility" on page 283 for complete information on the syntax, subcommands, and options of the `imqsvcadmin` command.

Reconfiguring the Broker Service

The procedure for reconfiguring a broker installed as a Windows service is as follows:

➤ To Reconfigure a Broker Running as a Windows Service

1. Stop the service.

   a. From the Settings submenu of the Windows Start menu, choose Control Panel.

   b. Open the Administrative Tools control panel.

   c. Run the Services tool by selecting its icon and choosing Open from the File menu or the pop-up context menu, or simply by double-clicking the icon.

d. Under Services (Local), select the Message Queue Broker service and choose Properties from the Action menu.

   Alternatively, you can right-click on Message Queue Broker and choose Properties from the pop-up context menu, or simply double-click on Message Queue Broker. In either case, the Message Queue Broker Properties dialog box will appear.

e. Under the General tab in the Properties dialog, click Stop to stop the broker service.

2. Remove the service.

   On the command line, enter the command

   ```
   imqsvcadmin remove
   ```

3. Reinstall the service, specifying different broker startup options with the `-args` option or different Java version arguments with the `-vmargs` option.

   For example, to change the service's host name and port number to `broker1` and `7878`, you could use the command

   ```
   imqsvcadmin install -args "-name broker1 -port 7878"
   ```

Using an Alternative Java Runtime

You can use either `imqsvcadmin` command's `-javahome` or `-jrehome` option to specify the location of an alternative Java runtime. (You can also specify these options in the Start Parameters field under the General tab in the service's Properties dialog window.)

NOTE The Start Parameters field treats the backslash character (\) as an escape character, so you must type it twice when using it as a path delimiter: for example,

```
-javahome c:\\j2sdk1.4.0
```

Displaying Broker Service Startup Options

To determine the startup options for the broker service, use the `query` option to the `imqsvcadmin` command, as shown in Code Example 3-1.

Code Example 3-1 Displaying Broker Service Startup Options

```
imqsvcadmin query

Service Message Queue Broker is installed.
Display Name: Message Queue Broker
Start Type: Automatic
Binary location: C:\Sun\MessageQueue\bin\imqbrokersvc.exe
JavaHome: c:\j2sdk1.4.0
Broker Args: -name broker1 -port 7878
```

Troubleshooting Service Startup Problems

If you get an error when you try to start a broker as a Windows service, you can view error events that were logged:

➤ To See Logged Service Error Events

1. Open the Windows Administrative Tools control panel.
2. Start the Event Viewer tool.
3. Select the Application event log.
4. Choose Refresh from the Action menu to display any error events.

Removing Brokers

The procedure for removing a broker again varies from one platform to another, as described in the following sections.

Removing a Broker on Solaris or Linux

To remove a broker instance on the Solaris or Linux platform, use the `imqbrokerd` command with the `-remove` option. The command format is as follows:

```
imqbrokerd [options…] -remove instance
```

For example, if the name of the broker is `myBroker`, the command would be

```
imqbrokerd -name myBroker -remove instance
```

The command deletes the entire instance directory for the specified broker.

If the broker is set up to start automatically at system startup, edit the configuration file `/etc/imq/imqbrokerd.conf` (Solaris) or `/etc/opt/sun/mq/imqbrokerd.conf` (Linux) and set the `AUTOSTART` property to `NO`.

See "Broker Utility" on page 266 for complete information on the syntax, subcommands, and options of the `imqbrokerd` command. For a quick summary of this information, enter the command

Removing a Windows Broker Service

To remove a broker that is running as a Windows service, use the command

```
imqcmd shutdown bkr
```

to shut down the broker, followed by

```
imqsvcadmin remove
```

to remove the service.

Alternatively, you can use the Windows `Services` tool, reached via the `Administrative Tools` control panel, to stop and remove the broker service.

Restart your computer after removing the broker service.

Starting Clients

Before starting a client application, obtain information from the application developer about how to set up the system. If you are starting Java client applications, you must set the `CLASSPATH` variable appropriately and make sure you have the correct `.jar` files installed. The *Message Queue Developer's Guide for Java Clients* contains information about generic steps for setting up the system, but your developer may have additional information to provide.

To start a Java client application, use the following command line format:

```
java clientAppName
```

To start a C client application, use the format supplied by the application developer.

The application's documentation should provide information on attribute values that the application sets; you may want to override some of these from the command line. You may also want to specify attributes on the command line for any Java client that uses a Java Naming and Directory Interface (JNDI) lookup to find its connection factory. If the lookup returns a connection factory that is older than the application, the connection factory may lack support for more recent attributes. In such cases, Message Queue sets those attributes to default values; if necessary, you can use the command line to override these default values.

To specify attribute values from the command line for a Java application, use the following syntax:

```
java [[-Dattribute=value]...] clientAppName
```

The value for *attribute* must be a connection factory administered object attribute, as described in Chapter 16, "Administered Object Attribute Reference." If there is a space in the value, put quotation marks around the *attribute*=*value* part of the command line.

The following example starts a client application named `MyMQClient`, connecting to a broker on the host `OtherHost` at port `7677`:

```
java -DimqAddressList=mq://OherHost:7677/jms MyMQClient
```

The host name and port specified on the command line override any others set by the application itself.

In some cases, you cannot use the command line to specify attribute values. An administrator can set an administered object to allow read access only, or an application developer can code the client application to do so. Communication with the application developer is necessary to understand the best way to start the client program.

Chapter 4

Configuring a Broker

A broker's configuration is governed by a set of configuration files and by the options passed to the `imqbrokerd` command at startup. This chapter describes the available configuration properties and how to use them to configure a broker.

The chapter contains the following sections:

- "Broker Services" on page 75
- "Setting Broker Properties" on page 89
- "Configuring a Persistent Data Store" on page 93

For full reference information about broker configuration properties, see Chapter 14, "Broker Properties Reference."

Broker Services

Broker configuration properties can be divided into several categories, depending on the services or broker components they affect:

- **Connection services** manage the physical connections between a broker and its clients that provide transport for incoming and outgoing messages.
- **Routing services** route and deliver JMS payload messages, as well as control messages used by the message service to support reliable delivery.
- **Persistence services** manage the writing and retrieval of data to and from persistent storage.
- **Security services** authenticate users connecting to the broker and authorize their actions.
- **Monitoring services** generate metric and diagnostic information about the broker's performance.

The following sections describe each of these services and the properties you use to customize them for your particular needs.

Connection Services

Message brokers can offer various *connection services* supporting both application and administrative clients, using a variety of transport protocols. Broker configuration properties related to connection services are listed under "Connection Properties" on page 285.

Table 4-1 shows the available connection services., which are distinguished by two characteristics:

- The *service type* specifies whether the service provides JMS message delivery (`NORMAL`) or Message Queue administration services (`ADMIN`).
- The *protocol type* specifies the underlying transport protocol.

Table 4-1 Message Queue Connection Services

Service Name	Service Type	Protocol Type
`jms`	`NORMAL`	TCP
`ssljms` (Enterprise Edition)	`NORMAL`	TLS (SSL-based security)
`httpjms` (Enterprise Edition)	`NORMAL`	HTTP
`httpsjms` (Enterprise Edition)	`NORMAL`	HTTPS (SSL-based security)
`admin`	`ADMIN`	TCP
`ssladmin`	`ADMIN`	TLS (SSL-based security)

By setting a broker's `imq.service.activelist` property, you can configure it to run any or all of these connection services. The value of this property is a list of connection services to be activated when the broker is started up; if the property is not specified explicitly, the `jms` and `admin` services will be activated by default.

Each connection service also supports specific authentication and authorization features; see "Security Services" on page 83 for more information.

Port Mapper

Each connection service is available at a particular port, specified by host name (or IP address) and port number. You can explicitly specify a static port number for a service or have the broker's *Port Mapper* assign one dynamically. The Port Mapper itself resides at the broker's *primary port*, which is normally located at the standard port number `7676`. (If necessary, you can use the broker configuration property `imq.portmapper.port` to override this with a different port number.) By default, each connection service registers itself with the Port Mapper when it starts up. When a client creates a connection to the broker, the Message Queue client runtime first contacts the Port Mapper, requesting a port number for the desired connection service.

Alternatively, you can override the Port Mapper and explicitly assign a static port number to a connection service, using the `imq.`*serviceName*.*protocolType*`.port` configuration property (where *serviceName* and *protocolType* identify the specific connection service, as shown in Table 4-1). (Only the `jms`, `ssljms`, `admin`, and `ssladmin` connection services can be configured this way; the `httpjms` and `httpsjms` services use different configuration properties, described in Appendix C, "HTTP / HTTPS Support.") Static ports are generally used only in special situations, however, such as in making connections through a firewall, and are not recommended for general use.

NOTE In cases where two or more hosts are available (such as when more than one network card is installed in a computer), you can use broker properties to specify which host the connection services should bind to. The `imq.hostname` property designates a single default host for all connection services; this can then be overridden, if necessary, with `imq.`*serviceName*.*protocolType*`.hostname` (for the `jms`, `ssljms`, `admin`, or `ssladmin` service) or `imq.portmapper.hostname` (for the Port Mapper itself).

When multiple Port Manager requests are received concurrently, they are stored in an operating system backlog while awaiting action. The `imq.portmapper.backlog` property specifies the maximum number of such backlogged requests. When this limit is exceeded, any further requests will be rejected until the backlog is reduced.

Thread Pool Management

Each connection service is multithreaded, supporting multiple connections. The threads needed for these connections are maintained by the broker in a separate *thread pool* for each service. As threads are needed by a connection, they are added to the thread pool for the service supporting that connection.

The threading model you choose specifies whether threads are dedicated to a single connection or shared by multiple connections:

- In the *dedicated model,* each connection to the broker requires two threads: one for incoming and one for outgoing messages. This limits the number of connections that can be supported, but provides higher performance.
- In the *shared model,* connections are processed by a shared thread when sending or receiving messages. Because each connection does not require dedicated threads, this model increases the number of possible connections, but at the cost of lower performance because of the additional overhead needed for thread management.

The broker's `imq.`*serviceName*`.threadpool_model` property specifies which of the two models to use for a given connection service. This property takes either of two string values: `dedicated` or `shared`. If you don't set the property explicitly, `dedicated` is assumed by default.

You can also set the broker properties `imq.`*serviceName*`.min_threads` and `imq.`*serviceName*. `max_threads` to specify a minimum and maximum number of threads in a service's thread pool. When the number of available threads exceeds the specified minimum threshold, Message Queue will shut down threads as they become free until the minimum is reached again, thereby saving on memory resources. Under heavy loads, the number of threads might increase until the pool's maximum number is reached; at this point, new connections are rejected until a thread becomes available.

The shared threading model uses *distributor threads* to assign threads to active connections. The broker property `imq.shared.connectionMonitor_limit` specifies the maximum number of connections that can be monitored by a single distributor thread. The smaller the value of this property, the faster threads can be assigned to connections. The `imq.ping.interval` property specifies the time interval, in seconds, at which the broker will periodically test ("ping") a connection to verify that it is still active, allowing connection failures to be detected preemptively before an attempted message transmission fails.

Routing Services

Once clients are connected to the broker, the routing and delivery of messages can proceed. In this phase, the broker is responsible for creating and managing different types of physical destinations, ensuring a smooth flow of messages, and using resources efficiently. You can use the broker configuration properties described under "Routing Properties" on page 287 to manage these tasks in a way that suits your application's needs.

The performance and stability of a broker depend on the system resources (such as memory) available and how efficiently they are utilized. You can set configuration properties to prevent the broker from becoming overwhelmed by incoming messages or running out of memory. These properties function at three different levels to keep the message service operating as resources become scarce:

- **Systemwide message limits** apply collectively to all physical destinations on the system. These include the maximum number of messages held by a broker (`imq.system.max_count`) and the maximum total number of bytes occupied by such messages (`imq.system.max_size`). If either of these limits is reached, the broker will reject any new messages until the pending messages fall below the limit. There is also a limit on the maximum size of an individual message (`imq.message.max_size`) and a time interval at which expired messages are reclaimed (`imq.message.expiration.interval`).
- **Individual destination limits** regulate the flow of messages to a specific physical destination. The configuration properties controlling these limits are described in Chapter 15, "Physical Destination Property Reference." They include limits on the number and size of messages the destination will hold, the number of message producers and consumers that can be created for it, and the number of messages that can be batched together for delivery to the destination.

  The destination can be configured to respond to memory limits by slowing down the delivery of message by message producers, by rejecting new incoming messages, or by throwing out the oldest or lowest-priority existing messages. Messages deleted from the destination in this way may optionally be moved to the dead message queue rather than discarded outright; the broker property `imq.destination.DMQ.truncateBody` controls whether the entire message body is saved in the dead message queue, or only the header and property data.

  As a convenience during application development and testing, you can configure a message broker to create new physical destinations automatically whenever a message producer or consumer attempts to access a nonexistent destination. The broker properties summarized in Table 14-3 on page 289 parallel the ones just described, but apply to such *auto-created destinations* instead of administratively created ones.
- **System memory thresholds** define levels of memory usage at which the broker takes increasingly serious action to prevent memory overload. Four such usage levels are defined:
  - **Green:** Plenty of memory is available.
  - **Yellow:** Broker memory is beginning to run low.

- **Orange:** The broker is low on memory.
- **Red:** The broker is out of memory.

The memory utilization percentages defining these levels are specified by the broker properties `imq.green.threshold`, `imq.yellow.threshold`, `imq.orange.threshold`, and `imq.red.threshold`, respectively; the default values are 0% for green, 80% for yellow, 90% for orange, and 98% for red.

As memory usage advances from one level to the next, the broker responds progressively, first by swapping messages out of active memory into persistent storage and then by throttling back producers of nonpersistent messages, eventually stopping the flow of messages into the broker. (Both of these measures degrade broker performance.) The throttling back of message production is done by limiting the size of each batch delivered to the number of messages specified by the properties `imq.`*resourceState*`.count`, where *resourceState* is `green`, `yellow`, `orange`, or `red`, respectively.

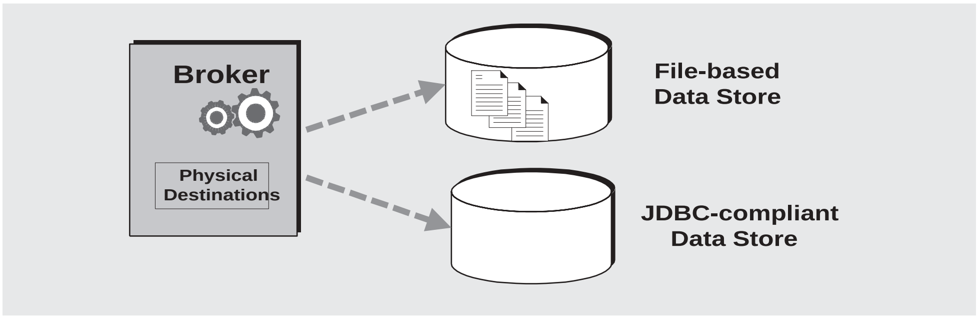
NOTE The triggering of these system memory thresholds is a sign that systemwide and destination message limits are set too high. Because the memory thresholds cannot always catch potential memory overloads in time, you should not rely on them to control memory usage, but rather reconfigure the systemwide and destination limits to optimize memory resources.

Persistence Services

For a broker to recover in case of failure, it needs to re-create the state of its message delivery operations. To do this, the broker must save state information to a *persistent data store*. When the broker restarts, it uses the saved data to re-create destinations and durable subscriptions, recover persistent messages, roll back open transactions, and rebuild its routing table for undelivered messages. It can then resume message delivery.

Message Queue supports both file-based and JDBC-based persistence modules (see Figure 4-1). File-based persistence uses individual files to store persistent data; JDBC-based persistence uses the Java Database Connectivity (JDBC™) interface to connect the broker to a JDBC-compliant data store. While file-based persistence is generally faster than JDBC-based, some users prefer the redundancy and administrative control provided by a JDBC-compliant store. The broker configuration property `imq.persist.store` (see Table 14-4 on page 292) specifies which of the two forms of persistence to use.

Figure 4-1 Persistent Data Storage



File-Based Persistence

By default, Message Queue uses a file-based persistent data store, in which individual files store persistent data such as messages, destinations, durable subscriptions, and transactions. Broker configuration properties related to file-based persistence are listed under "File-Based Persistence" on page 293.

The file-based store is located in a directory identified by the name of the broker instance (*instanceName*) to which the data store belongs:

…/`instances`/*instanceName*/`fs350/`

(See Appendix A, "Platform-Specific Locations of Message Queue Data," for the location of the `instances` directory.) Each destination on the broker has its own subdirectory holding messages delivered to that destination.

NOTE Because the persistent data store can contain messages of a sensitive or proprietary nature, you should secure the …/`instances`/*instanceName*/`fs350/` directory against unauthorized access; see "Securing Persistent Data" on page 95.

All persistent data other than messages is stored in separate files: one file for destinations, one for durable subscriptions, and one for transaction state information. Most messages are stored in a single file consisting of variable-sized records. You can compact this file to alleviate fragmentation as messages are added and removed (see "Compacting Physical Destinations" on page 126). In addition, messages above a certain threshold size are stored in their own individual files rather than in the variable-sized record file. You can configure this threshold size with the broker property `imq.persist.file.message.max_record_size`.

The broker maintains a file pool for these individual message files: instead of being deleted when it is no longer needed, a file is returned to the pool of free files in its destination directory so that it can later be reused for another message. The broker property `imq.persist.file.destination.message.filepool.limit` specifies the maximum number of files in the pool. When the number of individual message files for a destination exceeds this limit, files will be deleted when no longer needed instead of being returned to the pool.

When returning a file to the file pool, the broker can save time at the expense of storage space by simply tagging the file as available for reuse without deleting its previous contents. You can use the `imq.persist.file.message.filepool.cleanratio` broker property to specify the percentage of files in each destination's file pool that should be maintained in a "clean" (empty) state rather than simply marked for reuse. The higher you set this value, the less space will be required for the file pool, but the more overhead will be needed to empty the contents of files when they are returned to the pool. If the broker's `imq.persist.file.message.cleanup` property is `true`, all files in the pool will be emptied at broker shutdown, leaving them in a clean state; this conserves storage space but slows down the shutdown process.

In writing data to the persistent store, the operating system has some leeway in whether to write the data synchronously or "lazily" (asynchronously). Lazy storage can lead to data loss in the event of a system crash, if the broker believes the data to have been written to persistent storage when it has not. To ensure absolute reliability (at the expense of performance), you can require that all data be written synchronously by setting the broker property `imq.persist.file.sync.enabled` to `true`. In this case, the data is guaranteed to be available when the system comes back up after a crash, and the broker can reliably resume operation. Note, however, that although the data is not lost, it is not available to any other broker in a cluster, since clustered brokers do not currently share data.

JDBC-Based Persistence

Instead of using file-based persistence, you can set up a broker to access any data store accessible through a JDBC-compliant driver. This involves setting the appropriate JDBC-related broker configuration properties and using the Database Manager utility (`imqdbmgr`) to create a database with the proper schema. See "Configuring a JDBC-Based Store" on page 94 for specifics.

The properties for configuring a broker to use a JDBC database are listed under "JDBC-Based Persistence" on page 295. You can specify these properties either in the instance configuration file (`config.properties`) of each broker instance or by using the `-D` command line option to the Broker utility (`imqbrokerd`) or the Database Manager utility (`imqdbmgr`).

The `imq.persist.jdbc.driver` property gives the Java class name of the JDBC driver to use in connecting to the database. There are also properties specifying the URLs for connecting to an existing database (`imq.persist.jdbc.opendburl`), creating a new database (`imq.persist.jdbc.createdburl`), and closing a database connection (`imq.persist.jdbc.closedburl`).

The `imq.persist.jdbc.user` and `imq.persist.jdbc.password` properties give the user name and password for accessing the database; `imq.persist.jdbc.needpassword` is a boolean flag specifying whether a password is needed. For security reasons, the password should be specified only in a password file designated via the `-passfile` command line option; if no such password file is specified, the `imqbrokerd` and `imqdbmgr` commands will prompt for the password interactively. Similarly, the user name can be supplied from the command line using the `-dbuser` option to the `imqbrokerd` command or the `-u` option to `imqdbmgr`.

In a JDBC database shared by multiple broker instances, the configuration property `imq.persist.jdbc.brokerid` specifies a unique instance identifier for each, to be appended to the names of database tables. (This is usually unnecessary for an embedded database, which stores data for only one broker instance.) The remaining JDBC-related configuration properties are used to customize the SQL code that creates the database schema, one property for each database table. For instance, the `imq.persist.jdbc.table.IMQSV35` property gives the SQL command for creating the version table, `imq.persist.jdbc.table.IMQCCREC35` for the configuration change record table, `imq.persist.jdbc.table.IMQDEST35` for the destination table, and so on; see Table 14-6 on page 295 for the complete list.

NOTE Because database systems vary in the exact SQL syntax required, be sure to check the documentation from your database vendor for details.

Security Services

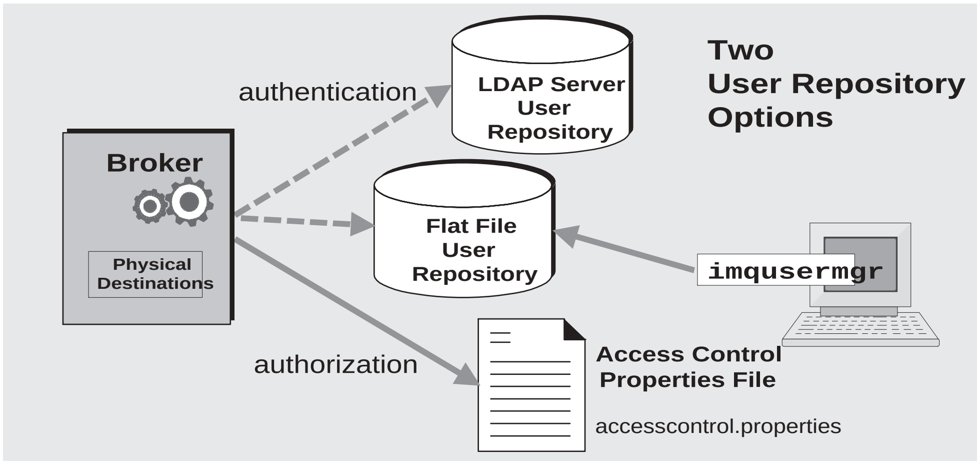
Message Queue provides security services for user access control (authentication and authorization) and for encryption:

- *Authentication* ensures that only verified users can establish a connection to a broker.
- *Authorization* specifies which users or groups have the right to access resources and to perform specific operations.
- *Encryption* protects messages from being tampered with during delivery over a connection.

As a Message Queue administrator, you are responsible for setting up the information the broker needs to authenticate users and authorize their actions. The broker properties pertaining to security services are listed under "Security Properties" on page 298. The boolean property `imq.accesscontrol.enabled` acts as a master switch that controls whether access control is applied on a brokerwide basis; for finer control, you can override this setting for a particular connection service by setting the `imq.`*serviceName*`.accesscontrol.enabled` property, where *serviceName* is the name of the connection service, as shown in Table 4-1 on page 76: for example, `imq.httpjms.accesscontrol.enabled`.

Figure 4-2 shows the components needed by the broker to provide authentication and authorization services. These services depend on a *user repository* containing information about the users of the messaging system: their names, passwords, and group memberships. In addition, to authorize specific operations for a user or group, the broker consults an *access control properties file* that specifies which operations a user or group can perform. You can designate a single access control properties file for the broker as a whole, using the configuration property `imq.accesscontrol.file.filename`, or for a single connection service with `imq.`*serviceName*`.accesscontrol.file.filename`.

Figure 4-2 Security Support


Two User Repository Options
authentication
LDAP Server User Repository
Broker
Physical Destinations
Flat File User Repository
imqusermgr
authorization
Access Control Properties File
accesscontrol.properties

As Figure 4-2 shows, you can store user data in a flat-file user repository that is provided with the Message Queue service or you can plug in a preexisting Lightweight Directory Access Protocol (LDAP) repository:

- If you choose a flat-file repository, you must use the Message Queue User Manager utility (`imqusermgr`) to manage the repository. This option is built-in and easy to use.

- If you want to use an existing LDAP server, you use the tools provided by the LDAP vendor to populate and manage the user repository. You must also set properties in the broker's instance configuration file to enable the broker to query the LDAP server for information about users and groups.

The broker's `imq.authentication.basic.user_repository` property specifies which type of repository to use. In general, an LDAP repository is preferable if scalability is important or if you need the repository to be shared by different brokers (if you are using broker clusters, for instance). See "Authenticating Users" on page 132 for more information on setting up a flat-file or LDAP user repository.

Authentication

A client requesting a connection to a broker must supply a user name and password, which the broker compares with those stored in the user repository. Passwords transmitted from client to broker are encoded using either base-64 encoding (for flat-file repositories) or message digest (MD5) hashing (for LDAP repositories). The choice is controlled by the `imq.authentication.type` property for the broker as a whole, or by `imq.`*serviceName*`.authentication.type` for a specific connection service. The `imq.authentication.client.response.timeout` property sets a timeout interval for authentication requests.

As described under "Using a Password File" on page 158, you can choose to put your passwords in a *password file* instead of being prompted for them interactively. The boolean broker property `imq.passfile.enabled` controls this option. If this property is true, the `imq.passfile.dirpath` and `imq.passfile.name` properties give the directory path and file name for the password file. The `imq.imqcmd.password` property (which can be embedded in the password file) specifies the password for authenticating an administrative user to use the Command utility (`imqcmd`) for managing brokers, connection services, connections, physical destinations, durable subscriptions, and transactions.

If you are using an LDAP-based user repository, there are a whole range of broker properties available for configuring various aspects of the LDAP lookup. The address (host name and port number) of the LDAP server itself is specified by `imq.user_repository.ldap.server`. The `imq.user_repository.ldap.principal` property gives the distinguished name for binding to the LDAP repository, while `imq.user_repository.ldap.password` supplies the associated password. Other properties specify the directory bases and optional JNDI filters for individual user and group searches, the provider-specific attribute identifiers for user and group names, and so forth; see "Security Properties" on page 298 for details.

Authorization

Once authenticated, a user can be authorized to perform various Message Queue-related activities. As a Message Queue administrator, you can define user groups and assign individual users membership in them. The default access control properties file explicitly refers to only one group, `admin` (see "Groups" on page 135). A user in this group has connection permission for the `admin` connection service, which allows the user to perform administrative functions such as creating destinations and monitoring and controlling a broker. A user in any other group that you define cannot, by default, get an `admin` service connection.

When a user attempts to perform an operation, the broker checks the user's name and group membership (from the user repository) against those specified for access to that operation (in the access control properties file). The access control properties file specifies permissions to users or groups for the following operations:

- Connecting to a broker
- Accessing destinations: creating a consumer, a producer, or a queue browser for any given destination or for all destinations
- Auto-creating destinations

Encryption

To encrypt messages sent between clients and broker, you need to use a connection service based on the Secure Socket Layer (SSL) standard. SSL provides security at the connection level by establishing an encrypted connection between an SSL-enabled broker and client.

To use an SSL-based Message Queue connection service, you generate a private/public key pair using the Key Tool utility (`imqkeytool`). This utility embeds the public key in a self-signed certificate and places it in a Message Queue key store. The key store is itself password-protected; to unlock it, you must provide a key store password at startup time, specified by the `imq.keystore.password` property. Once the key store is unlocked, a broker can pass the certificate to any client requesting a connection. The client then uses the certificate to set up an encrypted connection to the broker.

The `imq.audit.enabled` broker property controls the logging of audit records to the Message Queue broker log file; see "Creating an Audit Log" on page 160 for more information.

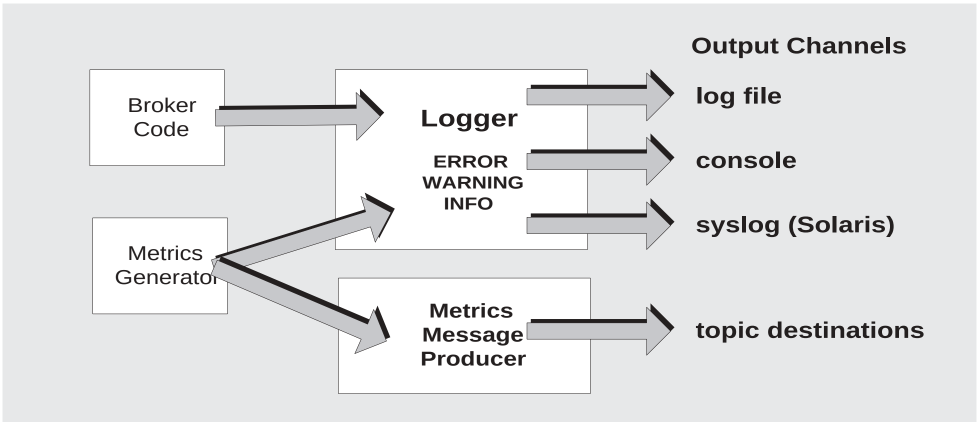
Monitoring Services

The broker includes components for monitoring and diagnosing application and broker performance. These include the following:

- Components that generate data, a Metrics Generator and broker code that logs events
- A Logger component that writes out information to a number of output channels
- A Metrics Message Producer that sends JMS messages containing metric information to topic destinations for consumption by JMS monitoring clients

The general scheme is illustrated in Figure 4-3. Broker properties for configuring the monitoring services are listed under "Monitoring Properties" on page 303.

Figure 4-3 Monitoring Support



Metrics Generator

The Metrics Generator provides information about broker activity, such as message flow in and out of the broker, the number of messages in broker memory and the memory they consume, the number of open connections, and the number of threads being used. The boolean broker property `imq.metrics.enabled` controls whether such information is logged; `imq.metrics.interval` specifies how often.

Logger

The Logger takes information generated by broker code and the Metrics Generator and writes that information to standard output (the console), to a log file, and, on Solaris platforms, to the `syslog` daemon process in case of errors. The log file to use is identified by the `imq.log.file.dirpath` and `imq.log.file.filename` broker properties; `imq.log.console.stream` specifies whether console output is directed to `stdout` or `stderr`.

The `imq.log.level` property controls the categories of metric information that the Logger gathers: `ERROR`, `WARNING`, or `INFO`. Each level includes those above it, so if you specify, for example, `WARNING` as the logging level, error messages will be logged as well. The `imq.log.console.output` and `imq.log.file.output` properties control which of the specified categories will be written to the console and the log file, respectively. In this case, however, the categories do *not* include those above them; so if you want, for instance, both errors and warnings written to the log file and informational messages to the console, you must explicitly set `imq.log.file.output` to `ERROR|WARNING` and `imq.log.console.output` to `INFO`. On Solaris platforms another property, `imq.log.syslog.output`, specifies the categories of metric information to be written to the `syslog` daemon. There is also an `imq.destination.logDeadMsgs` property that specifies whether to log when dead messages are discarded or moved to the dead message queue.

In the case of a log file, you can specify the point at which the file is closed and output is rolled over to a new file. Once the log file reaches a specified size (`imq.log.file.rolloverbytes`) or age (`imq.log.file.rolloversecs`), it is saved and a new log file created.

See "Monitoring Properties" on page 303 for additional broker properties related to logging, and "Configuring and Using Broker Logging" on page 191 for further details about how to configure the Logger and how to use it to obtain performance information.

Metrics Message Producer (Enterprise Edition)

The Metrics Message Producer receives information from the Metrics Generator at regular intervals and writes the information into *metrics messages,* which it then sends to one of a number of metric topic destinations, depending on the type of metric information contained in the message (see Table 4-2). Message Queue clients subscribed to these metric topic destinations can consume the messages and process the metric data they contain. This allows developers to create custom monitoring tools to support messaging applications. For details of the metric quantities reported in each type of metrics message, see the *Message Queue Developer's Guide for Java Clients.*

Table 4-2 Metric Topic Destinations

Topic Name	Type of Metric Information
`mq.metrics.broker`	Broker metrics
`mq.metrics.jvm`	Java Virtual Machine metrics
`mq.metrics.destination_list`	List of destinations and their types
`mq.metrics.destination.queue.`*queueName*	Destination metrics for specified queue
`mq.metrics.destination.topic.`*topicName*	Destination metrics for specified topic

The broker properties `imq.metrics.topic.enabled` and `imq.metrics.topic.interval` control, respectively, whether messages are sent to metric topic destinations and how often. The `imq.metrics.topic.timetolive` and `imq.metrics.topic.persist` properties specify the lifetime of such messages and whether they are persistent.

Besides the information contained in the body of a metrics message, the header of each message includes properties that provide the following additional information:

- The message type
- The address (host name and port number) of the broker that sent the message
- The time the metric sample was taken

These properties are useful to client applications that process metrics messages of different types or from different brokers.

Setting Broker Properties

You can specify a broker's configuration properties in either of two ways:

- Edit the broker's configuration file
- Supply the property values directly from the command line

The following two sections describe these two methods of configuring a broker.

Configuration Files

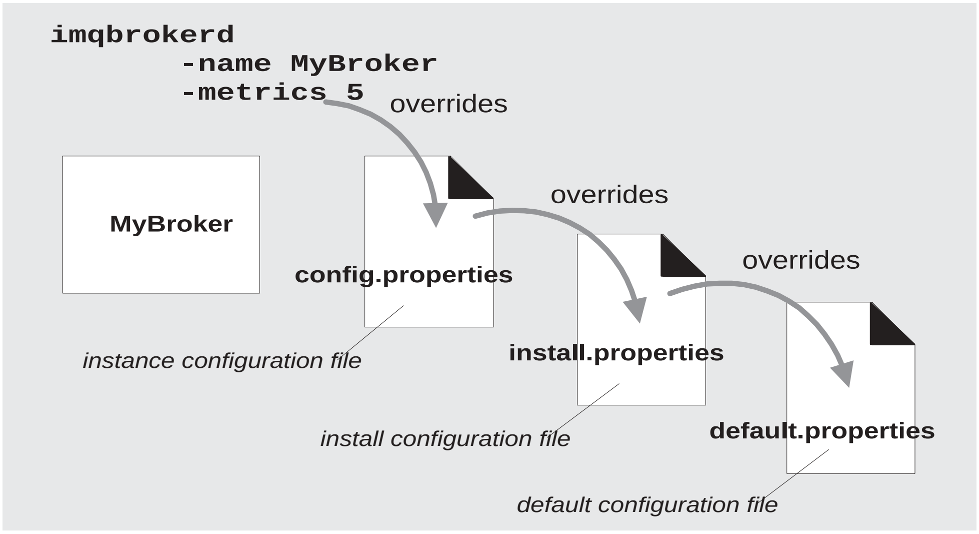
Broker configuration files contain property settings for configuring a broker. They are kept in a directory whose location depends on the operating system platform you are using; see Appendix A, "Platform-Specific Locations of Message Queue Data" for details. The directory stores the following files:

- A *default configuration file*, `default.properties`, that is loaded on startup. This file is not editable, but you can read it to determine default settings and find the exact names of properties you want to change.
- An *installation configuration file*, `install.properties`, containing any properties specified when Message Queue was installed. This file cannot be edited after installation.

In addition, each individual broker instance has its own *instance configuration file*, as described below. If you connect broker instances in a cluster, you may also need to use a *cluster configuration file* to specify configuration information for the cluster; see "Cluster Configuration Properties" on page 307 for more information.

At startup, the broker merges property values from the various configuration files. As shown in Figure 4-4, the files form a hierarchy in which values specified in the instance configuration file override those in the installation configuration file, which in turn override those in the default configuration file. At the top of the hierarchy, you can manually override any property values specified in the configuration files by using command line options to the `imqbrokerd` command.

Figure 4-4 Broker Configuration Files



Editing the Instance Configuration File

The first time you run a broker, an instance configuration file is created containing configuration properties for that particular broker instance. The instance configuration file is named `config.properties` and is stored in a directory identified by the name of the broker instance to which it belongs:

…`/instances/`*instanceName*`/props/config.properties`

(See Appendix A, "Platform-Specific Locations of Message Queue Data," for the location of the `instances` directory.) If the file does not yet exist, you must use the `-name` option when starting the broker (see "Broker Utility" on page 266), to specify an instance name that Message Queue can use to create the file.

NOTE The `instances/`*instanceName* directory and the instance configuration file are owned by the user who created the corresponding broker instance. The broker instance must always be restarted by that same user.

The instance configuration file is maintained by the broker instance and is modified when you make configuration changes using Message Queue administration utilities. You can also edit an instance configuration file by hand to customize the broker's behavior and resource use. To do so, you must be the owner of the `instances/`*instanceName* directory or log in as `root` to change the directory's access privileges.

The broker reads its instance configuration file only at startup. To make permanent changes to the broker's configuration, you must shut down the broker, edit the file, and then restart the broker. Property definitions in the file (or any configuration file) use the following syntax:

propertyName=*value*[[`,` *value1*]...]

For example, the following entry specifies that the broker will hold up to 50,000 messages in memory and persistent storage before rejecting additional messages:

```
imq.system.max_count=50000
```

The following entry specifies that a new log file will be created every day (86,400 seconds):

```
imq.log.file.rolloversecs=86400
```

See "Broker Services" on page 75 and Chapter 14, "Broker Properties Reference," for information on the available broker configuration properties and their default values.

Setting Configuration Options from the Command Line

You can enter broker configuration options from the command line when you start a broker, or afterward.

At startup time, you use the Broker utility (`imqbrokerd`) to start a broker instance. Using the command's `-D` option, you can specify any broker configuration property and its value; see "Starting Brokers" on page 66 and "Broker Utility" on page 266 for more information. If you start the broker as a Windows service, using the Service Administrator utility (`imqsvcadmin`), you use the `-args` option to specify startup configuration properties; see "Service Administrator Utility" on page 283.

You can also change certain broker properties while a broker instance is running. To modify the configuration of a running broker, you use the Command utility's `imqcmd update bkr` command; see "Updating Broker Properties" on page 102 and "Broker Management" on page 272.

Configuring a Persistent Data Store

A broker's persistent data store holds information about physical destinations, durable subscriptions, messages, transactions, and acknowledgments. Message Queue brokers are configured by default to use a file-based persistent store, but you can reconfigure them to plug in any data store accessible through a JDBC-compliant driver. The broker configuration property `imq.persist.store` (see Table 14-4 on page 292) specifies which of the two forms of persistence to use.

This section explains how to set up a broker to use a persistent store. It includes the following topics:

- "Configuring a File-Based Store" on page 93
- "Configuring a JDBC-Based Store" on page 94
- "Securing Persistent Data" on page 95

Configuring a File-Based Store

A file-based data store is automatically created when you create a broker instance. The store is located in the broker's instance directory; see Appendix A, "Platform-Specific Locations of Message Queue Data," for the exact location.

By default, Message Queue performs asynchronous write operations to disk. The operating system can buffer these operations for efficient performance. However, if an unexpected system failure should occur between write operations, messages could be lost. To improve reliability (at the cost of reduced performance), you can set the broker property `imq.persist.file.sync` to write data synchronously instead. For further discussion about this property, see "File-Based Persistence" on page 81 and Table 14-5 on page 293.

When you start a broker instance, you can use the `imqbrokerd` command's `-reset` option to clear the file system store. For more information about this option and its suboptions, see "Broker Utility" on page 266.

Configuring a JDBC-Based Store

To configure a broker to use JDBC-based persistence, you set JDBC-related properties in the broker's instance configuration file and create the appropriate database schema. The Message Queue Database Manager utility (`imqdbmgr`) uses your JDBC driver and the broker configuration properties to create and manage the database. You can also use the Database Manager to delete corrupted tables from the database or if you want to use a different database as a data store. See "Database Manager Utility" on page 280 for more information.

NOTE Example configurations for Oracle and PointBase database products are available. The location of these files is platform-dependent, and is listed under "Example applications and configurations" in the relevant tables of Appendix A, "Platform-Specific Locations of Message Queue Data." In addition, examples for PointBase embedded version, PointBase server version, and Oracle are provided as commented-out values in the instance configuration file, `config.properties`.

➤ To Configure a JDBC-Based Data Store

1. Set JDBC-related properties in the broker's configuration file.

   The relevant properties are discussed under "JDBC-Based Persistence" on page 82 and listed in Table 14-6 on page 295. In particular, you must set the broker's `imq.persist.store` property to `jdbc` (see Table 14-4 on page 292).

2. Place a copy of, or a symbolic link to, your JDBC driver's `.jar` file in the following location:

   `/usr/share/lib/imq/ext/` (Solaris)
   `/opt/sun/mq/share/lib/` (Linux)
   `IMQ_VARHOME\lib\ext` (Windows)

   For example, if you are using PointBase on a Solaris system, the following command copies the driver's `.jar` file to the appropriate location:

   `% cp` *j2eeSDKInstallDirectory*`/pointbase/lib/pointbase.jar /usr/share/lib/imq/ext`

   The following command creates a symbolic link instead:

   `% ln -s` *j2eeSDKInstallDirectory*`/lib/pointbase/pointbase.jar /usr/share/lib/imq/ext`

3. Create the database schema needed for Message Queue persistence.

Use the `imqdbmgr create all` command (for an embedded database) or the `imqdbmgr create tbl` command (for an external database); see "Database Manager Utility" on page 280.

a. Change to the directory where `imqdbmgr` resides:

`cd /usr/bin` (Solaris)
`cd /opt/sun/mq/bin` (Linux)
`cd IMQ_HOME\bin` (Windows)

b. Enter the `imqdbmgr` command:

```
imqdbmgr create all
```

NOTE If you use an embedded database, it is best to create it under the following directory:

…`/instances/`*instanceName*`/dbstore/`*databaseName*

If an embedded database is not protected by a user name and password, it is probably protected by file system permissions. To ensure that the database is readable and writable by the broker, the user who runs the broker should be the same user who created the embedded database using the `imqdbmgr` command.

Securing Persistent Data

The persistent store can contain, among other information, message files that are being temporarily stored. Since these messages may contain proprietary information, it is important to secure the data store against unauthorized access. This section describes how to secure data in a file-based or JDBC-based data store.

Securing a File-Based Store

A broker using file-based persistence writes persistent data to a flat-file data store whose location is platform-dependent (see Appendix A, "Platform-Specific Locations of Message Queue Data"):

…`/instances/`*instanceName*`/fs350/`

where *instanceName* is a name identifying the broker instance.

The *instanceName*/`fs350/` directory is created when the broker instance is started for the first time. The procedure for securing this directory depends on the operating system platform on which the broker is running:

- On Solaris and Linux, the directory's permissions are determined by the file mode creation mask (`umask`) of the user who started the broker instance. Hence, permission to start a broker instance and to read its persistent files can be restricted by setting the mask appropriately. Alternatively, an administrator (superuser) can secure persistent data by setting the permissions on the `instances` directory to `700`.
- On Windows, the directory's permissions can be set using the mechanisms provided by the Windows operating system. This generally involves opening a Properties dialog for the directory.

Securing a JDBC-Based Store

A broker using JDBC-based persistence writes persistent data to a JDBC-compliant database. For a database managed by a database server (such as Oracle), it is recommended that you create a user name and password to access the Message Queue database tables (tables whose names start with `IMQ`). If the database does not allow individual tables to be protected, create a dedicated database to be used only by Message Queue brokers. See the documentation provided by your database vendor for information on how to create user name / password access.

The user name and password required to open a database connection by a broker can be provided as broker configuration properties. However it is more secure to provide them as command line options when starting up the broker, using the `imqbrokerd` command's `-dbuser`and `-dbpassword` options (see "Broker Utility" on page 266).

For an embedded database that is accessed directly by the broker via the database's JDBC driver, security is usually provided by setting file permissions on the directory where the persistent data will be stored, as described above under "Securing a File-Based Store." To ensure that the database is readable and writable by both the broker and the Database Manager utility, however, both should be run by the same user.

Chapter 5

Managing a Broker

This chapter explains how yo use the `imqcmd` utility to manage the broker and its services. This chapter has the following sections:

This chapter does not cover all topics related to managing a broker. Additional topics are covered in the following separate chapters:

- Management of physical destinations on the broker. For information about topics such as how to create, display, update and destroy physical destinations, and how to use the dead message queue, see Chapter 6, "Managing Physical Destinations."
- Setting up security for the broker. For information about topics such as user authentication, access control, encryption, password files, and audit logging, see Chapter 7, "Managing Security."

Prerequisites

You use the imqcmd and imqusermgr command line utilities to manage the broker. Before managing the broker, you must do the following:

- Start the broker using the imqbrokerd utility command. You cannot use the other command line utilities until a broker is running.
- Determine whether you want to set up a Message Queue administrative user or use the default account. You must specify a user name and password to use management commands.

  When you install Message Queue, a default flat-file user repository is installed. The repository is shipped with two default entries: an admin user and a guest user. If you are testing Message Queue, you can use the default user name and password (admin/admin) to run the imqcmd utility.

  If you are setting up a production system, you must set up authentication and authorization for administrative users. See Chapter 7, "Managing Security" for information on setting up a file-based user repository or configuring the use of an LDAP directory server. In a production environment, it is a good security practice to use a nondefault user name and password.
- Set up and enable the ssladmin service on the target broker instance, if you want to use a secure connection to the broker. For more information, see "Working With an SSL-Based Service" on page 148.

Using the imqcmd Utility

The imqcmd utility enables you to manage the broker and its services.

Reference information about the syntax, subcommands, and options of the imqcmd command is in Chapter 13, "Command Line Reference" on page 265. Reference information for use in managing physical destinations is in a separate chapter, Chapter 15, "Physical Destination Property Reference" on page 313.

Displaying Help

To display help on the imqcmd utility, use the -h or -H option, and do not use a subcommand. You cannot get help about specific subcommands.

For example, the following command displays help about `imqcmd`:

```
imqcmd -H
```

If you enter a command line that contains the `-h` or `-H` option in addition to a subcommand or other options, the `imqcmd` utility processes only the `-h` or `-H` option. All other items on the command line are ignored.

Displaying the Product Version

To display the Message Queue product version, use the `-v` option. For example:

```
imqcmd -v
```

If you enter a command line that contains the `-v` option in addition to a subcommand or other options, the `imqcmd` utility processes only the `-v` option. All other items on the command line are ignored.

Specifying the User Name and Password

Because each `imqcmd` subcommand is authenticated against the user repository, it requires a user name and password. The only exceptions are commands that use the `-h` or `-H` option to display help, and commands that use the `-v` option to display the product version.

Specifying the User Name

Use the `-u` option to specify an administrative user name. If you omit the user name, the command prompts you for it. For example, the following command displays information about the default broker:

```
imqcmd query bkr -u admin
```

To make the examples in this chapter easy to read, the default user name `admin` is shown as the argument to the `-u` option. In a production environment, you would use a custom user name.

Specifying the Password

Specify the password using one of the following methods:

- Create a password file (passfile) and enter the password into that file. On the command line, use the `-passfile` option to provide the name of the password file.

- Let the command prompt you for the password.

In previous versions of Message Queue, you could use the `-p` option to specify a password on the `imqcmd` command line. This option is being deprecated and will be removed in a future version.

Specifying the Broker Name and Port

The default broker for `imqcmd` is one that is running on the local host, and the default port is `7676`.

If you are issuing a command to a broker running on a remote host or listening on a nondefault port, or both, you must use the `-b` option to specify the broker's host and port.

Examples

The examples in this section illustrate how to use `imqcmd`.

The first example lists the properties of the broker running on `localhost` at port `7676`, so the `-b` option is unnecessary. The command uses the default administrative user name (`admin`) and omits the password, so that the command prompts for it.

```
imqcmd query bkr -u admin
```

The following example lists the properties of the broker running on the host `myserver` at port `1564`. The user name is `aladdin`. (For this command to work, the user repository would need to be updated to add the user name `aladdin` to the `admin` group.)

```
imqcmd query bkr -b myserver:1564 -u aladdin
```

The following example lists the properties of the broker running on `localhost` at port `7676`. The initial timeout for the command is set to 20 seconds and the number of retries after timeout is set to 7. The user's password is in a password file called `myPassfile`, located in the current directory at the time the command is invoked.

```
imqcmd query bkr -u admin -passfile myPassfile -rtm 20 -rtr 7
```

For a secure connection to the broker, these examples could include the `-secure` option. The `-secure` option causes `imqcmd` to use the `ssladmin` service if the service has been configured and started.

Displaying Broker Information

To query and display information about a single broker, use the `query bkr` subcommand.

This is the syntax of the `query bkr` subcommand:

```
imqcmd query bkr -b hostName:portNumber
```

This subcommand lists the current settings of properties for the default broker or a broker at the specified host and port. It also shows the list of running brokers (in a multi-broker cluster) that are connected to the specified broker.

For example:

```
imqcmd query bkr -u admin
```

After prompting you for the password, the command produces output like the following:

```
Version                                                  3.6
Instance Name                                            imqbroker
Primary Port                                             7676

Current Number of Messages in System                     0
Current Total Message Bytes in System                    0

Current Number of Messages in Dead Message Queue         0
Current Total Message Bytes in Dead Message Queue        0

Log Dead Messages                                        true
Truncate Message Body in Dead Message Queue              false

Max Number of Messages in System                         unlimited (-1)
Max Total Message Bytes in System                        unlimited (-1)
Max Message Size                                         70m

Auto Create Queues                                       true
Auto Create Topics                                       true
Auto Created Queue Max Number of Active Consumers        1
Auto Created Queue Max Number of Backup Consumers        0

Cluster Broker List (active)
Cluster Broker List (configured)
Cluster Master Broker
Cluster URL

Log Level                                                INFO
Log Rollover Interval (seconds)                          604800
Log Rollover Size (bytes)                                unlimited (-1)
```

Updating Broker Properties

You can use the `update bkr` subcommand to update the following broker properties:

- `imq.autocreate.queue`
- `imq.autocreate.topic`
- `imq.autocreate.queue.maxNumActiveConsumers`
- `imq.autocreate.queue.maxNumBackupConsumers`
- `imq.cluster.url`
- `imq.destination.DMQ.truncateBody`
- `imq.destination.logDeadMsgs`
- `imq.log.level`
- `imq.log.file.rolloversecs`
- `imq.log.file.rolloverbytes`
- `imq.system.max_count`
- `imq.system.max_size`
- `imq.message.max_size`
- `imq.portmapper.port`

This is the syntax of the `update bkr` subcommand:

```
imqcmd update bkr [-b hostName:portNumber]-o attribute=value [[-o attribute=value1]...]
```

The subcommand changes the specified attributes for the default broker or a broker at the specified host and port. For example, the following command turns off the auto-creation of queue destinations:

```
imqcmd update bkr -o "imq.autocreate.queue=false" -u admin
```

The properties are described in Chapter 14, "Broker Properties Reference."

Pausing and Resuming a Broker

After you start the broker, you can use `imqcmd` subcommands to control the state of the broker.

Pausing a Broker

Pausing a broker suspends the broker’s connection service threads, which causes the broker to stop listening on the connection ports. As a result, the broker will no longer be able to accept new connections, receive messages, or dispatch messages.

However, pausing a broker does not suspend the `admin` connection service, letting you perform administration tasks needed to regulate the flow of messages to the broker. Pausing a broker also does not suspend the `cluster` connection service. However message delivery within a cluster depends on the delivery functions performed by the different brokers in the cluster. Therefore, pausing a broker in a cluster might result in a slowing of some message traffic.

This is the syntax of the `pause bkr` subcommand:

```
imqcmd pause bkr [-b hostName:portNumber]
```

The command pauses the default broker or a broker at the specified host and port.

The following command pauses the broker running on `myhost` at port `1588`.

```
imqcmd pause bkr -b myhost:1588 -u admin
```

You can also pause individual connection services and individual physical destinations. For more information, see “Pausing and Resuming a Connection Service” on page 110 and “Pausing and Resuming Physical Destinations” on page 123.

Resuming a Broker

Resuming a broker reactivates the broker’s service threads and the broker resumes listening on the ports.

This is the syntax of the `resume bkr` subcommand:

```
imqcmd resume bkr [-b hostName:portNumber]
```

The subcommand resumes the default broker or a broker at the specified host and port.

The following command resumes the broker running on `localhost` at port `7676`.

```
imqcmd resume bkr -u admin
```

Shutting Down and Restarting a Broker

Shutting down the broker gracefully terminates the broker process. The broker stops accepting new connections and messages, completes delivery of existing messages, and terminates the broker process.

This is the syntax of the `shutdown bkr` subcommand:

```
imqcmd shutdown bkr [-b hostName:portNumber]
```

The subcommand shuts down the default broker or a broker at the specified host and port.

The following command shuts down the broker running on `ctrlsrv` at port `1572`:

```
imqcmd shutdown bkr -b ctrlsrv:1572 -u admin
```

Use the `restart bkr` subcommand to shut down and restart the broker. This is the syntax of the `restart bkr` subcommand:

```
imqcmd restart bkr [-b hostName:portNumber]
```

The subcommand shuts down and restarts the default broker or a broker at the specified host and port, using the options specified when the broker first started. To choose different options, shut down the broker and then restart it, specifying the options you want.

Displaying Broker Metrics

To display metrics information about a broker, use the `metrics bkr` subcommand.

This is the syntax of the `metrics bkr` subcommand:

```
imqcmd metrics bkr [-b hostName:portNumber]
        [-m metricType] [-int interval] [-msp numSamples]
```

The subcommand displays broker metrics for the default broker or a broker at the specified host and port.

Use the `-m` option to specify one of the following metric types to display:

- **`ttl`** Displays metrics about the messages and packets flowing into and out of the broker (default metric type).
- **`rts`** Displays metrics about the rate of flow of messages and packets into and out of the broker (per second).
- **`cxn`** Displays connections, virtual memory heap, and threads.

Use the `-int` option to specify the interval (in seconds) at which to display the metrics. The default is 5 seconds.

Use the `-msp` option to specify the number of samples displayed in the output. The default is an unlimited number (infinite).

For example, to get the rate of message flow into and out of the broker at ten second intervals:

```
imqcmd metrics bkr -m rts -int 10 -u admin
```

This command produces output like the following:

```
--------------------------------------------------------
 Msgs/sec   Msg Bytes/sec   Pkts/sec    Pkt Bytes/sec
 In   Out     In      Out     In   Out     In      Out
--------------------------------------------------------
 0     0      27      56      0     0      38      66
 10    0      7365    56      10    10     7457    1132
 0     0      27      56      0     0      38      73
 0     10     27     7402     10    20     1400    8459
 0     0      27      56      0     0      38      73
```

For a more detailed descriptionabout the data gathered and reported by the broker, see "Brokerwide Metrics" on page 336.

Managing Connection Services

The `imqcmd` utility includes subcommands that allow you to perform the following connection service management tasks:

- Listing Connection Services
- Displaying Connection Service Information
- Updating Connection Service Properties
- Displaying Connection Service Metrics
- Pausing and Resuming a Connection Service

A broker supports connections from both application clients and administration clients. The connection services currently available from a Message Queue broker are shown in Table 5-1. As shown in the table, each service is associated with the service type it uses (`NORMAL` for application clients or `ADMIN` for administration clients) and with an underlying transport protocol.

Table 5-1 Connection Services Supported by a Broker

Service Name	Service Type	Protocol Type
jms	NORMAL	tcp
ssljms (Enterprise Edition)	NORMAL	tls (SSL-based security)
httpjms (Enterprise Edition)	NORMAL	http
httpsjms (Enterprise Edition)	NORMAL	https (SSL-based security)
admin	ADMIN	tcp
ssladmin (Enterprise Edition)	ADMIN	tls (SSL-based security)

You can use `imqcmd` subcommands to manage connection services as a whole or to manage a particular connection service. If the target of a subcommand is a particular service, use the `-n` option to specify one of the names listed in the Service Name column of Table 5-1.

Listing Connection Services

To list available connection services on a broker, use the `list svc` subcommand.

This is the syntax of the `list svc` subcommand:

```
imqcmd list svc [-b hostName:portNumber]
```

The subcommand lists all connection services on the default broker or on a broker at the specified host and port.

The following command lists all services on the broker running on `localhost` at port `7676`:

```
imqcmd list svc -u admin
```

The command will output information like the following:

```
------------------------------------------------
Service Name    Port Number        Service State
------------------------------------------------
admin           41844 (dynamic)    RUNNING
httpjms         -                  UNKNOWN
httpsjms        -                  UNKNOWN
jms             41843 (dynamic)    RUNNING
ssladmin        dynamic            UNKNOWN
ssljms          dynamic            UNKNOWN
```

Displaying Connection Service Information

To query and display information about a single service, use the `query` subcommand.

This is the syntax for the `query svc` subcommand:

```
imqcmd query svc -n serviceName [-b hostName:portNumber]
```

The `query svc` subcommand displays information about the specified service running on the default broker or on a broker at the specified host and port.

For example:

```
imqcmd query svc -n jms -u admin
```

After prompting for the password, the command produces output like the following:

```
Service Name                          jms
Service State                         RUNNING
Port Number                           60920 (dynamic)

Current Number of Allocated Threads   0
Current Number of Connections         0

Min Number of Threads                 10
Max Number of Threads                 1000
```

Updating Connection Service Properties

You can use the `update` subcommand to change the value of one or more of the service properties listed in Table 5-2.

Table 5-2 Connection Service Properties Updated by `imqcmd`

Property	**Description**
`port`	The port assigned to the service to be updated (does not apply to httpjms or httpsjms). A value of `0` means the port is dynamically allocated by the Port Mapper.
`minThreads`	The minimum number of threads assigned to the service.
`maxThreads`	The maximum number of threads assigned to the service.

This is the syntax of the `update` subcommand:

```
imqcmd update svc -n serviceName [-b hostName:portNumber]
          -o attribute=value [-o attribute=value1]...
```

This subcommand updates the specified attribute of the specified service running on the default broker or on a broker at the specified host and port. For a description of service attributes, see “Connection Properties” on page 285.

The following command changes the minimum number of threads assigned to the jms service to 20.

```
imqcmd update svc -n jms -o “minThreads=20” -u admin
```

Displaying Connection Service Metrics

To display metrics information about a single service, use the `metrics` subcommand.

This is the syntax of the `metrics` subcommand:

```
imqcmd metrics svc -n serviceName [-b hostName:portNumber] [-m metricType]
  [-int interval] [-msp numSamples]
```

The subcommand displays metrics for the specified service on the default broker or on a broker at the specified host and port.

Use the `-m` option to specify the type of metric to display:

- **`ttl`** Displays metrics on messages and packets flowing into and out of the broker by way of the specified connection service. (default metric type).
- **`rts`** Displays metrics on rate of flow of messages and packets into and out of the broker (per second) by way of the specified connection service.
- **`cxn`** Displays connections, virtual memory heap, and threads.

Use the `-int` option to specify the interval (in seconds) at which to display the metrics. The default is 5 seconds.

Use the `-msp` option to specify the number of samples displayed in the output. The default is an unlimited number (infinite).

For example, to get cumulative totals for messages and packets handled by the jms connection service:

```
imqcmd metrics svc -n jms -m ttl -u admin
```

After prompting for the password, the command produces output like the following:

```
-------------------------------------------------
  Msgs      Msg Bytes      Pkts      Pkt Bytes
In   Out    In      Out   In   Out    In     Out
-------------------------------------------------
164  100  120704  73600  282  383  135967  102127
657  100  483552  73600  775  876  498815  149948
```

For a more detailed description of the use of `imqcmd` to report connection service metrics, see "Connection Service Metrics" on page 338.

Pausing and Resuming a Connection Service

To pause any service other than the admin service (which cannot be paused), use the `pause svc` and `resume svc` subcommands.

This is the syntax of the `pause svc` subcommand:

```
imqcmd pause svc -n serviceName [-b hostName:portNumber]
```

The subcommand pauses the specified service running on the default broker or on a broker at the specified host and port. For example, the following command pauses the `httpjms` service running on the default broker.

```
imqcmd pause svc -n httpjms -u admin
```

Pausing a service has the following effects:

- The broker stops accepting new client connections on the paused service. If a Message Queue client attempts to open a new connection, it will get an exception.
- All the existing connections on the paused service are kept alive, but the broker suspends all message processing on such connections until the service is resumed. (For example, if a client attempts to send a message, the `send` method will block until the service is resumed.)
- The message delivery state of any messages already received by the broker is maintained. (For example, transactions are not disrupted and message delivery will resume when the service is resumed.)

To resume a service, use the `resume svc` subcommand.

This is the syntax of the `resume svc` subcommand:

```
imqcmd resume svc -n serviceName[-b hostName:portNumber]
```

The subcommand resumes the specified service running on the default broker or on a broker at the specified host and port.

Getting Information About Connections

The `imqcmd` utility includes subcommands that allow you to list and get information about connections.

The `list cxn` subcommand lists all connections of a specified service name. This is the syntax of the list cxn subcommand:

```
imqcmd list cxn [-svn serviceName] [-b hostName:portNumber]
```

The subcommand lists all connections of the specified service name on the default broker or on a broker at the specified host and port. If the service name is not specified, all connections are listed.

For example, the following command lists all connections on the default broker:

```
imqcmd list cxn -u admin
```

After prompting for the password, the command produces output like the following:

```
Listing all the connections on the broker specified by:
-----------------------------------
Host                   Primary Port
------------------------------------
localhost              7676

---------------------------------------------------------------------------
Connection ID         User    Service   Producers  Consumers    Host
---------------------------------------------------------------------------
1964412264455443200   guest   jms       0          1            127.0.0.1
1964412264493829311   admin   admin     1          1            127.0.0.1

Successfully listed connections.
```

To query and display information about a single connection service, use the `query` subcommand.

```
query cxn -n connectionID [-b hostName:portNumber]
```

The subcommand displays information about the specified connection on the default broker or on a broker at the specified host and port.

For example:

```
imqcmd query cxn -n 421085509902214374 -u admin
```

After prompting for the password, the command produces output like the following:

```
Connection ID      421085509902214374
User               guest
Service            jms
Producers          0
Consumers          1
Host               111.22.333.444
Port               60953
Client ID
Client Platform
```

Managing Durable Subscriptions

Using `imqcmd` subcommands you can manage a broker's durable subscriptions by doing one or more of the following:

- Listing durable subscriptions
- Purging all messages for a durable subscription
- Destroying a durable subscription

A *durable subscription* is a subscription to a topic that is registered by a client as durable; it has a unique identity and it requires the broker to retain messages for that subscription even when its consumer becomes inactive. Normally, the broker may only delete a message held for a durable subscriber when the message expires.

To list durable subscriptions for a specified physical destination, use the `list dur` subcommand. This is the syntax for the list dur subcommand:

```
imqcmd list dur -d destName
```

For example, the following command lists all durable subscriptions to the topic `SPQuotes`, using the broker at the default port on the local host:

```
imqcmd list dur -d SPQuotes
```

For each durable subscription to a topic, the `list dur` subcommand returns the name of the durable subscription, the client ID of the user, the number of messages queued to this topic, and the state of the durable subscription (active/inactive). For example:

```
Name        Client ID       Number of   Durable Sub
                            Messages      State
----------------------------------------------------------------
myDurable   myClientID       1           INACTIVE
```

You can use the information returned from the `list dur` subcommand to identify a durable subscription you might want to destroy or for which you want to purge messages.

The `purge dur` subcommand purges all messages for the specified durable subscription with the specified Client Identifier. This is the syntax for the `purge dur` subcommand:

`imqcmd purge dur -n` *subscrName* `-c` *clientID*

The `destroy dur` subcommand destroys a specified durable subscription with the specified client identifier. This is the syntax for the `destroy dur` subcommand:

`imqcmd destroy dur -n` *subscrName* `-c` *clientID*

For example, the following command destroys the durable subscription `myDurable` and clientID, `myClientID`.

```
imqcmd destroy dur -n myDurable -c myClientID
```

Managing Transactions

All transactions initiated by client applications are tracked by the broker. These can be simple Message Queue transactions or distributed transactions managed by a distributed transaction (XA resource) manager.

Each transaction has a Message Queue transaction ID—a 64 bit number that uniquely identifies a transaction on the broker. Distributed transactions also have a distributed transaction ID (XID) assigned by the distributed transaction manager—up to 128 bytes long. Message Queue maintains the association of an Message Queue transaction ID with an XID.

For distributed transactions, in cases of failure, it is possible that transactions could be left in a `PREPARED` state without ever being committed. Hence, as an administrator you might need to monitor and then roll back or commit transactions left in a prepared state.

To list all transactions, being tracked by the broker, use the `list txn` command. This is the syntax for the `list tx` subcommand:

```
imqcmd list txn
```

For example, the following command lists all transactions in a broker.

```
imqcmd list txn
```

For each transaction, the `list` subcommand returns the transaction ID, state, user name, number of messages or acknowledgments, and creation time. For example:

```
---------------------------------------------------------------
Transaction ID  State    User name   # Msgs/   Creation time
                                     # Acks
---------------------------------------------------------------

64248349708800  PREPARED  guest      4/0      1/30/02 10:08:31 AM
64248371287808  PREPARED  guest      0/4      1/30/02 10:09:55 AM
```

The command shows all transactions in the broker, both local and distributed. You can only commit or roll back transactions in the `PREPARED` state. You should only do so if you know that the transaction has been left in this state by a failure and is not in the process of being committed by the distributed transaction manager.

For example, if the broker's auto-rollback property is set to false (see Table 14-2 on page 287), you must manually commit or roll back transactions found in a `PREPARED` state at broker startup.

The `list` subcommand also shows the number of messages that were produced in the transaction and the number of messages that were acknowledged in the transaction (`#Msgs/#Acks`). These messages will not be delivered and the acknowledgments will not be processed until the transaction is committed.

The `query` subcommand lets you see the same information plus a number of additional values: the Client ID, connection identification, and distributed transaction ID (XID). This is the syntax of the `query txn` subcommand:

imqcmd query txn -n *transactionID*

For example, the following example produces the output shown below:

```
imqcmd query txn -n 64248349708800
```

```
Client ID
Connection                 guest@192.18.116.219:62209->jms:62195
Creation time              1/30/02 10:08:31 AM
Number of acknowledgments 0
Number of messages         4
State                      PREPARED
Transaction ID             64248349708800
User name                  guest
XID
6469706F6C7369646577696E64657231303132343134313130303373230
```

Use the `commit` and `rollback` subcommands to commit or roll back a distributed transaction. As mentioned previously, only a transaction in the `PREPARED` state can be committed or rolled back.

This is the syntax of the commit subcommand:

imqcmd commit txn -n *transactionID*

For example:

```
imqcmd commit txn -n 64248349708800
```

This is the syntax of the `rollback`. subcommand:

imqcmd rollback txn -n *transactionID*

See the `imq.transaction.autorollback` property in Table 14-2 on page 287 for more information.

It is also possible to configure the broker to automatically roll back transactions in the `PREPARED` state at broker startup.

Chapter 6

Managing Physical Destinations

This chapter explains how you use the `imqcmd` utility to manage physical destinations. A Message Queue message is routed to its consumer clients by way of a physical destination on a broker. The broker manages the memory and persistent storage associated with the physical destinations, and sets their behaviors.

In a cluster, you create a physical destination on one broker, and the cluster propagates that physical destination to all brokers. An application client can subscribe to a topic or consume from a queue that is on any broker in the cluster, because the brokers cooperate to route messages across the cluster. However, only the broker to which a message was originally produced manages persistence and acknowledgment for that message.

This chapter covers the following topics:

Table 13-5 provides full reference information about the `imqcmd` subcommands for managing physical destinations and accomplishing these tasks.

See the *Message Queue Technical Overview* for an introduction to physical destinations.

NOTE	A client application uses a `Destination` object whenever it interacts with a physical destination. For provider-independence and portability, clients typically use administrator-created destination objects, which are called destination administered objects. You can configure administered objects for use by client applications, as described in Chapter 8, “Managing Administered Objects.”

Using the Command Utility

You use the Message Queue Command utility (`imqcmd`) to manage physical destinations. The syntax of the `imqcmd` command is the same as when you use it for managing other broker services.

Full reference information about `imqcmd`, its subcommands, and its options, is available in Chapter 13, “Command Line Reference” on page 265.

Subcommands

Table 6-1 lists the `imqcmd` subcommands whose use is described in this chapter. For reference information about these subcommands, see “Physical Destination Management” on page 275.

Table 6-1 Physical Destination Subcommands for the Command Utility

Subcommand and Argument	Description
`compact dst`	Compacts the file-based data store for one or more physical destinations.
`create dst`	Creates a physical destination.
`destroy dst`	Destroys a physical destination.
`list dst`	Lists physical destinations on a broker.
`metrics dst`	Displays physical destination metrics.
`pause dst`	Pauses one or more physical destinations on a broker.
`purge dst`	Purges all messages on a physical destination without destroying the physical destination.

Table 6-1 Physical Destination Subcommands for the Command Utility *(Continued)*

Subcommand and Argument	Description
`query dst`	Queries and displays information on a physical destination.
`resume dst`	Resumes one or more paused physical destinations on a broker.
`update dst`	Updates properties of a destination.

Creating a Physical Destination

To create a physical destination, you use the `imqcmd create` subcommand. This is the syntax for the `create` subcommand:

`create dst -t` *destType* `-n` *destName* `[-o` *property*=*value*`] [-o` *property*=*value1*`]...`

When creating a physical destination, you specify the following:

- The physical destination type, `t` (topic) or `q` (queue).
- The physical destination name. The naming rules are as follows:
  - The name must contain only alphanumeric characters. It cannot contain spaces.
  - The name can begin with an alphabetic character, the underscore character (_) or the dollar sign ($). It cannot begin with the character string “mq.”
- Nondefault values for the physical destination’s properties.

You can also set properties when you update a physical destination.

Many physical destination properties affect broker memory resources and message flow. For example, you can specify the number of producers that can send to a physical destination, the number and size of the messages they can send, and the response that the broker should take when physical destination limits are reached. The limits are similar to brokerwide limits controlled by broker configuration properties.

The following properties are used for both queue destinations and topic destinations:

- `maxNumMsgs.` Specifies the maximum number of unconsumed messages allowed in the physical destination.
- `maxTotalMsgBytes`. Specifies the maximum total amount of memory (in bytes) allowed for unconsumed messages in the physical destination.

- `limitBehavior`. Specifies how the broker responds when a memory-limit threshold is reached.
- `maxBytesPerMsg`. Specifies the maximum size (in bytes) of any single message allowed in the physical destination.
- `maxNumProducers`. Specifies the maximum number of producers for the physical destination.
- `consumerFlowLimit`. Specifies the maximum number of messages to be delivered to a consumer in a single batch.
- `isLocalOnly`. Applies only to broker clusters. Specifies that a physical destination is not replicated on other brokers, and is limited to delivering messages only to local consumers (consumers connected to the broker on which the physical destination is created).
- `useDMQ`. Specifies whether a physical destination's dead messages are discarded or put on the dead message queue.

The following properties are used for queue destinations only:

- `maxNumActiveConsumers`. Specifies the maximum number of consumers that can be active in load-balanced delivery from a queue destination.)
- `maxNumBackupConsumers`. Specifies the maximum number of backup consumers that can take the place of active consumers, if any fail during load-balanced delivery from a queue destination.
- `localDeliveryPreferred`. Applies only to load-balanced queue delivery in broker clusters. Specifies that messages be delivered to remote consumers only if there are no consumers on the local broker.

See Chapter 15, "Physical Destination Property Reference" on page 313 for full reference information about physical destination properties.

For auto-created destinations, you set default property values in the broker's instance configuration file. Reference information on auto-create properties is located in Table 14-3 on page 289.

➤ To create a physical destination

- To create a queue destination, enter a command like the following:

  ```
  imqcmd create dst -n myQueue -t q -o "maxNumActiveConsumers=5"
  ```

- To create a topic destination, enter a command like the following:

  ```
  imqcmd create dst -n myTopic -t t -o "maxBytesPerMsg=5000"
  ```

Listing Physical Destinations

You can get information about a physical destination's current property values, about the number of producers or consumers associated with a physical destination, and about messaging metrics, such as the number and size of messages in the physical destination.

To find a physical destination about which you want to get information, list all physical destinations on a broker using the `list dst` subcommand. This is the syntax for the `list dst` subcommand:

```
list dst [-t destType] [-tmp]
```

The command lists physical destinations of the specified type. The value for the destination type (`-t`) option can be `q` (queue) or `t` (topic).

If you omit the destination type, physical destinations of all types are listed.

The `list dst` subcommand can optionally specify the type of destination to list or include temporary destinations (using the `-tmp` option). Temporary destinations are created by clients, normally for the purpose of receiving replies to messages sent to other clients.

For example, to get a list of all physical destinations on the broker running on `myHost` at port `4545`, enter the following command:

```
imqcmd list dst -b myHost:4545
```

Information for the dead message queue, `mq.sys.dmq`, is always displayed, in addition to any other physical destinations, unless you specify the destination type `t` to include only topics.

Displaying Information about Physical Destinations

To get information about a physical destination's current properties, use the `query dst` subcommand. This is the syntax of the `query dst` subcommand:

```
query dst -t destType -n destName
```

The command lists information about the destination of the specified type and name. For example, the following command displays information about the queue destination `XQueue`:

```
imqcmd query dst -t q -n XQueue -u admin
```

The command produces output like the following:

```
------------------------------------
Destination Name    Destination Type
------------------------------------
XQueue              Queue

On the broker specified by:

-------------------------
Host         Primary Port
-------------------------
localhost    7676

Destination Name                      XQueue
Destination Type                      Queue
Destination State                     RUNNING
Created Administratively              true

Current Number of Messages            0
Current Total Message Bytes           0
Current Number of Producers           0
Current Number of Active Consumers    0
Current Number of Backup Consumers    0

Max Number of Messages                unlimited (-1)
Max Total Message Bytes               unlimited (-1)
Max Bytes per Message                 unlimited (-1)
Max Number of Producers               100
Max Number of Active Consumers        1
Max Number of Backup Consumers        0

Limit Behavior                        REJECT_NEWEST
Consumer Flow Limit                   1000
Is Local Destination                  false
Local Delivery is Preferred           false
Use Dead Message Queue                true
```

The output also shows the number of producers and consumers associated with the destination. For queue destinations, the number includes active consumers and backup consumers.

You can use the `update dst` subcommand to change the value of one or more properties (see "Updating Physical Destination Properties" on page 123).

Updating Physical Destination Properties

You can change the properties of a physical destination by using the `update dst` subcommand and the `-o` option to specify the property to update. This is the syntax for the `update dst` subcommand:

`update dst -t` *destType* `-n` *destName* `-o` *property=value* `[[-o` *property=value1*`]`*...]*

The command updates the value of the specified properties at the specified destination. The property name can be any property listed in Table 15-1.

You can use the `-o` option multiple times to update multiple properties. For example, the following command changes the `maxBytesPerMsg` property to `1000` and the `MaxNumMsgs` property to `2000`:

```
imqcmd update dst -t q -n myQueue -o "maxBytesPerMsg=1000"
             -o "maxNumMsgs=2000" -u admin
```

See Chapter 15, "Physical Destination Property Reference" for a list of the properties that you can update.

You cannot use the `update dst` subcommand to update the *type* of a physical destination or to update the `isLocalOnly` property.

NOTE The dead message queue is a specialized physical destination whose properties differ somewhat from those of other destinations. For more information, see "Configuring Use of the Dead Message Queue" on page 128.

Pausing and Resuming Physical Destinations

You can pause a physical destination to control the delivery of messages from producers to the destination, or from the destination to consumers, or both. In particular, you can pause the flow of messages into a destination to help prevent destinations from being overwhelmed with messages when production of messages is much faster than consumption. You must pause a physical destination before compacting it.

To pause the delivery of messages to or from a physical destination, use the `pause dst` subcommand. This is the syntax of the `pause dst` subcommand:

`pause dst [-t` *destType* `-n` *destName*`] [-pst` *pauseType*`]`

The subcommand pauses the delivery of messages to consumers (`-pst CONSUMERS`), or from producers (`-pst PRODUCERS`), or both (`-pst ALL`), for the destination of the specified type and name. If no destination type and name are specified, all physical destinations are paused. The default is `ALL`.

Example:

```
imqcmd pause dst -n myQueue -t q -pst PRODUCERS -u admin
imqcmd pause dst -n myTopic -t t -pst CONSUMERS -u admin
```

To resume delivery to a paused destination, use the `resume dst` subcommand. This is the syntax of the `resume dst` subcommand:

`resume dst [-t` *destType* `-n` *destName*`]`

The subcommand resumes delivery of messages to the paused destination of the specified type and name. If no destination type and name are specified, all destinations are resumed.

Example:

```
imqcmd resume dst -n myQueue -t q
```

In a broker cluster, instances of the physical destination reside on each broker in the cluster. You must pause each one individually.

Purging Physical Destinations

You can purge all messages currently queued at a physical destination. Purging a physical destination means that all messages stored at the destination are deleted.

You might want to purge messages when the accumulated messages are taking up too much of the system's resources. This might happen when a queue does not have registered consumer clients and is receiving many messages. It might also happen if inactive durable subscribers to a topic do not become active. In both cases, messages are held unnecessarily.

To purge messages at a physical destination, use the `purge dst` subcommand. This is the syntax of the `purge dst` subcommand:

`purge dst -t` *destType* `-n` *destName*

The subcommand purges messages at the physical destination of the specified type and name.

Examples:

```
imqcmd purge dst -n myQueue -t q -u admin
```

```
imqcmd purge dst -n myTopic -t t -u admin
```

If you have shut down the broker and do not want old messages to be delivered when you restart it, use the `-reset messages` option to purge stale messages; for example:

```
imqbrokerd -reset messages -u admin
```

This saves you the trouble of purging destinations after restarting the broker.

In a broker cluster, instances of the physical destination reside on each broker in the cluster. You must purge each of these destinations individually.

Destroying Physical Destinations

To destroy a physical destination, use the `destroy dst` subcommand. This is the syntax of the `destroy dst` subcommand:

`destroy dst -t` *destType* `-n` *destName*

The subcommand destroys the physical destination of the specified type and name.

Example:

```
imqcmd destroy dst -t q -n myQueue -u admin
```

Destroying a physical destination purges all messages at that destination and removes it from the broker; the operation is not reversible.

You cannot destroy the dead message queue.

Compacting Physical Destinations

If you are using a file-based data store as the persistent store for messages, you can monitor disk utilization and compact the disk when necessary.

The file-based message store is structured so that messages are stored in directories corresponding to the physical destinations in which they are being held. In each physical destination's directory, most messages are stored in one file consisting of variable-sized records, the variable-sized record file. (To alleviate fragmentation, messages whose size exceeds a configurable threshold are stored in their own individual files.)

As messages of varying sizes are persisted and then removed from the variable-sized record file, holes may develop in the file where free records are not being re-used.

To manage unused free records, the Command utility includes subcommands for monitoring disk utilization per physical destination and for reclaiming free disk space when utilization drops.

Monitoring a Physical Destination's Disk Utilization

To monitor a physical destination's disk utilization, use a command like the following:

```
imqcmd metrics dst -t q -n myQueue -m dsk -u admin
```

This command produces output like the following:

```
--------------------------------------
Reserved   Used      Utilization Ratio
--------------------------------------
806400     804096    99
1793024    1793024   100
2544640    2518272   98
```

The columns in the subcommand output have the following meaning:

Table 6-2 Physical Destination Disk Utilization Metrics

Metric	Description
Reserved	Disk space in bytes used by all records, including records that hold active messages and free records waiting to be reused.
Used	Disk space in bytes used by records that hold active messages.
Utilization Ratio	Quotient of used disk space divided by reserved disk space. The higher the ratio, the more the disk space is being used to hold active messages.

Reclaiming Unused Physical Destination Disk Space

The disk utilization pattern depends on the characteristics of the messaging application that uses a particular physical destination. Depending on the relative flow of messages into and out of a physical destination, and the relative size of messages, the reserved disk space might grow over time.

If the message producing rate is greater than the message consuming rate, free records should generally be reused and the utilization ratio should be on the high side. However, if the message producing rate is similar to or smaller than the message consuming rate, you can expect that the utilization ratio will be low.

In general, you want the reserved disk space to stabilize and the utilization to remain high. As a rule, if the system reaches a steady state in which the amount of reserved disk space generally stays constant and utilization rate is high (above 75%), there is no need to reclaim the unused disk space. If the system reaches a steady state and utilization rate is low (below 50%), you can compact the disk to reclaim the disk space occupied by free records.

Use the `compact dst` subcommand to compact the data store. This is the syntax for the `compact dst` subcommand:

```
compact dst [-t destType -n destName]
```

The subcommand compacts the file-based data store for the physical destination of the specified type and name. If no destination type and name are specified, all destinations are compacted. Physical destinations must be paused before they can be compacted.

If the reserved disk space continues to increase over time, reconfigure the destination's memory management by setting destination memory limit properties and limit behaviors (see Table 15-1 on page 313).

➤ To Reclaim Unused Physical Destination Disk Space

1. Pause the destination.

```
imqcmd pause dst -t q -n myQueue -u admin
```

2. Compact the disk.

```
imqcmd compact dst -t q -n myQueue -u admin
```

3. Resume the physical destination.

```
imqcmd resume dst -t q -n myQueue -u admin
```

If destination type and name are not specified, these operations are performed for *all* physical destinations.

Configuring Use of the Dead Message Queue

The dead message queue, `mq.sys.dmq`, is a system-created physical destination that holds the dead messages of a broker and its other physical destinations. The dead message queue is a tool for monitoring, tuning system efficiency, and troubleshooting. For a definition of the term "dead message" and a more detailed introduction to the dead message queue, see the *Message Queue Technical Overview*.

The broker automatically creates a dead message queue when it starts. The broker places messages on the queue if it cannot process them, or if their time-to-live has expired. In addition, other physical destinations can use the dead message queue to hold discarded messages. Use of the dead message queue provides information that is useful for troubleshooting the system.

Configuring Use of the Dead Message Queue

By default, a physical destination is configured to use the dead message queue. You can disable a physical destination from using the dead message queue, or enable it to do so, by setting the physical destination property `useDMQ`.

The following example creates a queue called `myDist` that uses the dead message queue by default:

```
imqcmd create dst -n -myDist -t q
```

The following example disables use of the dead message queue for the same queue:

```
imqcmd update dst -n myDist -t q -o useDMQ=false
```

You can enable all auto-created physical destinations on a broker to use the dead message queue, or disable them from doing so, by setting the `imq.autocreate.destination.useDMQ` broker property.

Configuring and Managing the Dead Message Queue

You can use the Message Queue Command utility (`imqcmd`) to manage the dead message queue as you manage other queues, with some differences. For example, because the dead message queue is system-created, you cannot create, pause, or destroy it. Also, as shown in Table 6-3, default values for the dead message queue sometimes differ from those of normal queues.

Dead Message Queue Properties

You configure the dead message queue as you configure other queues, but certain physical destination properties do not apply or have different default values. Table 6-3 lists queue properties that the dead message queue handles in a unique way.

Table 6-3 Dead Message Queue Treatment of Standard Physical Destination Properties

Property	Unique Treatment by Dead Message Queue
`limitBehavior`	The default value for the dead message queue is `REMOVE_OLDEST`. (The default value for other queues is `REJECT_NEWEST`.) Flow control is not supported on the dead message queue.
`localDeliveryPreferred`	Does not apply to the dead message queue.
`maxNumMsgs`	The default value for the dead message queue is `1000`. The default value for other queues is `-1` (unlimited).
`maxNumProducers`	Does not apply to the dead message queue.
`maxTotalMsgBytes`	The default value for the dead message queue is 10 MB. The default value for other queues is `-1` (unlimited).
`isLocalOnly`	In a broker cluster, a dead message queue is always a local physical destination and this property is permanently set to true. However, a local broker's dead message queue can contain messages produced by clients of other brokers in the cluster, if the local broker marks the messages as dead.

Message Contents

A broker can place an entire message on the dead message queue, or it can discard the message body contents, retaining just the header and property data. By default, the dead message queue stores entire messages.

If you want to reduce the size of the dead message queue and if you do not plan to restore dead messages, consider setting the `imq.destination.DMQ.truncateBody` broker property to `true`:

```
imqcmd update bkr -o imq.destination.DMQ.truncateBody=true
```

This will discard the message body and retain only the headers and property data.

Enabling Dead Message Logging

Dead message logging is disabled by default. Enabling dead message logging allows the broker to log the following events:

- The broker moves a message to the dead message queue
- The broker discards a message from the dead message queue and from any physical destination that does not use the dead message queue
- A physical destination reaches its limits

The following command enables dead message logging:

```
imqcmd update bkr -o imq.destination.logDeadMsgs=true
```

Dead message logging applies to all physical destinations that use the dead message queue. You cannot enable or disable logging for an individual physical destination.

Chapter 7

Managing Security

As administrator, you configure a user repository to authenticate users, to define access control, to configure a Secure Socket Layer (SSL) connection service that encrypts client-broker communication and to set up a password file for use in broker startup.

The chapter includes the following sections:

Authenticating Users

When a user attempts to connect to the broker, the broker authenticates the user by inspecting the name and password provided. The broker grants the connection if the name and password match those in a broker-specific user repository that each broker is configured to consult.

You are responsible for maintaining a list of users, their groups, and their passwords in a user repository. You can use a different user repository for each broker instance. This section explains how you create, populate, and manage that repository.

The repository can be one of the following types:

- A flat-file repository that is shipped with Message Queue

  This type of user repository is very easy to use. You can populate and manage the repository using the User Manager utility (`imqusermgr`). To enable authentication, you populate the user repository with each user's name and password and the name of the user's group.

  For more information on setting up and managing the user repository, see "Using a Flat-File User Repository."

- An LDAP server

  This could be an existing or new LDAP directory server that uses the LDAP v2 or v3 protocol. It is not as easy to use as the flat-file repository, but it is more scalable, and therefore better for production environments.

  If you are using an LDAP user repository, you use the tools provided by the LDAP vendor to populate and manage the user repository. For more information, see "Using an LDAP Server for a User Repository" on page 139.

Using a Flat-File User Repository

Message Queue provides a flat-file user repository and a command line tool, the User Manager utility (`imqusermgr`), that you can use to populate and manage the flat-file user repository. The following sections describe the flat-file user repository and how you use the User Manager utility to populate and manage that repository.

Creating a User Repository

The flat-file user repository is instance-specific. A default user repository (named `passwd`) is automatically created for each broker instance that you start. This user repository is placed in a directory identified by the name of the broker instance with which the repository is associated (see Appendix A, "Platform-Specific Locations of Message Queue Data"):

…`/instances/`*instanceName*`/etc/passwd`

The repository is created with two entries. Each row of Table 7-1 shows an entry.

Table 7-1 Initial Entries in User Repository

User Name	Password	Group	State
`admin`	`admin`	`admin`	`active`
`guest`	`guest`	`anonymous`	`active`

These initial entries allow the Message Queue broker to be used immediately after installation without intervention by the administrator:

- The initial `guest` user entry allows clients to connect to a broker instance using the default `guest` user name and password.
- The initial `admin` user entry lets you use `imqcmd` commands to administer a broker instance using the default `admin` user name and password. You should update this initial entry to change the password (see "Changing the Default Administrator Password" on page 138).

The following sections explain how you populate and manage a flat-file user repository.

User Manager Utility

The Message Queue User Manager utility (`imqusermgr`) lets you edit or populate a flat-file user repository. This section introduces the User Manager utility. Subsequent sections explain how you use the `imqusermgr` subcommands to accomplish specific tasks.

For full reference information about the `imqusermgr` command, see Chapter 13, "Command Line Reference."

Before using the User Manager, keep the following things in mind:

- If a broker-specific user repository does not yet exist, you must start up the corresponding broker instance to create it.
- The `imqusermgr` command has to be run on the host where the broker is installed.
- You must have appropriate permissions to write to the repository,: namely, on Solaris and Linux, you must be the root user or the user who first created the broker instance.

NOTE Examples in the following sections assume the default broker instance.

Subcommands

The `imqusermgr` command has the subcommands `add`, `delete`, `list`, and `update`.

- The `add` subcommand adds a user and associated password to the specified (or default) broker instance repository, and optionally specifies the user's group. The subcommand syntax is as follows:

  `add [-i` *instanceName*`] -u` *userName* `-p` *passwd* `[-g` *group*`] [-s]`

- The `delete` subcommand deletes the specified user from the specified (or default) broker instance repository. The subcommand syntax is as follows:

  `delete [-i` *instanceName*`] -u` *userName* `[-s] [-f]`

- The `list` subcommand displays information about the specified user or all users in the specified (or default) broker instance repository. The subcommand syntax is as follows:

  `list [-i` *instanceName*`] [-u` *userName*`]`

- The `update` subcommand updates the password and/or state of the specified user in the specified (or default) broker instance repository. The subcommand syntax is as follows:

  `update [-i` *instanceName*`] -u` *userName* `-p` *passwd* `[-a` *state*`] [-s] [-f]`

  `update [-i` *instanceName*`] -u` *userName* `-a` *state* `[-p` *passwd*`] [-s] [-f]`

Command Options

Table 7-2 lists the options to the `imqusermgr` command.

Table 7-2 `imqusermgr` Options

Option	**Description**
`-a` *activeState*	Specifies (`true`/`false`) whether the user's state should be active. A value of `true` means that the state is active. This is the default.
`-f`	Performs action without user confirmation.
`-h`	Displays usage help. Nothing else on the command line is executed.
`-i` *instanceName*	Specifies the broker instance name to which the command applies. If not specified, the default instance name, `imqbroker`, is assumed.
`-p` *passwd*	Specifies the user's password.
`-g` *group*	Specifies the user group. Valid values are `admin`, `user`, `anonymous`.
`-s`	Sets silent mode.
`-u` *userName*	Specifies the user name.
`-v`	Displays version information. Nothing else on the command line is executed.

Groups

When adding a user entry to the user repository for a broker instance, you can specify one of three predefined groups: `admin`, `user`, or `anonymous`. If no group is specified, the default group `user` is assigned. Groups should be assigned as follows:

- **`admin` group**. For broker administrators. Users who are assigned this group can, by default, configure, administer, and manage the broker. You can assign more than one user to the `admin` group.

- **`user` group**. For normal (non-administration) Message Queue client users. Most client users are in the `user` group. By default, users in this group can produce messages to all topics and queues, consume messages from all topics and queues, and browse messages in any queue.

- **anonymous group**. For Message Queue clients that do not want a user name that is known to the broker, possibly because the client application does not know of a real user name to use. This account is analogous to the anonymous account present in most FTP servers. You can assign only one user at a time to the anonymous group. You should restrict the access privileges of this group as compared to the user group or you should remove users from the group at deployment time.

To change a user's group, you must delete the user entry and then add another entry for the user, specifying the new group.

You cannot rename or delete these system-created groups, or create new groups. However, you can specify access rules that define the operations that the members of that group can perform. For more information, see "Authorizing Users: The Access Control Properties File" on page 142.

User States

When you add a user to a repository, the user's state is active by default. To make the user inactive, you must use the update command. For example, the following command makes the user JoeD inactive:

```
imqusermgr update -u JoeD -a false
```

Entries for users that have been rendered inactive are retained in the repository; however, inactive users cannot open new connections. If a user is inactive and you add another user who has the same name, the operation will fail. You must delete the inactive user entry or change the new user's name or use a different name for the new user. This prevents you from adding duplicate user names.

Format of User Names and Passwords

User names and passwords must follow these guidelines:

- A user name cannot contain an asterisk (*), comma (,), colon (:), or a new-line or carriage-return character.
- A user name or password must be at least one character long.
- If a user name or password contains a space, the entire name or password must be enclosed in quotation marks.
- There is no limit on the length of passwords or user names, except for command shell restrictions on the maximum number of characters that can be entered on a command line.

Populating and Managing a User Repository

Use the `add` subcommand to add a user to a repository. For example, the following command adds the user `Katharine` with the password `sesame` to the default broker instance user repository.

```
imqusermgr add -u Katharine -p sesame -g user
```

Use the `delete` subcommand to delete a user from a repository. For example, the following command deletes the user, `Bob`:

```
imqusermgr delete -u Bob
```

Use the `update` subcommand to change a user's password or state. For example, the following command changes Katharine's password to `aladdin`:

```
imqusermgr update -u Katharine -p aladdin
```

To list information about one user or all users, use the `list` command. The following command shows information about the user named `isa`:

```
imqusermgr list -u isa
```

```
% imqusermgr list -u isa

User repository for broker instance: imqbroker
---------------------------------
User Name    Group    Active State
---------------------------------
isa          admin    true
```

The following command lists information about all users:

```
imqusermgr list
```

```
% imqusermgr list

User repository for broker instance: imqbroker
--------------------------------------
User Name    Group        Active State
--------------------------------------
admin        admin        true
guest        anonymous    true
isa          admin        true
testuser1    user         true
testuser2    user         true
testuser3    user         true
testuser4    user         false
testuser5    user         false
```

Changing the Default Administrator Password

For the sake of security, you should change the default password of `admin` to one that is known only to you. The following command changes the default administrator password for the `mybroker` broker instance from `admin` to `grandpoobah`.

```
imqusermgr update mybroker -u admin -p grandpoobah
```

You can quickly confirm that this change is in effect by running any of the command line tools when the broker instance is running. For example, the following command will prompt you for a password:

```
imqcmd list svc mybroker -u admin
```

Entering the new password (`grandpoobah`) should work; the old password should fail.

After changing the password, you should supply the new password any time you use any of the Message Queue administration tools, including the Administration Console.

Using an LDAP Server for a User Repository

To use an LDAP server for a user repository, you perform the following tasks:

- Editing the instance configuration file
- Setting up access control for administrators

Editing the Instance Configuration File

To have a broker use a directory server, you set the values for certain properties in the broker instance configuration file, `config.properties`. These properties enable the broker instance to query the LDAP server for information about users and groups whenever a user attempts to connect to the broker instance or perform messaging operations.

The instance configuration file is located in a directory under the broker instance directory. The path has the following format:

…`/instances/`*instanceName*`/props/config.properties`

For information about the operating system-specific location of instance directories, see Appendix A, "Platform-Specific Locations of Message Queue Data."

➤ To Edit the Configuration File to Use an LDAP Server

1. Specify that you are using an LDAP user repository by setting the following property:

   ```
   imq.authentication.basic.user_repository=ldap
   ```

2. Set the `imq.authentication.type` property to determine whether a password should be passed from client to broker in base-64 (`basic`) or MD5 (`digest`) encoding. When using an LDAP directory server for a user repository, you must set the authentication type to `basic`. For example,

   ```
   imq.authentication.type=basic
   ```

3. You must also set the broker properties that control LDAP access. These properties are stored in a broker's instance configuration file. The properties are discussed under "Security Services" on page 83 and summarized under "Security Properties" on page 298.

   Message Queue uses JNDI APIs to communicate with the LDAP directory server. Consult JNDI documentation for more information on syntax and on terms referenced in these properties. Message Queue uses a Sun JNDI LDAP provider and uses simple authentication.

   Message Queue supports LDAP authentication failover: you can specify a list of LDAP directory servers for which authentication will be attempted (see the reference information for the `imq.user.repos.ldap.server` property).

   See the broker's `config.properties` file for a sample of how to set properties related to LDAP user-repository.

4. If necessary, you need to edit the users/groups and rules in the access control properties file. For more information about the use of access control property files, see "Authorizing Users: The Access Control Properties File" on page 142.

5. If you want the broker to communicate with the LDAP directory server over SSL during connection authentication and group searches, you need to activate SSL in the LDAP server and then set the following properties in the broker configuration file:

   - Specify the port used by the LDAP server for SSL communications. For example:

     ```
     imq.user_repository.ldap.server=myhost:7878
     ```

   - Set the broker property `imq.user_repository.ldap.ssl.enabled` to `true`.

     When employing multiple LDAP directory servers, use `ldap://` to specify each additional directory server. For example:

     `imq.user_repository.ldap.server=`*myHost*`:7878ldap://`*otherHost*`:7878..`

     Separate each additional directory server with a space. All directory servers in the list must use the same values for other LDAP-related properties.

Setting Up Access Control for Administrators

To create administrative users, you use the access control properties file to specify users and groups that can create `ADMIN` connections. These users and groups must be predefined in the LDAP directory.

Any user or group who can create an `ADMIN` connection can issue administrative commands.

➤ To Set Up an Administrative User

1. Enable the use of the access control file by setting the broker property `imq.accesscontrol.enabled` to `true`, which is the default value.

   The `imq.accesscontrol.enabled` property enables use of the access control file.

2. Open the access control file, `accesscontrol.properties`. The location for the file is listed in Appendix A, "Platform-Specific Locations of Message Queue Data."

   The file contains an entry such as the following:

   ```
   service connection access control
   ##################################
   connection.NORMAL.allow.user=*
   connection.ADMIN.allow.group=admin
   ```

   The entries listed are examples. Note that the `admin` group exists in the file-based user repository but does not exist by default in the LDAP directory. You must substitute the name of a group that is defined in the LDAP directory, to which you want to grant Message Queue administrator privileges.

3. To grant Message Queue administrator privileges to users, enter the user names as follows:

   `connection.ADMIN.allow.user=`*userName*[[`,`*userName2*]…]

4. To grant Message Queue administrator privileges to groups, enter the group names as follows:

   `connection.ADMIN.allow.group=`*groupName*[[`,`*groupName2*]…]

Authorizing Users: The Access Control Properties File

An *access control properties file* (ACL file) contains rules that specify the operations that users and groups of users can perform. You edit the ACL file to restrict operations to certain users and groups. You can use a different ACL file for each broker instance.

The ACL file is used whether user information is placed in a flat-file user repository or in an LDAP user repository. A broker checks its ACL file when a client application performs one of the following operations:

- Creates a connection
- Creates a producer
- Creates a consumer
- Browses a queue

The broker checks the ACL file to determine whether the user that generated the request, or a group to which the user belongs, is authorized to perform the operation.

If you edit an ACL file, the new settings take effect the next time the broker checks the file to verify authorization. You need not restart the broker after editing the file.

Creating an Access Control Properties File

The ACL file is instance specific. Each time you start a broker instance, a default file named `accesscontrol.properties` is created in the instance directory. The path to the file has the following format (see Appendix A, "Platform-Specific Locations of Message Queue Data"):

…/instances/*brokerInstanceName*/etc/accesscontrol.properties

The ACL file is formatted like a Java properties file. It starts by defining the version of the file and then specifies access control rules in three sections:

- Connection access control
- Physical destination access control
- Physical destination auto-create access control

The `version` property defines the version of the ACL properties file; you may not change this entry.

```
version=JMQFileAccessControlModel/100
```

The three sections of the ACL file that specify access control are described below, following a description of the basic syntax of access rules and an explanation of how permissions are calculated.

Syntax of Access Rules

In the ACL properties file, access control defines what access specific users or groups have to protected resources like physical destinations and connection services. Access control is expressed by a rule or set of rules, with each rule presented as a Java property:

The basic syntax of these rules is as follows:

resourceType.*resourceVariant*.*operation*.*access*.*principalType*=*principals*

Table 7-3 describes the elements of syntax rules.

Table 7-3 Syntactic Elements of Access Rules

Element	Description
resourceType	One of the following: `connection`, `queue` or `topic`.
resourceVariant	An instance of the type specified by *resourceType*. For example, myQueue. The wild card character (*) may be used to mean all connection service types or all physical destinations.
operation	Value depends on the kind of access rule being formulated.
access	One of the following: `allow` or `deny`.
principalType	One of the following: `user` or `group`. For more information, see "Groups" on page 135.
principals	Who may have the access specified on the left-hand side of the rule. This may be an individual user or a list of users (comma delimited) if the `principalType` is `user`; it may be a single group or a list of groups (comma delimited list) if the `principalType` is `group`. The wild card character (*) may be used to represent all users or all groups.

Here are some examples of access rules:

- The following rule means that all users may send a message to the queue named q1.

  ```
  queue.q1.produce.allow.user=*
  ```

- The following rule means that any user may send messages to any queue.

  ```
  queue.*.produce.allow.user=*
  ```

NOTE To specify non-ASCII user, group, or destination names, use Unicode escape (`\uXXXX`) notation. If you have edited and saved the ACL file with these names in a non-ASCII encoding, you can convert the file to ASCII with the Java `native2ascii` tool. For more detailed information, see

`http://java.sun.com/j2se/1.4/docs/guide/intl/faq.html`

How Permissions are Computed

When there are multiple access rules in the file, permissions are computed as follows:

- Specific access rules override general access rules. After applying the following two rules, all users can send to all queues, but Bob cannot send to `tq1`.

  ```
  queue.*.produce.allow.user=*
  queue.tq1.produce.deny.user=Bob
  ```

- Access given to an explicit *principal* overrides access given to a * *principal*. The following rules deny Bob the right to produce messages to `tq1`, but allow everyone else to do it.

  ```
  queue.tq1.produce.allow.user=*
  queue.tq1.produce.deny.user=Bob
  ```

- The * *principal* rule for users overrides the corresponding * *principal* for groups. For example, the following two rules allow all authenticated users to send messages to `tq1`.

  ```
  queue.tq1.produce.allow.user=*
  queue.tq1.produce.deny.group=*
  ```

- Access granted a user overrides access granted to the user's group. In the following example, even if Bob is a member of User, he cannot produce messages to `tq1`. All other members of User will be able to do so.

  ```
  queue.tq1.produce.allow.group=User
  queue.tq1.produce.deny.user=Bob
  ```

- Any access permission not explicitly granted through an access rule is implicitly denied. For example, if the ACL file contains no access rules, all users are denied all operations.

- Deny and allow permissions for the same user or group cancel themselves out. For example, the following two rules cause Bob to be unable to browse `q1`:

  ```
  queue.q1.browse.allow.user=Bob
  queue.q1.browse.deny.user=Bob
  ```

  The following two rules prevent the group User from consuming messages at `q5`.

  ```
  queue.q5.consume.allow.group=User
  queue.q5.consume.deny.group=User
  ```

- When multiple same left-hand rules exist, only the last entry takes effect.

Access Control for Connection Services

The connection access control section in the ACL properties file contains access control rules for the broker's connection services. The syntax of connection access control rules is as follows:

`connection.`*resourceVariant*`.`*access*`.`*principalType*`=`*principals*

Two values are defined for *resourceVariant*: `NORMAL` and `ADMIN`. These predefined values are the only types of connection services to which you can grant access.

The default ACL properties file gives all users access to `NORMAL` connection services and gives users in the group `admin` access to `ADMIN` connection services:

```
connection.NORMAL.allow.user=*
connection.ADMIN.allow.group=admin
```

If you are using a file-based user repository, the default group `admin` is created by the User Manager utility. If you are using an LDAP user repository, you can do one of the following to use the default ACL properties file:

- Define a group called `admin` in the LDAP directory.
- Replace the name `admin` in the ACL properties file with the names of one or more groups that are defined in the LDAP directory.

You can restrict connection access privileges. For example, the following rules deny Bob access to `NORMAL` but allow everyone else:

```
connection.NORMAL.deny.user=Bob
connection.NORMAL.allow.user=*
```

You can use the asterisk (*) character to specify all authenticated users or groups.

The way that you use the ACL properties file to grant access to `ADMIN` connections differs for file-based user repositories and LDAP user repositories, as follows:

- **File-based user repository**
    - If access control is disabled, users in the group `admin` have `ADMIN` connection privileges.
    - If access control is enabled, edit the ACL file. Explicitly grant users or groups access to the `ADMIN` connection service.
- **LDAP user repository.** If you are using an LDAP user repository, do all of the following:
    - Enable access control.
    - Edit the ACL file and provide the names of users or groups who can make `ADMIN` connections. Specify any users or groups that are defined in the LDAP directory server.

Access Control for Physical Destinations

The destination access control section of the access control properties file contains physical destination-based access control rules. These rules determine who (users/groups) may do what (operations) where (physical destinations). The types of access that are regulated by these rules include sending messages to a queue, publishing messages to a topic, receiving messages from a queue, subscribing to a topic, and browsing messages in a queue.

By default, any user or group can have all types of access to any physical destination. You can add more specific destination access rules or edit the default rules. The rest of this section explains the syntax of physical destination access rules, which you must understand to write your own rules.

The syntax of destination rules is as follows:

resourceType.*resourceVariant*.*operation*.*access*.*principalType*=*principals*

Table 7-4 describes these elements:

Table 7-4 Elements of Physical Destination Access Control Rules

Component	Description
resourceType	Can be `queue` or `topic`.
resourceVariant	A physical destination name or all physical destinations (*), meaning all queues or all topics.
operation	Can be `produce`, `consume`, or `browse`.
access	Can be `allow` or `deny`.
principalType	Can be `user` or `group`.

Access can be given to one or more users and/or one or more groups.

The following examples illustrate different kinds of physical destination access control rules:

- Allow all users to send messages to any queue destinations.

  ```
  queue.*.produce.allow.user=*
  ```

- Deny any member of the group `user` the ability to subscribe to the topic `Admissions`.

  ```
  topic.Admissions.consume.deny.group=user
  ```

Access Control for Auto-Created Physical Destinations

The final section of the ACL properties file, includes access rules that specify for which users and groups the broker will auto-create a physical destination.

When a user creates a producer or consumer at a physical destination that does not already exist, the broker will create the destination if the broker's auto-create property has been enabled.

By default, any user or group has the privilege of having a physical destination auto-created by the broker. This privilege is specified by the following rules:

```
queue.create.allow.user=*
```

```
topic.create.allow.user=*
```

You can edit the ACL file to restrict this type of access.

The general syntax for physical destination auto-create access rules is as follows:

resourceType`.create.`*access*`.`*principalType*`=`*principals*

Where *resourceType* is either `queue` or `topic`.

For example, the following rules allow the broker to auto-create topic destinations for everyone except Snoopy.

```
topic.create.allow.user=*
```

```
topic.create.deny.user=Snoopy
```

Note that the effect of physical destination auto-create rules must be congruent with that of physical destination access rules. For example, if you 1) change the destination access rule to forbid any user from sending a message to a destination but 2) enable the auto-creation of the destination, the broker *will* create the physical destination if it does not exist but it will *not* deliver a message to it.

Working With an SSL-Based Service

A connection service that is based on the Secure Socket Layer (SSL) standard sends encrypted messages sent between clients and broker. This section explains how to set up an SSL-based connection service.

Message Queue supports the following connection services that are based on the Secure Socket Layer (SSL) standard:

- `ssljms`, `ssladmin`, and `cluster` are used over TCP/IP.
- `httpsjms` is used over HTTP.

These connection services allow for the encryption of messages sent between clients and broker. Message Queue supports SSL encryption based on either self-signed server certificates or signed certificates.

To use an SSL-based connection service, you generate a private key / public key pair using the Key Tool utility (`imqkeytool`). This utility embeds the public key in a self-signed certificate that is passed to any client requesting a connection to the broker, and the client uses the certificate to set up an encrypted connection.

While Message Queue's SSL-based connection services are similar in concept, there are some differences in how you set them up.

The rest of this section describes how to set up secure connections over TCP / IP.

The SSL-based connection service for user over HTTP, `httpsjms`, lets a client and broker establish a secure connection by way of an HTTPS tunnel servlet. For information on setting up secure connections over HTTP, see Appendix C, "HTTP / HTTPS Support" on page 355.

Secure Connection Services for TCP/IP

The following SSL-based connection services provide a direct, secure connection over TCP / IP:

- The `ssljms` service delivers messages over a secure, encrypted connection between a client and broker.
- The `ssladmin` service creates a secure, encrypted connection between the Message Queue Command utility (`imqcmd`) and a broker. A secure connection is not supported for the Administration Console (`imqadmin`).
- The `cluster` service delivers messages and provides inter-broker communication over a secure, encrypted connection between brokers in a cluster (see "Secure Connections Between Brokers" on page 184).

Configuring the Use of Self-Signed Certificates

This section describes how to set up an SSL-based service using self-signed certificates.

For a stronger level of authentication, you can use signed certificates that are verified by a certificate authority. First follow the steps in this section and then go to "Configuring the Use of Signed Certificates" on page 155 to perform additional steps.

➤ **To Set Up an SSL-based Connection Service**

1. Generate a self-signed certificate.

2. Enable the `ssljms`, `ssladmin`, or `cluster` connection service in the broker.
3. Start the broker.
4. Configure and run the client (applies only to `ssljms` connection service).

The procedures for setting up `ssljms` and `ssladmin` connection services are identical, except for Step 4, configuring and running the client.

Each of the steps is discussed in some detail in the sections that follow.

Step 1. Generating a Self-Signed Certificate

Message Queue SSL support with self-signed certificates is oriented toward securing on-the-wire data with the assumption that the client is communicating with a known and trusted server.

Run the Key Tool utility to generate a self-signed certificate for the broker. On UNIX® systems you may need to run Key Tool as the superuser (`root`) in order to have permission to create the key store.

The same certificate can be used for the `ssljms`, `ssladmin`, or `cluster` connection service.

Enter the following at the command prompt:

```
imqkeytool -broker
```

The utility prompts you for a key store password.

```
Generating keystore for the broker ...
Enter keystore password:
```

Next, the utility prompts for information that identifies the broker whose certificate this is. The information that you supply will make up an X.500 distinguished name. The following table lists the prompts, describes them, and provides an example for each prompt. Values are case-insensitive and can include spaces.

Table 7-5 Distinguished Name Information Required for a Self-Signed Certificate

Prompt	Description	Example
What is your first and last name?	The X.500 commonName (CN). Enter the fully qualified name of the server that is running the broker.	`myhost.sun.com`
What is the name of your organizational unit?	The X.500 organizationalUnit (OU). Enter the name of a department or division.	`purchasing`

Table 7-5 Distinguished Name Information Required for a Self-Signed Certificate *(Continued)*

Prompt	Description	Example
What is the name of your organization?	The X.500 organizationName (ON). Name of a larger organization, such as a company or government entity.	`My Company, Inc.`
What is the name of your city or locality?	The X.500 localityName (L).	`San Francisco`
What is the name of your state or province?	The X.500 stateName (ST). Enter the full name of the state or province, without abbreviating.	`California`
What is the two-letter country code for this unit?	The X.500 country (C).	`US`

When you have entered the information, Key Tool displays it for confirmation. For example:

```
Is CN=mqserver.sun.com, OU=purchasing, O=My Company, Inc., L=San
Francisco, ST=California, C=US correct?
```

To re-enter values, accept the default or enter `no`; to accept the current values and proceed, enter `yes`. After you confirm, Key Tool pauses while it generates a key pair.

Next, Key Tool asks for a password to lock the particular key pair (key password). Enter Return in response to this prompt to use the same password as the key password and key store password.

NOTE Remember the password you provide. You must provide this password when you start the broker, to allow the broker to open the key store. You can store the key store password in a password file (see "Using a Password File" on page 158).

The `imqkeytool` command runs the JDK `keytool` utility to generate a self-signed certificate and places it in Message Queue's key store, located in a directory that depends upon the operating system, as shown in Appendix A, "Platform-Specific Locations of Message Queue Data."

The key store is in the same format as that supported by the JDK1.2 `keytool` utility.

These are the configurable properties for the Message Queue key store:

- `imq.keystore.file.dirpath`. For SSL-based services: specifies the path to the directory containing the key store file. For the default value, see Appendix A, "Platform-Specific Locations of Message Queue Data."

- `imq.keystore.file.name`. For SSL-based services: specifies the name of the key store file.
- `imq.keystore.password`. For SSL-based services: specifies the key store password.

You might need to regenerate a key pair in order to solve certain problems; for example:

- You forgot the key store password.
- The SSL-based service fails to initialize when you start a broker and you get the exception `java.security.UnrecoverableKeyException: Cannot recover key`.

  This exception may result from the fact that you had provided a key password that was different from the key store password when you generated the self-signed certificate in "Step 1. Generating a Self-Signed Certificate" on page 150.

➤ To Regenerate a Key Pair

1. Remove the broker's key store, located as shown in Appendix A, "Platform-Specific Locations of Message Queue Data."
2. Rerun `imqkeytool` to generate a key pair as described in "Step 1. Generating a Self-Signed Certificate" on page 150.

Step 2. Enabling the SSL-Based Service in the Broker

To enable the SSL-based service in the broker, you need to add `ssljms` (or `ssladmin`) to the `imq.service.activelist` property.

NOTE	The SSL-based `cluster` connection service is enabled using the `imq.cluster.transport` property rather than the `imq.service.activelist` property. See "Secure Connections Between Brokers" on page 184.

➤ To Enable an SSL-based Service in the Broker

1. Open the broker's instance configuration file.

   The instance configuration file is located in a directory identified by the name of the broker instance (*instanceName*) with which the configuration file is associated (see Appendix A, "Platform-Specific Locations of Message Queue Data"):

   …/`instances`/*instanceName*/`props/config.properties`

2. Add an entry (if one does not already exist) for the `imq.service.activelist` property and include SSL-based services in the list.

   By default, the property includes the jms and admin connection services. You need to add the ssljms or ssladmin connection services or both (depending on the services you want to activate):

   ```
   imq.service.activelist=jms,admin,ssljms,ssladmin
   ```

Step 3. Starting the Broker

Start the broker, providing the key store password. You can provide the password in any one of the following ways:

- Allow the broker to prompt you for the password when it starts up:

  ```
  imqbrokerd
  Please enter Keystore password: myPassword
  ```

- Put the password in a password file, as described in "Using a Password File" on page 158. Once you have put the password in the password file and set the property `imq.passfile.enabled=true`, do one of the following:

  - Pass the location of the password file to the `imqbrokerd` command:

    ```
    imqbrokerd -passfile /tmp/myPassfile
    ```

  - Start the broker without the `-passfile` option, but specify the location of the password file using the following two broker configuration properties:

    ```
    imq.passfile.dirpath=/tmp
    ```

    ```
    imq.passfile.name=myPassfile
    ```

When you start a broker or client with SSL, you might notice that it consumes a lot of cpu cycles for a few seconds. This is because Message Queue uses JSSE (Java Secure Socket Extension) to implement SSL. JSSE uses `java.security.SecureRandom` to generate random numbers. This method takes a significant amount of time to create the initial random number seed, and that is why you are seeing increased cpu usage. After the seed is created, the cpu level will drop to normal.

Step 4. Configuring and Running SSL-Based Clients

Finally, you configure clients to use the secure connection services. There are two types of secure connection scenarios over TCP/IP:

- Application clients using `ssljms`
- Message Queue administration clients (such as `imqcmd`) using `ssladmin`

These are treated separately in the following sections.

Application Clients Using ssljms

You must make sure the client has the necessary Secure Socket Extension (JSSE) `.jar` files in its classpath, and you need to tell it to use the `ssljms` connection service.

1. If your client is not using J2SDK1.4 (which has JSSE and JNDI support built in), make sure the client has the following `.jar` files in its class path:

   ```
   jsse.jar, jnet.jar, jcert.jar, jndi.jar
   ```

2. Make sure the client has the following Message Queue `.jar` files in its class path:

   ```
   imq.jar, jms.jar
   ```

3. Start the client and connect to the broker's ssljms service. One way to do this is by entering a command like the following:

   ```
   java -DimqConnectionType=TLS clientAppName
   ```

   Setting `imqConnectionType` tells the connection to use SSL.

   For more information on using `ssljms` connection services in client applications, see the chapter on using administered objects in the *Message Queue Developer's Guide for Java Clients*.

Administration Clients (imqcmd) Using ssladmin

You can establish a secure administration connection by including the `-secure` option when using `imqcmd`. For example:

```
imqcmd list svc -b hostName:portNumber -u adminName -secure
```

where *adminName* is a valid entry in the Message Queue user repository and the command will prompt for the password. (If you are using a flat-file repository, see "Changing the Default Administrator Password" on page 138).

Listing the connection services is a way to show that the `ssladmin` service is running, and that you can successfully make a secure admin connection, as shown in the following output:

```
Listing all the services on the broker specified by:

Host                 Primary Port
localhost            7676

Service Name     Port Number       Service State
admin            33984 (dynamic)   RUNNING
httpjms          -                 UNKNOWN
httpsjms         -                 UNKNOWN
jms              33983 (dynamic)   RUNNING
ssladmin         35988 (dynamic)   RUNNING
ssljms           dynamic           UNKNOWN

Successfully listed services.
```

Configuring the Use of Signed Certificates

Signed certificates provide a stronger level of server authentication than self-signed certificates. To implement signed certificates, you install a signed certificate into the key store, and then configure the Message Queue client so that it requires a signed certificate when it establishes an SSL connection to `imqbrokerd`.

You can implement signed certificates only between client and broker, and not between multiple brokers in a cluster.

The instructions that follow assume that you have already performed the steps documented under "Configuring the Use of Self-Signed Certificates" on page 149. While you are following the instructions, it might be helpful to have access to the information about J2SE keytool and X.509 certificates at `http://java.sun.com`.

Step 1: Obtaining and Installing a Signed Certificate

➤ To Obtain a Signed Certificate

1. Use the J2SE keytool to generate a Certificate Signing Request (CSR) for the self-signed certificate you just generated.

   Here is an example:

   ```
   keytool -certreq -keyalg RSA -alias imq -file certreq.csr
           -keystore /etc/imq/keystore -storepass myStorePassword
   ```

   The CSR now encapsulates the certificate in the file `certreq.csr`.

2. Generate or request a signed certificate by one of the following methods:

   - Have the certificate signed by a well known certificate authority (CA), such as Thawte or Verisign. See your CA's documentation for more information on this process.

   - Sign the certificate yourself by using an SSL signing software package.

   The resulting signed certificate is a sequence of ASCII characters. If you receive the signed certificate from a CA, it might arrive as an email attachment or in the text of a message.

3. When you get the signed certificate, save it in a file.

   These instructions use the example name `broker.cer` to represent the broker certificate.

➤ To Install a Signed Certificate

1. Check `$JAVA_HOME/lib/security/cacerts` to find out whether J2SE supports your CA by default, as follows:

   ```
   keytool -v -list -keystore $JAVA_HOME/lib/security/cacerts
   ```

   The command lists the root CAs in the system key store.

   If your CA is listed, skip the next step.

2. If your CA is not supported in J2SE, import the certificate authority's root certificate into the `imqbrokerd` key store.

   Here is an example:

   ```
   keytool -import -alias ca -file ca.cer -noprompt -trustcacerts
           -keystore /etc/imq/keystore -storepass myStorePassword
   ```

   The `ca.cer` value is the CA root certificate obtained from the CA.

   If you are using a CA test certificate, you probably need to import the Test CA Root certificate. Your CA should have instructions on how to obtain a copy of the Test CA Root.

3. Import the signed certificate into the key store to replace the original self-signed certificate.

   For example:

   ```
   keytool -import -alias imq -file broker.cer -noprompt -trustcacerts
           -keystore /etc/imq/keystore -storepass myStorePassword
   ```

   The `broker.cer` value is the file that contains the signed certificate that you received from the CA.

The `imqbrokerd` key store now has a signed certificate to use for SSL connections.

Step 2: Configuring the Client Runtime to Require a Signed Certificate

➤ To Configure the Java Client Runtime

By default, the Message Queue client runtime trusts `imqbrokerd` and accepts any certificate that is presented to it. You must now configure the client runtime to require signed certificates, and ensure that the client trusts the CA that signed the certificate.

1. To configure the client to require a valid, signed certificate from `imqbrokerd`, set the `imqSSLIsHostTrusted` attribute to `false` for the client's `ConnectionFactory` object.

2. Try to establish an SSL connection to `imqbrokrd`, as described under "Step 4. Configuring and Running SSL-Based Clients" on page 154.

   If the `broker's` certificate was signed by a well-known CA, the connection will probably succeed and you can skip the next step. If the connection fails with a certificate validation error, perform the next step.

3. Install the signing CA's root certificate in the client's trust store, as described in the following sections.

There are three options for configuring the client with a trust store:

- Install the root CA into the default system `cacerts` file.
- Install the root CA into the alternative system file `jssecacerts`. This is the recommended option.
- Install the root CA into any key store file and configure the client to use that as its trust store.

The following sections contain examples of how to install a Verisign Test Root CA using these options. The root CA is contained in a file called `testrootca.cer`. The examples assume that J2SE is installed in `/usr/j2se`.

Installing into the Default System cacerts File

This example installs the root CA into the file `$JAVA_HOME/usr/jre/lib/security/cacerts`.

```
keytool -import -keystore /usr/j2se/jre/lib/security/cacerts
         -alias VerisignTestCA -file testrootca.cer -noprompt
         -trustcacerts -storepass myStorePassword
```

The client searches this key store by default, so no further client configuration is necessary.

Installing into jssecacerts

This example installs the root CA into the file `$JAVA_HOME/usr/jre/lib/security/jssecacerts`.

```
keytool -import -keystore /usr/j2se/jre/lib/security/jssecacerts
         -alias VerisignTestCA -file testrootca.cer -noprompt
         -trustcacerts -storepass myStorePassword
```

The client searches this key store by default, so no further client configuration is necessary.

Installing into Other Files

This example installs the root CA into the file `/home/smith/.keystore`.

```
keytool -import -keystore /home/smith/.keystore
         -alias VerisignTestCA -file testrootca.cer -noprompt
         -trustcacerts -storepass myStorePassword
```

The client does not search this key store by default, so you must provide the location of the trust store to the client. To do so, set the Java system property `javax.net.ssl.trustStore` once the client is running. For example:

```
javax.net.ssl.trustStore=/home/smith/.keystore
```

Using a Password File

Several types of commands require passwords. In Table 7-6, the first column lists the commands that require passwords and the second column lists the reason that passwords are needed.

Table 7-6 Commands That Use Passwords

Command	Purpose	Purpose of Password
`imqbrokerd`	Start the broker	Access a JDBC-based persistent data store, an SSL certificate key store, or an LDAP user repository
`imqcmd`	Manage the broker	Authenticate an administrative user who is authorized to use the command
`imqdbmgr`	Manage a JDBC-based data store	Access the data store

You can specify these passwords in a *password file* and use the `-passfile` option to specify the name of the file. This is the format for the `-passfile` option:

```
imqbrokerd -passfile myPassfile
```

NOTE In previous releases, you could use the `-p`, `-password`, `-dbpassword`, and `-ldappassword` options to specify passwords on a command line. These options are deprecated and will be removed in a future release. In the current release, a value on the command line for one of these options supersedes the associated value in a password file.

Security Concerns

Specifying a password interactively, in response to a prompt, is the most secure method of specifying a password, unless your monitor is visible to other people. You can also specify a password file on the command line. For non-interactive use of commands, however, you must use a password file.

A password file is unencrypted, so you must set its permissions to protect it from unauthorized access. Set permissions such that they limit the users who can view the file, but provide read access to the user who starts the broker.

Password File Contents

A password file is a simple text file that contains a set of properties and values. Each value is a password used by a command.

A password file can contain the passwords shown in Table 7-7:

Table 7-7 Passwords in a Password File

Password	Affected Commands	Description
`imq.imqcmd.password`	`imqcmd`	Specifies the administrator password for an `imqcmd` command line. The password is authenticated for each command.
`imq.keystore.password`	`imqbrokerd`	Specifies the key store password for SSL-based services.
`imq.persist.jdbc.password`	`imqbrokerd` `imdbmgr`	Specifies the password used to open a database connection, if required.
`imq.user_repository.ldap.password`	`imqbrokerd`	Specifies the password associated with the distinguished name assigned to a broker for binding to a configured LDAP user repository.

A sample password file is part of the Message Queue product. For the location of the sample file, see Appendix A, “Platform-Specific Locations of Message Queue Data.”

Creating an Audit Log

Message Queue supports audit logging in Enterprise Edition only. When audit logging is enabled, Message Queue generates a record for the following types of events:

- Startup, shutdown, restart, and removal of a broker instance
- User authentication and authorization
- Reset of a persistent store
- Creation, purge, and destruction of a physical destination
- Administrative destruction of a durable subscriber

To log audit records to the Message Queue broker log file, set the `imq.audit.enabled` broker property to `true`. All audit records in the log contain the keyword `AUDIT`.

Chapter 8

Managing Administered Objects

Administered objects encapsulate provider-specific configuration and naming information, enabling the development of client applications that are portable from one JMS provider to another. A Message Queue administrator typically creates administered objects for client applications to use in obtaining broker connections for sending and receiving messages.

This chapter tells how to use the Object Manager utility (`imqobjmgr`) to create and manage administered objects. It contains the following sections:

Object Stores

Administered objects are placed in a readily available object store where they can be accessed by client applications via the Java Naming and Directory Interface (JNDI). There are two types of object store you can use: a standard Lightweight Directory Access Protocol (LDAP) directory server or a directory in the local file system.

LDAP Server Object Stores

An LDAP server is the recommended object store for production messaging systems. LDAP servers are designed for use in distributed systems and provide security features that are useful in production environments.

LDAP implementations are available from a number of vendors. To manage an object store on an LDAP server with Message Queue administration tools, you may first need to configure the server to store Java objects and perform JNDI lookups; see the documentation provided with your LDAP implementation for details.

To use an LDAP server as your object store, you must specify the attributes shown in Table 8-1. These attributes fall into the following categories:

- **Initial context.** The `java.naming.factory.initial` attribute specifies the initial context for JNDI lookups on the server. The value of this attribute is fixed for a given LDAP object store.
- **Location.** The `java.naming.provider.url` attribute specifies the URL and directory path for the LDAP server. You must verify that the specified directory path exists.
- **Security.** The attributes `java.naming.security.principal`, `java.naming.security.credentials`, and `java.naming.security.authentication` govern the authentication of callers attempting to access the object store. The exact format and values of these attributes depend on the LDAP service provider; see the documentation provided with your LDAP implementation for details and to determine whether security information is required on all operations or only on those that change the stored data.

Table 8-1 LDAP Object Store Attributes

Attribute	Description
`java.naming.factory.initial`	Initial context for JNDI lookup Example: `com.sun.jndi.ldap.LdapCtxFactory`
`java.naming.provider.url`	Server URL and directory path Example: `ldap://mydomain.com:389/ou=mqobjs,o=myapp` where administered objects are stored in the directory `/myapp/mqobjs`.

Table 8-1 LDAP Object Store Attributes *(Continued)*

Attribute	Description
`java.naming.security.principal`	Identity of the principal for authenticating callers The format of this attribute depends on the authentication scheme: for example, `uid=homerSimpson,ou=People,o=mq` If this attribute is unspecified, the behavior is determined by the LDAP service provider.
`java.naming.security.credentials`	Credentials of the authentication principal The value of this attribute depends on the authentication scheme: for example, it might be a hashed password, a clear-text password, a key, or a certificate. If this property is unspecified, the behavior is determined by the LDAP service provider.
`java.naming.security.authentication`	Security level for authentication The value of this attribute is one of the keywords `none`, `simple`, or `strong`. For example, If you specify `simple,` you will be prompted for any missing principal or credential values. This will allow you a more secure way of providing identifying information. If this property is unspecified, the behavior is determined by the LDAP service provider.

File-System Object Stores

Message Queue also supports the use of a directory in the local file system as an object store for administered objects. While this approach is not recommended for production systems, it has the advantage of being very easy to use in development environments. Note, however, that for a directory to be used as a centralized object store for clients deployed across multiple computer nodes, all of those clients must have access to the directory. In addition, any user with access to the directory can use Message Queue administration tools to create and manage administered objects.

To use a file-system directory as your object store, you must specify the attributes shown in Table 8-2. These attributes have the same general meanings described above for LDAP object stores; in particular, the `java.naming.provider.url` attribute specifies the directory path of the directory holding the object store. This directory must exist and have the proper access permissions for the user of Message Queue administration tools as well as the users of the client applications that will access the store.

Table 8-2 File-system Object Store Attributes

Attribute	Description
`java.naming.factory.initial`	Initial context for JNDI lookup Example: `com.sun.jndi.fscontext.RefFSContextFactory`
`java.naming.provider.url`	Directory path Example: `file:///C:/myapp/mqobjs`

Administered Object Attributes

Message Queue administered objects are of two basic kinds:

- *Connection factories* are used by client applications to create connections to brokers.
- *Destinations* represent locations on a broker with which client applications can exchange (send and retrieve) messages.

Each of these types of administered object has certain attributes that determine the object's properties and behavior. This section describes how to use the Object Manager command line utility (`imqobjmgr`) to set these attributes; you can also set them with the GUI Administration Console, as described in Chapter 2 (see "Working With Administered Objects" on page 55).

Connection Factory Attributes

Client applications use *connection factory* administered objects to create connections with which to exchange messages with a broker. A connection factory's attributes define the properties of all connections it creates. Once a connection has been created, its properties cannot be changed; thus the only way to configure a connection's properties is by setting the attributes of the connection factory used to create it.

Message Queue defines two classes of connection factory objects:

- `ConnectionFactory` objects support normal messaging and nondistributed transactions.
- `XAConnectionFactory` objects support distributed transactions.

Both classes share the same configuration attributes, which you can use to optimize resources, performance, and message throughput. These attributes are listed and described in detail in Chapter 16, "Administered Object Attribute Reference," and are discussed in the following sections below:

- "Connection Handling" on page 165
- "Client Identification" on page 168
- "Reliability And Flow Control" on page 170
- "Queue Browser and Server Sessions" on page 171
- "Standard Message Properties" on page 171
- "Message Header Overrides" on page 171

Connection Handling

Connection handling attributes specify the message server address to which to connect and, if required, how to detect connection failure and attempt reconnection. They are summarized in Table 16-1 on page 318.

Message Server Address List

The most important connection handling attribute is `imqAddressList`, which specifies the broker or brokers to which to establish a connection. The value of this attribute is a string containing a message server address or (in the case of a broker cluster) multiple addresses separated by commas. Server addresses can use a variety of addressing schemes, depending on the connection service to be used (see "Connection Services" on page 76) and the method of establishing a connection:

- mq uses the broker's Port Mapper to assign a port dynamically for either the jms or ssljms connection service.
- mqtcp bypasses the Port Mapper and connects directly to a specified port, using the jms connection service.
- mqssl makes a Secure Socket Layer (SSL) connection to a specified port, using the ssljms connection service.
- http makes a Hypertext Transport Protocol (HTTP) connection to a Message Queue tunnel servlet at a specified URL, using the httpjms connection service.
- https makes a Secure Hypertext Transport Protocol (HTTPS) connection to a Message Queue tunnel servlet at a specified URL, using the httpsjms connection service.

These addressing schemes are summarized in Table 16-2 on page 320.

The general format for each message server address is

scheme://*address*

where *scheme* is one of the addressing schemes listed above and *address* denotes the server address itself. The exact syntax for specifying the address varies depending on the addressing scheme, as shown in the last column of Table 16-2. Table 16-3 on page 321 shows examples of the various address formats.

In a multiple-broker cluster environment, the address list can contain more than one server address. If the first connection attempt fails, the Message Queue client runtime will attempt to connect to another address in the list, and so on until the list is exhausted. Two additional connection factory attributes control the way this is done:

- imqAddressListBehavior specifies the order in which to try the specified addresses. If this attribute is set to the string PRIORITY, addresses will be tried in the order in which they appear in the address list. If the attribute value is RANDOM, the addresses will instead be tried in random order; this is useful, for instance, when many Message Queue clients are sharing the same connection factory object, to prevent them from all attempting to connect to the same server address.
- imqAddressListIterations specifies how many times to cycle through the list before giving up and reporting failure. A value of -1 denotes an unlimited number of iterations: the client runtime will keep trying until it succeeds in establishing a connection or until the end of time, whichever occurs first.

Automatic Reconnection

By setting a connection factory's `imqReconnectEnabled` attribute to `true`, you can enable a client to reconnect automatically to a broker if a connection fails. The `imqReconnectAttempts` attribute controls the number of reconnection attempts to a given server address; `imqReconnectInterval` specifies the interval, in milliseconds, to wait between attempts.

In a broker cluster, where the message server address list (`imqAddressList`) specifies multiple addresses, a failed connection can be restored not only on the original broker, but also on a different one in the cluster. If reconnection to the original broker fails, the client runtime will try the other addresses in the list. The `imqAddressListBehavior` and `imqAddressListIterations` attributes control the order in which addresses are tried and the number of iterations through the list, as described in the preceding section. Each address is tried repeatedly at intervals of `imqReconnectInterval` milliseconds, up to the maximum number of attempts specified by `imqReconnectAttempts`.

Automatic reconnection supports all client acknowledgment modes for message consumption. Once a connection has been reestablished, the broker will redeliver all unacknowledged messages it had previously delivered, marking them with a `Redeliver` flag. Application code can use this flag to determine whether any message has already been consumed but not yet acknowledged. (In the case of nondurable subscribers, however, the message server does not hold messages once their connections have been closed. Thus any messages produced for such subscribers while the connection is down cannot be delivered after reconnection and will be lost.) Message production is blocked while automatic reconnection is in progress; message producers cannot send messages to the server until after the connection has been reestablished.

Automatic reconnection provides connection failover, but not data failover: persistent messages and other state information held by a failed or disconnected broker can be lost when the client is reconnected to a different broker instance. While attempting to reestablish a connection, Message Queue does maintain objects (such as sessions, message consumers, and message producers) provided by the client runtime. Temporary destinations are also maintained for a time when a connection fails, because clients might reconnect and access them again; after giving clients time to reconnect and use these destinations, the broker will delete them. In circumstances where the client-side state cannot be fully restored on the broker on reconnection (for example, when using transacted sessions, which exist only for the duration of a connection), automatic reconnection will not take place and the connection's exception handler will be called instead. It is then up to the application code to catch the exception, reconnect, and restore state.

Periodic Testing (Pinging) of Connections

The Message Queue client runtime can be configured to periodically test, or "ping," a connection, allowing connection failures to be detected preemptively before an attempted message transmission fails. Such testing is particularly important for client applications that only consume messages and do not produce them, since such applications cannot otherwise detect when a connection has failed. Clients that produce messages only infrequently can also benefit from this feature.

The connection factory attribute `imqPingInterval` specifies the frequency, in seconds, with which to ping a connection. By default, this interval is set to 30 seconds; a value of `-1` disables the ping operation.

The response to an unsuccessful ping varies from one operating-system platform to another. On some operating systems, an exception is immediately thrown to the client application's exception listener. (If the client does not have an exception listener, its next attempt to use the connection will fail.) Other systems may continue trying to establish a connection to the broker, buffering successive pings until one succeeds or the buffer overflows.

Client Identification

The connection factory attributes listed in Table 16-4 on page 322 support client authentication and the setting of client identifiers for durable subscribers.

Client Authentication

All attempts to connect to a broker must be authenticated by user name and password against a user repository maintained by the message server. The connection factory attributes `imqDefaultUsername` and `imqDefaultPassword` specify a default user name and password to be used if the client does not supply them explicitly when creating a connection.

As a convenience for developers who do not wish to bother populating a user repository during application development and testing, Message Queue provides a guest user account with user name and password both equal to `guest`. This is also the default value for the `imqDefaultUsername` and `imqDefaultPassword` attributes, so that if they are not specified explicitly, clients can always obtain a connection under the guest account. In a production environment, access to broker connections should be restricted to users who are explicitly registered in the user repository.

Client Identifier

The *Java Message Service Specification* requires that a connection provide a unique *client identifier* whenever the message server must maintain a persistent state on behalf of a client. Message Queue uses such client identifiers to keep track of durable subscribers to a topic destination. When a durable subscriber becomes inactive, the broker retains all incoming messages for the topic and delivers them when the subscriber becomes active again. The broker identifies the subscriber by means of its client identifier.

While it is possible for a client application to set its own client identifier programmatically using the connection object's `setClientID` method, this makes it difficult to coordinate client identifiers to ensure that each is unique. It is generally better to have Message Queue automatically assign a unique identifier when creating a connection on behalf of a client. This can be done by setting the connection factory's `imqConfiguredClientID` attribute to a value of the form

`${u}`*factoryID*

The characters `${u}` must be the first four characters of the attribute value. (Any character other than `u` between the braces will cause an exception to be thrown on connection creation; in any other position, these characters have no special meaning and will be treated as plain text.) The value for *factoryID* is a character string uniquely associated with this connection factory object.

When creating a connection for a particular client, Message Queue will construct a client identifier by replacing the characters `${u}` with `u:`*userName*, where *userName* is the user name authenticated for the connection. This ensures that connections created by a given connection factory, although identical in all other respects, will each have their own unique client identifier. For example, if the user name is `Calvin` and the string specified for the connection factory's `imqConfiguredClientID` attribute is `${u}Hobbes`, the client identifier assigned will be `u:CalvinHobbes`.

NOTE This scheme will not work if two clients both attempt to obtain connections using the default user name `guest`, since each would have a client identifier with the same `${u}` component. In this case, only the first client to request a connection will get one; the second client's connection attempt will fail, because Message Queue cannot create two connections with the same client identifier.

Even if you specify a client identifier with `imqConfiguredClientID`, client applications can override this setting with the connection method `setClientID`. You can prevent this by setting the connection factory's `imqDisableSetClientID` attribute to `true`. Note that for an application that uses durable subscribers, the client identifier *must* be set one way or the other: either administratively with `imqConfiguredClientID` or programmatically with `setClientID`.

Reliability And Flow Control

Because "payload" messages sent and received by clients and control messages (such as broker acknowledgments) used by Message Queue itself pass over the same client-broker connection, excessive levels of payolad traffic can interfere with the delivery of control messages. To help alleviate this problem, the connection factory attributes listed in Table 16-5 on page 322 allow you to manage the relative flow of the two types of message. These attributes fall into four categories:

- **Acknowledgment timeout** specifies the maximum time (`imqAckTimeout`) to wait for a broker acknowledgment before throwing an exception.
- **Connection flow metering** limits the transmission of payload messages to batches of a specified size (`imqConnectionFlowCount`), ensuring periodic opportunities to deliver any accumulated control messages.
- **Connection flow control** limits the number of payload messages (`imqConnectionFlowLimit`) that can be held pending on a connection, waiting to be consumed. When the limit is reached, delivery of payload messages to the connection is suspended until the number of messages awaiting consumption falls below the limit. Use of this feature is controlled by a boolean flag (`imqConnectionFlowLimitEnabled`).
- **Consumer flow control** limits the number of payload messages (`imqConsumerFlowLimit`) that can be held pending for any single consumer, waiting to be consumed. (This limit can also be specified as a property of a specific queue destination, `consumerFlowLimit`.) When the limit is reached, delivery of payload messages to the consumer is suspended until the number of messages awaiting consumption, as a percentage of `imqConsumerFlowLimit`, falls below the limit specified by the `imqConsumerFlowThreshold` attribute. This helps improve load balancing among multiple consumers by preventing any one consumer from starving others on the same connection.

The use of any of these flow control techniques involves a tradeoff between reliability and throughput; see "Client Runtime Message Flow Adjustments" on page 229 for further discussion.

Queue Browser and Server Sessions

Table 16-6 on page 324 lists connection factory attributes affecting client queue browsing and server sessions. The `imqQueueBrowserMaxMessagesPerRetrieve` attribute specifies the maximum number of messages to retrieve at one time when browsing the contents of a queue destination; `imqQueueBrowserRetrieveTimeout` gives the maximum waiting time for retrieving them. The boolean attribute `imqLoadMaxToServerSession` governs the behavior of connection consumers in an application server session: if the value of this attribute is `true`, the client will load up to the maximum number of messages into a server session; if `false`, it will load only a single message at a time.

Standard Message Properties

The *Java Message Service Specification* defines certain standard message properties, which JMS providers (such as Message Queue) may optionally choose to support. By convention, the names of all such standard properties begin with the letters `JMSX`. The connection factory attributes listed in Table 16-7 on page 324 control whether the Message Queue client runtime sets certain of these standard properties. For produced messages, these include the following properties:

`JMSXUserID`	Identity of the user sending the message
`JMSXAppID`	Identity of the application sending the message
`JMSXProducerTXID`	Transaction identifier of the transaction within which the message was produced

For consumed messages, they include

`JMSXConsumerTXID`	Transaction identifier of the transaction within which the message was consumed
`JMSXRcvTimestamp`	Time the message was delivered to the consumer

Message Header Overrides

You can use the connection factory attributes listed in Table 16-8 on page 325 to override the values set by a client for certain JMS message header fields. The settings you specify will be used for all messages produced by connections obtained from that connection factory. Header fields that you can override in this way are

• `JMSDeliveryMode`	Delivery mode (persistent or nonpersistent
• `JMSExpiration`	Expiration time
• `JMSPriority`	Priority level

There are two attributes for each of these fields: one boolean, to control whether the field can be overridden, and another to specify its value. For instance, the attributes for setting the priority level are `imqOverrideJMSPriority` and `imqJMSPriority`. There is also an additional attribute, `imqOverrideJMSHeadersToTemporaryDestinations`, that controls whether override values apply to temporary destinations.

NOTE Because overriding message headers may interfere with the needs of specific applications, these attributes should only be used in consultation with an application's designers or users.

Destination Attributes

The *destination* administered object that identifies a physical queue or topic destination has only two attributes, listed in Table 16-9 on page 326. The important one is `imqDestinationName`, which gives the name of the physical destination that this administered object represents; this is the name that was specified with the `-n` option to the `imqcmd create dst` command that created the physical destination. There is also an optional descriptive string, `imqDestinationDescription`, which you can use to help identify the destination object and distinguish it from others you may have created.

Using the Object Manager Utility

The Message Queue Object Manager utility (`imqobjmgr`) allows you to create and manage administered objects. The `imqobjmgr` command provides the following subcommands for performing various operations on administered objects:

`add`	Add an administered object to an object store
`delete`	Delete an administered object from an object store
`list`	List existing administered objects in an object store
`query`	Display information about an administered object
`update`	Modify the attributes of an administered object

See "Object Manager Utility" on page 279 for reference information about the syntax, subcommands, and options of the `imqobjmgr` command.

Most Object Manager operations require you to specify the following information as options to the `imqobjmgr` command:

- The **JNDI lookup name** of the administered object (`-l` option)

  This is the logical name by which client applications can look up the administered object in the object store, using the Java Naming and Directory Interface.

- The **attributes of the JNDI object store** (`-j` option)

  See "Object Stores" on page 161 for information on the possible attributes and their values.

- The **type of the administered object** (`-t` option)

  Possible types include the following:

`q`	Queue destination
`t`	Topic destination
`cf`	Connection factory
`qf`	Queue connection factory
`tf`	Topic connection factory
`xcf`	Connection factory for distributed transactions
`xqf`	Queue connection factory for distributed transactions
`xtf`	Topic connection factory for distributed transactions
`e`	SOAP endpoint

- The **attributes of the administered object** (`-o` option)

  See "Administered Object Attributes" on page 164 for information on the possible attributes and their values.

Adding Administered Objects

The `imqobjmgr` command's `add` subcommand adds administered objects for connection factories and topic or queue destinations to the object store. Administered objects stored in an LDAP object store must have lookup names beginning with the prefix `cn=`; lookup names in a file-system object store need not begin with any particular prefix, but must not include the slash character (`/`).

NOTE The Object Manager lists and displays only Message Queue administered objects. If an object store should contain a non–Message Queue object with the same lookup name as an administered object that you wish to add, you will receive an error when you attempt the add operation.

Adding a Connection Factory

To enable client applications to create broker connections, add a connection factory administered object for the type of connection to be created: a queue connection factory or a topic connection factory. Code Example 8-1shows a command to add a queue connection factory (administered object type `qf`) to an LDAP object store. The object has lookup name `cn=myQCF` and connects to a broker running on host `myHost` at port number `7272`, using the `jms` connection service.

Code Example 8-1 Adding a Connection Factory

```
imqobjmgr add
   -l "cn=myQCF"
   -j "java.naming.factory.initial=com.sun.jndi.ldap.LdapCtxFactory"
   -j "java.naming.provider.url=ldap://mydomain.com:389/o=imq"
   -j "java.naming.security.principal=uid=homerSimpson,ou=People,o=imq"
   -j "java.naming.security.credentials=doh"
   -j "java.naming.security.authentication=simple"
   -t qf
   -o "imqAddressList=mq://myHost:7272/jms"
```

Adding a Destination

When creating an administered object representing a destination, it is good practice to create the physical destination first, before adding the administered object to the object store. Use the Command utility (`imqcmd`) to create the physical destination, as described in "Creating a Physical Destination" on page 119.

The command shown in Code Example 8-2 adds an administered object to an LDAP object store representing a topic destination with lookup name `myTopic` and physical destination name is `physTopic`. The command for adding a queue destination would be similar, except that the administered object type (`-t` option) would be `q` (for "queue destination") instead of `t` (for "topic destination").

Code Example 8-2 Adding a Destination to an LDAP Object Store

```
imqobjmgr add
   -l "cn=myTopic"
   -j "java.naming.factory.initial=com.sun.jndi.ldap.LdapCtxFactory"
   -j "java.naming.provider.url=ldap://mydomain.com:389/o=imq"
   -j "java.naming.security.principal=uid=homerSimpson,ou=People,o=imq"
   -j "java.naming.security.credentials=doh"
   -j "java.naming.security.authentication=simple"
   -t t
   -o "imqDestinationName=physTopic"
```

Code Example 8-3 shows the same command, but with the administered object stored in a Solaris file system instead of an LDAP server.

Code Example 8-3 Adding a Destination to a File-System Object Store

```
imqobjmgr add
   -l "cn=myTopic"
   -j "java.naming.factory.initial=
           com.sun.jndi.fscontext.RefFSContextFactory"
   -j "java.naming.provider.url=file:///home/foo/imq_admin_objects"
   -t t
   -o "imqDestinationName=physTopic"
```

Deleting Administered Objects

To delete an administered object from the object store, you use the `delete` subcommand of the `imqobjmgr` command, specify lookup name, type, and location of the object to be deleted. The command shown in Code Example 8-4 deletes the object that was added in Code Example 8-2 above.

Code Example 8-4 Deleting an Administered Object

```
imqobjmgr delete
   -l "cn=myTopic"
   -j "java.naming.factory.initial=com.sun.jndi.ldap.LdapCtxFactory"
   -j "java.naming.provider.url=ldap://mydomain.com:389/o=imq"
   -j "java.naming.security.principal=uid=homerSimpson,ou=People,o=imq"
   -j "java.naming.security.credentials=doh"
   -j "java.naming.security.authentication=simple"
   -t t
```

Listing Administered Objects

You can use the Object Manager's `list` subcommand to get a list of all administered objects in an object store or those of a specific type. Code Example 8-5 shows how to list all administered objects on an LDAP server.

Code Example 8-5 Listing All Administered Objects

```
imqobjmgr list
   -j "java.naming.factory.initial=com.sun.jndi.ldap.LdapCtxFactory"
   -j "java.naming.provider.url=ldap://mydomain.com:389/o=imq"
   -j "java.naming.security.principal=uid=homerSimpson,ou=People,o=imq"
   -j "java.naming.security.credentials=doh"
   -j "java.naming.security.authentication=simple"
```

Code Example 8-6 lists all queue destinations (type `q`).

Code Example 8-6 Listing Administered Objects of a Specific Type

```
imqobjmgr list
   -j "java.naming.factory.initial=com.sun.jndi.ldap.LdapCtxFactory"
   -j "java.naming.provider.url=ldap://mydomain.com:389/o=imq"
   -j "java.naming.security.principal=uid=homerSimpson,ou=People,o=imq"
   -j "java.naming.security.credentials=doh"
   -j "java.naming.security.authentication=simple"
   -t q
```

Viewing Administered Object Information

The `query` subcommand displays information about a specified administered object, identified by its lookup name and the attributes of the object store containing it. Code Example 8-7 displays information about an object whose lookup name is cn=`myTopic`.

Code Example 8-7 Viewing Administered Object Information

```
imqobjmgr query
   -l "cn=myTopic"
   -j "java.naming.factory.initial=com.sun.jndi.ldap.LdapCtxFactory"
   -j "java.naming.provider.url=ldap://mydomain.com:389/o=imq"
   -j "java.naming.security.principal=uid=homerSimpson,ou=People,o=imq"
   -j "java.naming.security.credentials=doh"
   -j "java.naming.security.authentication=simple"
```

Modifying Administered Object Attributes

To modify the attributes of an administered object, use the `imqobjmgr update` subcommand. You supply the object's lookup name and location, and use the `-o` option to specify the new attribute values.

Code Example 8-8 changes the value of the `imqReconnectAttempts` attribute for the queue connection factory that was added to the object store in Code Example 8-1 on page 174.

Code Example 8-8 Modifying an Administered Object's Attributes

```
imqobjmgr update
   -l "cn=myQCF"
   -j "java.naming.factory.initial=com.sun.jndi.ldap.LdapCtxFactory"
   -j "java.naming.provider.url=ldap://mydomain.com:389/o=imq"
   -j "java.naming.security.principal=uid=homerSimpson,ou=People,o=imq"
   -j "java.naming.security.credentials=doh"
   -j "java.naming.security.authentication=simple"
   -t qf
   -o "imqReconnectAttempts=3"
```

Using Command Files

The -i option to the imqobjmgr command allows you to specify the name of a command file that uses Java property file syntax to represent all or part of the subcommand clause. This feature is especially useful for specifying object store attributes, which typically require a lot of typing and are likely to be the same across multiple invocations of imqobjmgr. Using a command file can also allow you to avoid exceeding the maximum number of characters allowed for the command line.

Code Example 8-9 shows the general syntax for an Object Manager command file. Note that the version property is not a command line option: it refers to the version of the command file itself (not that of the Message Queue product) and must be set to the value 2.0.

Code Example 8-9 Object Manager Command File Syntax

```
version=2.0
cmdtype=[ add | delete | list | query | update ]
obj.lookupName=lookup name
objstore.attrs.objStoreAttrName1=value1
objstore.attrs.objStoreAttrName2=value2
   . . .
objstore.attrs.objStoreAttrNameN=valueN
obj.type=[ q | t | cf | qf | tf | xcf | xqf | xtf | e ]
obj.attrs.objAttrName1=value1
obj.attrs.objAttrName2=value2
   . . .
obj.attrs.objAttrNameN=valueN
```

As an example, consider the Object Manager command shown earlier in Code Example 8-1 on page 174, which adds a queue connection factory to an LDAP object store. This command can be encapsulated in a command file as shown in Code Example 8-10. If the command file is named MyCmdFile, you can then execute the command with the command line

```
imqobjmgr -i MyCmdFile
```

Code Example 8-10 Example Command File

```
version=2.0
cmdtype=add
obj.lookupName=cn=myQCF
objstore.attrs.java.naming.factory.initial=com.sun.jndi.ldap.LdapCtxFactory
objstore.attrs.java.naming.provider.url=ldap://mydomain.com:389/o=imq
objstore.attrs.java.naming.security.principal=\
                                       uid=homerSimpson,ou=People,o=imq
objstore.attrs.java.naming.security.credentials=doh
objstore.attrs.java.naming.security.authentication=simple
obj.type=qf
obj.attrs.imqAddressList=mq://myHost:7272/jms
```

A command file can also be used to specify only part of the `imqobjmgr` subcommand clause, with the remainder supplied explicitly on the command line. For example, the command file shown in Code Example 8-12 specifies only the attribute values for an LDAP object store.

Code Example 8-11 Partial Command File

```
version=2.0
objstore.attrs.java.naming.factory.initial=com.sun.jndi.ldap.LdapCtxFactory
objstore.attrs.java.naming.provider.url=ldap://mydomain.com:389/o=imq
objstore.attrs.java.naming.security.principal=\
                                       uid=homerSimpson,ou=People,o=imq
objstore.attrs.java.naming.security.credentials=doh
objstore.attrs.java.naming.security.authentication=simple
```

You could then use this command file to specify the object store in an `imqobjmgr` command while supplying the remaining options explicitly, as shown in Code Example 8-12.

Code Example 8-12 Using a Partial Command File

```
imqobjmgr add
   -l "cn=myQCF"
   -i MyCmdFile
   -t qf
   -o "imqAddressList=mq://myHost:7272/jms"
```

Additional examples of command files can be found at the following locations, depending on your platform:

Solaris: `/usr/demo/imq/imqobjmgr`
Linux: `/opt/sun/mq/examples/imqobjmgr`
Windows: `IMQ_HOME/demo/imqobjmgr`

Chapter 9

Working With Broker Clusters

Message Queue Enterprise Edition supports the use of *broker clusters:* groups of brokers working together to provide message delivery services to clients. Clusters enable a message server to scale its operations with the volume of message traffic by distributing client connections among multiple brokers. See the *Message Queue Technical Overview* for a general discussion of clusters and how they operate.

This chapter describes how to manage broker clusters, connect brokers to them, and configure them. It contains the following sections:

- "Cluster Configuration Properties" on page 181
- "Managing Clusters" on page 183
- "Master Broker" on page 186

Cluster Configuration Properties

You define a cluster by specifying *cluster configuration properties* for each of its member brokers. You can set these properties individually for each broker in the cluster, but it is generally more convenient to collect them into a central *cluster configuration file* that all of the brokers reference. This prevents the settings from getting out of agreement and ensures that all brokers in a cluster share the same, consistent configuration information.

The cluster configuration properties are described in detail in Table 14-9 on page 307. They include the following:

- `imq.cluster.brokerlist` gives the host names and port numbers for all brokers belonging to the cluster.
- `imq.cluster.masterbroker` designates which broker (if any) is the master broker that keeps track of state changes.

- `imq.cluster.url` specifies the location of the cluster configuration file, if any.
- `imq.cluster.hostname` gives the host name or IP address for the `cluster` connection service, used for internal communication between brokers in the cluster. This setting can be useful if more than one host is available: for example, if there is more than one network interface card in a computer.
- `imq.cluster.port` gives the port number for the `cluster` connection service.
- `imq.cluster.transport` specifies the transport protocol used by the `cluster` connection service, such as `tcp` or `ssl`.

The `hostname` and `port` properties can be set independently for each individual broker, but `brokerlist`, `masterbroker`, `url`, and `transport` must have the same values for all brokers in the cluster.

The following sections describe how to set a broker's cluster configuration properties, either individually for each broker in a cluster or centrally, using a cluster configuration file.

Setting Cluster Properties for Individual Brokers

You can set a broker's cluster configuration properties in its instance configuration file (or on the command line when you start the broker). For example, to create a cluster consisting of brokers at port `9876` on `host1`, port `5000` on `host2`, and the default port (`7676`) on `ctrlhost`, you would include the following property in the instance configuration files for all three brokers:

```
imq.cluster.brokerlist=host1:9876,host2:5000,ctrlhost
```

Notice that if you need to change the cluster configuration, this method requires you to update the instance configuration file for every broker in the cluster.

Using a Cluster Configuration File

For consistency and ease of maintenance, it is recommended that you collect all of the shared cluster configuration properties into a single cluster configuration file instead of setting them separately for each individual broker. In this method, each broker's instance configuration file must set the `imq.cluster.url` property to point to the location of the cluster configuration file: for example,

```
imq.cluster.url=file:/home/cluster.properties
```

The cluster donfiguration file then defines the shared configuration properties for all of the brokers in the cluster, such as the list of brokers to be connected (imq.cluster.brokerlist), the transport protocol to use for the cluster connection service (imq.cluster.transport), and optionally, the address of the master broker (imq.cluster.masterbroker). The following code defines the same cluster as in the previous example, with the broker running on ctrlhost serving as the master broker:

```
imq.cluster.brokerlist=host1:9876,host2:5000,ctrlhost
imq.cluster.masterbroker=ctrlhost
```

Managing Clusters

This section describes how to connect a set of brokers to form a cluster, add new brokers to an existing cluster, and remove brokers from a cluster.

Connecting Brokers

There are two general methods of connecting brokers into a cluster: from the command line (using the -cluster option) or by setting the imq.cluster.brokerlist property in the cluster configuration file. Whichever method you use, each broker that you start attempts to connect to the other brokers every five seconds; the connection will succeed once the master broker is started up (if one is configured). If a broker in the cluster starts before the master broker, it will remain in a suspended state, rejecting client connections, until the master broker starts; the suspended broker then will automatically become fully functional.

To configure a broker cluster from the command line, use the -cluster option to the imqbrokerd command to specify the complete list of brokers in the cluster when you start each one. For example, the following command starts a new broker and connects it to the brokers running at the default port (7676) on host1, port 5000 on host2, and port 9876 on the default host (localhost):

```
imqbrokerd -cluster host1,host2:5000,:9876
```

An alternative method, better suited for production systems, is to create a cluster configuration file that uses the imq.cluster.brokerlist property to specify the list of brokers to be connected. Each broker in the cluster must then set its own imq.cluster.url property to point to this cluster configuration file.

Linux Prerequisite: Setting the IP Address

There is a special prerequisite for connecting brokers into a cluster on Linux systems. Some Linux installers automatically set the `localhost` entry to the network loopback IP address (`127.0.0.1`). You must set the system's IP address so that all brokers in the cluster can be addressed properly.

For all Linux systems that participate in a cluster, check the `/etc/hosts` file as part of cluster setup. If the system uses a static IP address, edit the `/etc/hosts` file to specify the correct address for `localhost`. If the address is registered with Domain Name Service (DNS), edit the file `/etc/nsswitch.conf` to change the order of the entries so that the system performs DNS lookup before consulting the local `hosts` file. The line in the `/etc/nsswitch.conf` file should read as follows:

```
hosts: dns files
```

Secure Connections Between Brokers

If you want secure, encrypted message delivery between brokers in a cluster, configure the `cluster` connection service to use an SSL-based transport protocol. For each broker in the cluster, set up SSL-based connection services, as described in "Working With an SSL-Based Service" on page 148. Then set each broker's `imq.cluster.transport` property to `ssl`, either in the cluster configuration file or individually for each broker.

Adding Brokers to a Cluster

The procedure for adding a new broker to a cluster depends on whether the cluster uses a cluster configuration file.

➤ **To Add a New Broker to a Cluster Using a Cluster Configuration File**

1. Add the new broker to the `imq.cluster.brokerlist` property in the cluster configuration file.

2. Issue the following command to every broker in the cluster:

   ```
   imqcmd reload cls
   ```

   This forces each broker to reload the cluster configuration, ensuring that all persistent information for brokers in the cluster is up to date.

3. *(Optional)* Set the value of the `imq.cluster.url` property in the broker's `config.properties` file to point to the cluster configuration file.

4. Start the new broker.

   If you did not perform step 3, use the `-D` option on the `imqbrokerd` command line to set the value of `imq.cluster.url`.

➤ To Add a New Broker to a Cluster Without a Cluster Configuration File

Set the value of the following properties, either by editing the `config.properties` file or by using the `-D` option on the `imqbrokerd` command line:

- `imq.cluster.brokerlist`
- `imq.cluster.masterbroker` (if necessary)
- `imq.cluster.transport` (if you are using a secure `cluster` connection service)

Removing Brokers From a Cluster

The method you use to remove a broker from a cluster depends on whether you originally created the cluster via the command line or by means of a central cluster configuration file.

Removing a Broker Using the Command Line

If you used the `imqbrokerd` command from the command line to connect the brokers into a cluster, you must stop each of the brokers and then restart them, specifying the new set of cluster members on the command line. The procedure is as follows:

➤ To Remove a Broker From a Cluster Using the Command Line

1. Stop each broker in the cluster, using the `imqcmd` command.
2. Restart the brokers that will remain in the cluster, using the `imqbrokerd` command's `-cluster` option to specify only those remaining brokers.

   For example, suppose you originally created a cluster consisting of brokers *A*, *B*, and *C* by starting each of the three with the command

   ```
   imqbrokerd -cluster A,B,C
   ```

   To remove broker *A* from the cluster, restart brokers *B* and *C* with the command

   ```
   imqbrokerd -cluster B,C
   ```

Removing a Broker Using a Cluster Configuration File

If you originally created a cluster by specifying its member brokers with the `imq.cluster.brokerlist` property in a central cluster configuration file, it isn't necessary to stop the brokers in order to remove one of them. Instead, you can simply edit the configuration file to exclude the broker you want to remove, force the remaining cluster members to reload the cluster configuration, and reconfigure the excluded broker so that it no longer points to the same cluster configuration file. Here is the procedure:

➤ **To Remove a Broker From a Cluster Using a Cluster Configuration File**

1. Edit the cluster configuration file to remove the excluded broker from the list specified for the `imq.cluster.brokerlist` property.

2. Issue the following command to each broker remaining in the cluster:

   ```
   imqcmd reload cls
   ```

   This forces the broker to reload the cluster configuration.

3. Stop the broker you're removing from the cluster.

4. Edit that broker's `config.properties` file, removing or specifying a different value for its `imq.cluster.url` property.

Master Broker

A cluster can optionally have one *master broker*, which maintains a *configuration change record* to keep track of any changes in the cluster's persistent state. The master broker is identified by the `imq.cluster.masterbroker` configuration property, either in the cluster configuration file or in the instance configuration files of the individual brokers.

The configuration change record contains information about changes in the persistent entities associated with the cluster, such as durable subscriptions and administrator-created physical destinations. All brokers in the cluster consult the master broker during startup in order to update their information about these persistent entities. Failure of the master broker makes such synchronization impossible; see "When a Master Broker Is Unavailable" on page 188 for more information.

Managing the Configuration Change Record

Because of the important information that the configuration change record contains, it is important to back it up regularly so that it can be restored in case of failure. Although restoring from a backup will lose any changes in the cluster's persistent state that have occurred since the backup was made, frequent backups can minimize this potential loss of information. The backup and restore operations also have the positive effect of compressing and optimizing the change history contained in the configuration change record, which can grow significantly over time.

➤ To Back Up the Configuration Change Record

Use the `-backup` option of the `imqbrokerd` command, specifying the name of the backup file. For example:

```
imqbrokerd -backup mybackuplog
```

➤ To Restore the Configuration Change Record

1. Shut down all brokers in the cluster.
2. Restore the master broker's configuration change record from the backup file with the command

   ```
   imqbrokerd -restore mybackuplog
   ```

3. If you assign a new name or port number to the master broker, update the `imq.cluster.brokerlist` and `imq.cluster.masterbroker` properties accordingly in the cluster configuration file.
4. Restart all brokers in the cluster.

When a Master Broker Is Unavailable

Because all brokers in a cluster need the master broker in order to perform persistent operations, the following `imqcmd` subcommands for any broker in the cluster will return an error when no master broker is available:

- `create dst`
- `destroy dst`
- `update dst`
- `destroy dur`

Auto-created physical destinations and temporary destinations are unaffected.

In the absence of a master broker, any client application attempting to create a durable subscriber or unsubscribe from a durable subscription will get an error. However, a client can successfully specify and interact with an existing durable subscription.

Chapter 10

Monitoring a Message Server

This chapter describes the tools you can use to monitor a message server and how you can get metrics data. The chapter has the following sections:

- "Introduction to Monitoring Tools" on page 189
- "Configuring and Using Broker Logging" on page 191
- "Interactively Displaying Metrics" on page 196
- "Writing an Application to Monitor Brokers" on page 201

Reference information on specific metrics is available in Chapter 18, "Metrics Reference."

Introduction to Monitoring Tools

There are three monitoring interfaces for Message Queue information: log files, interactive commands, and a client API that can obtain metrics. Each has its advantages and disadvantages, as follows:

- Log files provide a long-term record of metrics data, but cannot easily be parsed.
- Commands enable you to quickly sample information tailored to your needs, but do not enable you to look at historical information or manipulate the data programmatically.
- The client API lets you extract information, process it, manipulate the data, present graphs or send alerts. However, to use it, you must write a custom application to capture and analyze the data.

Table 10-1 compares the different tools.

Table 10-1 Benefits and Limitations of Metrics Monitoring Tools

Metrics Monitoring Tool	Benefits	LImitations
`imqcmd metrics`	Remote monitoring Convenient for spot checking Reporting interval set in command option; can be changed on the fly Easy to select specific data of interest Data presented in easy tabular format	No single command gets all data Difficult to analyze data programmatically Doesn't create historical record Difficult to see historical trends
Log files	Regular sampling Creates a historical record	Need to configure broker properties; must shut down and restart broker to take effect Local monitoring only Data format very difficult to read or parse; no parsing tools Reporting interval cannot be changed on the fly; the same for all metrics data Does not provide flexibility in selection of data Broker metrics only; destination and connection service metrics not included Possible performance hit if interval set too short
Client API	Remote monitoring Easy to select specific data of interest Data can be analyzed programmatically and presented in any format	Need to configure broker properties; must shut down and restart broker to take effect You need to write your own metrics monitoring client Reporting interval cannot be changed on the fly; the same for all metrics data

In addition to the differences shown in the table, each tool gathers a somewhat different subset of the metrics information generated by the broker. For information on which metrics data is gathered by each monitoring tool, see Chapter 18, "Metrics Reference" on page 335.

Configuring and Using Broker Logging

The Message Queue logger takes information generated by broker code, a debugger, and a metrics generator and writes that information to a number of output channels: to standard output (the console), to a log file, and, on Solaris™ operating systems, to the `syslog` daemon process.

You can specify the type of information gathered by the logger as well as the type written to each of the output channels. In particular, you can specify that you want metrics information written out to a log file.

This section describes the default logging configuration for the broker and explains how to redirect log information to alternative output channels, how to change log file rollover criteria, and how to send metrics data to a log file.

Default Logging Configuration

A broker is automatically configured to save log output to a set of rolling log files. The log files are located in a directory identified by the instance name of the associated broker (see Appendix A, "Platform-Specific Locations of Message Queue Data"):

```
.../instances/instanceName/log/
```

The log files are simple text files. They are named as follows, from earliest to latest:

```
log.txt
log_1.txt
log_2.txt
...
log_9.txt
```

By default, log files are rolled over once a week; the system maintains nine backup files.

- To change the directory in which the log files are kept, set the property `imq.log.file.dirpath` to the desired path.
- To change the root name of the log files from `log` to something else, set the `imq.log.file.filename` property.

The broker supports three log levels: `ERROR`, `WARNING`, `INFO`. Table 10-2 explains each level.

Table 10-2 Logging Levels

Level	Description
`ERROR`	Messages indicating problems that could cause system failure.
`WARNING`	Alerts that should be heeded but will not cause system failure.
`INFO`	Reporting of metrics and other informational messages.

Setting a logging level gathers messages for that level and all higher levels. The default log level is `INFO`, so `ERROR`, `WARNING`, and `INFO` messages are all logged by default.

Log Message Format

A logged message consists of a time stamp, message code, and the message itself. The volume of information varies with the log level you have set. The following is an example of an `INFO` message.

```
[13/Sep/2000:16:13:36 PDT] B1004 Starting the broker service using tcp [
25374,100] with min threads 50 and max threads of 500
```

To change the time stamp time zone, see information about the `imq.log.timezone` property, which is described in Table 14-8 on page 303.

Changing the Logger Configuration

Log-related properties are described in Table 14-8 on page 303.

➤ To Change the Logger Configuration for a Broker

1. Set the log level.
2. Set the output channel (file, console, or both) for one or more logging categories.
3. If you log output to a file, configure the rollover criteria for the file.

You complete these steps by setting logger properties. You can do this in one of two ways:

- Change or add logger properties in the `config.properties` file for a broker before you start the broker.
- Specify logger command line options in the `imqbrokerd` command that starts the broker. You can also use the broker option `-D` to change logger properties (or *any* broker property).

Options passed on the command line override properties specified in the broker instance configuration files. Table 10-3 lists the `imqbrokerd` options that affect logging.

Table 10-3 `imqbrokerd` Logger Options and Corresponding Properties

imqbrokerd Options	Description
`-metrics` *interval*	Specifies the interval (in seconds) at which metrics information is written to the logger.
`-loglevel` *level*	Sets the log level to one of `ERROR`, `WARNING`, `INFO`.
`-silent`	Turns off logging to the console.
`-tty`	Sends all messages to the console. By default only `WARNING` and `ERROR` level messages are displayed.

The following sections describe how you can change the default configuration in order to do the following:

- Change the output channel (the destination of log messages)
- Change rollover criteria

Changing the Output Channel

By default, error and warning messages are displayed on the terminal as well as being logged to a log file. (On Solaris, error messages are also written to the system's `syslog` daemon.)

You can change the output channel for log messages in the following ways:

- To have *all* log categories (for a given level) output displayed on the screen, use the `-tty` option to the `imqbrokerd` command.
- To prevent log output from being displayed on the screen, use the `-silent` option to the `imqbrokerd` command.

- Use the `imq.log.file.output` property to specify which categories of logging information should be written to the log file. For example,

  ```
  imq.log.file.output=ERROR
  ```

- Use the `imq.log.console.output` property to specify which categories of logging information should be written to the console. For example,

  ```
  imq.log.console.output=INFO
  ```

- On Solaris, use the `imq.log.syslog.output` property to specify which categories of logging information should be written to Solaris `syslog`. For example,

  ```
  imq.log.syslog.output=NONE
  ```

NOTE Before changing logger output channels, you must make sure that logging is set at a level that supports the information you are mapping to the output channel. For example, if you set the log level to `ERROR` and then set the `imq.log.console.output` property to `WARNING`, no messages will be logged because you have not enabled the logging of `WARNING` messages.

Changing Log File Rollover Criteria

There are two criteria for rolling over log files: time and size. The default is to use a time criteria and roll over files every seven days.

- To change the time interval, you need to change the property `imq.log.file.rolloversecs`. For example, the following property definition changes the time interval to ten days:

  ```
  imq.log.file.rolloversecs=864000
  ```

- To change the rollover criteria to depend on file size, you need to set the `imq.log.file.rolloverbytes` property. For example, the following definition directs the broker to rollover files after they reach a limit of 500,000 bytes

  ```
  imq.log.file.rolloverbytes=500000
  ```

If you set both the time-related and the size-related rollover properties, the first limit reached will trigger the rollover. As noted before, the broker maintains up to nine rollover files.

You can set or change the log file rollover properties when a broker is running. To set these properties, use the `imqcmd update bkr` command.

Sending Metrics Data to Log Files

This section describes the procedure for using broker log files to report metrics information. For general information on configuring the logger, see "Configuring and Using Broker Logging" on page 191.

➤ To Use Log Files to Report Metrics Information

1. Configure the broker's metrics generation capability:

   a. Confirm `imq.metrics.enabled=true`

      Generation of metrics for logging is turned on by default.

   b. Set the metrics generation interval to a convenient number of seconds.

      `imq.metrics.interval=`*interval*

      This value can be set in the `config.properties` file or using the `-metrics` *interval* command line option when starting up the broker.

2. Confirm that the logger gathers metrics information:

   `imq.log.level=INFO`

   This is the default value. This value can be set in the `config.properties` file or using the `-loglevel` *level* command line option when starting up the broker.

3. Confirm that the logger is set to write metrics information to the log file:

   `imq.log.file.output=INFO`

   This is the default value. It can be set in the `config.properties` file.

4. Start up the broker.

The following shows sample broker metrics output to the log file:

```
[21/Jul/2004:11:21:18 PDT]
Connections: 0    JVM Heap: 8323072 bytes (7226576 free) Threads: 0 (14-1010)
      In: 0 msgs (0bytes) 0 pkts (0 bytes)
     Out: 0 msgs (0bytes) 0 pkts (0 bytes)
 Rate In: 0 msgs/sec (0 bytes/sec) 0 pkts/sec (0 bytes/sec)
Rate Out: 0 msgs/sec (0 bytes/sec) 0 pkts/sec (0 bytes/sec)
```

For reference information about metrics data, see Chapter 18, "Metrics Reference."

Logging Dead Messages

You can monitor physical destinations by enabling dead message logging for a broker. You can log dead messages whether or not you are using a dead message queue.

If you enable dead message logging, the broker logs the following types of events:

- A physical destination exceeded its maximum size.
- The broker removed a message from a physical destination, for a reason such as the following:
    - The destination size limit has been reached.
    - The message time to live expired.
    - The message is too large.
    - An error occurred when the broker attempted to process the message.

If a dead message queue is in use, logging also includes the following types of events:

- The broker moved a message to the dead message queue.
- The broker removed a message from the dead message queue and discarded it.

Dead message logging is disabled by default. To enable it, set the broker attribute `imq.destination.logDeadMsgs`.

Interactively Displaying Metrics

A Message Queue broker can report the following types of metrics:

- **Java Virtual Machine (JVM) metrics**. Information about the JVM heap size.
- **Brokerwide metrics**. Information about messages stored in a broker, message flows into and out of a broker, and memory use. Messages are tracked in terms of numbers of messages and numbers of bytes.
- **Connection Service metrics**. Information about connections and connection thread resources, a nd information about message flows for a particular connection service.
- **Destination metrics**. Information about message flows into and out of a particular physical destination, information about a physical destination's consumers, and information about memory and disk space usage.

The `imqcmd` command can obtain metrics information for the broker as a whole, for individual connection services, and for individual physical destinations. To obtain metrics data, you generally use the `metrics` subcommand of `imqcmd`. Metrics data is written at an interval you specify, or the number of times you specify, to the console screen.

You can also use the `query` subcommand to view similar data that also includes configuration information. See "imqcmd query" on page 201 for more information.

imqcmd metrics

The syntax and options of `imqcmd metrics` are shown in Table 10-4 and Table 10-5, respectively.

Table 10-4 `imqcmd metrics` Subcommand Syntax

Subcommand Syntax	Metrics Data Provided

`metrics bkr`
`[-b` *hostName*`:`*portNumber*`]`
`[-m` *metricType*`]`
`[-int` *interval*`]`
`[-msp` *numSamples*`]`
	Displays broker metrics for the default broker or a broker at the specified host and port.

or	

`metrics svc -n` *serviceName*
`[-b` *hostName*`:`*portNumber*`]`
`[-m` *metricType*`]`
`[-int` *interval*`]`
`[-msp` *numSamples*`]`
	Displays metrics for the specified service on the default broker or on a broker at the specified host and port.

or	

`metrics dst -t` *destType*
`-n` *destName*
`[-b` *hostName*`:`*portNumber*`]`
`[-m` *metricType*`]`
`[-int` *interval*`]`
`[-msp` *numSamples*`]`
	Displays metrics information for the physical destination of the specified type and name.

Table 10-5 `imqcmd metrics` Subcommand Options

Subcommand Options	Description
`-b` *hostName*`:`*portNumber*	Specifies the hostname and port of the broker for which metrics data is reported. The default is `localhost:7676`.
`-int` *interval*	Specifies the interval (in seconds) at which to display the metrics. The default is 5 seconds.
`-m` *metricType*	Specifies the type of metric to display: **`ttl`** Displays metrics on messages and packets flowing into and out of the broker, service, or destination (default metric type). **`rts`** Displays metrics on rate of flow of messages and packets into and out of the broker, connection service, or destination (per second). **`cxn`** Displays connections, virtual memory heap, and threads (brokers and connection services only). **`con`** Displays consumer-related metrics (destinations only). **`dsk`** Displays disk usage metrics (destinations only).
`-msp` *numSamples*	Specifies the number of samples displayed in the output. The default is an unlimited number (infinite).
`-n` *destName*	Specifies the name of the physical destination (if any) for which metrics data is reported. There is no default.
`-n` *serviceName*	Specifies the connection service (if any) for which metrics data is reported. There is no default.
`-t` *destType*	Specifies the type (queue or topic) of the physical destination (if any) for which metrics data is reported. There is no default.

Using the metrics Subcommand to Display Metrics Data

This section describes the procedure for using the `metrics` subcommand to report metrics information.

➤ **To Use the metrics Subcommand**

1. Start the broker for which metrics information is desired.

   See "Starting Brokers" on page 66.

2. Issue the appropriate `imqcmd metrics` subcommand and options as shown in Table 10-4 and Table 10-5.

Metrics Outputs: imqcmd metrics

This section contains examples of output for the `imqcmd metrics` s subcommand. The examples show brokerwide, connection service, and physical destination metrics.

Brokerwide Metrics

To get the rate of message and packet flow into and out of the broker at 10 second intervals, use the `metrics bkr` subcommand:

```
imqcmd metrics bkr -m rts -int 10 -u admin
```

This command produces output similar to the following (see data descriptions in Table 18-2 on page 336):

```
--------------------------------------------------------
 Msgs/sec   Msg Bytes/sec   Pkts/sec    Pkt Bytes/sec
 In    Out     In      Out     In    Out      In      Out
--------------------------------------------------------
 0     0       27      56      0     0        38      66
 10    0       7365    56      10    10       7457    1132
 0     0       27      56      0     0        38      73
 0     10      27      7402    10    20       1400    8459
 0     0       27      56      0     0        38      73
```

Connection Service Metrics

To get cumulative totals for messages and packets handled by the jms connection service, use the `metrics svc` subcommand:

```
imqcmd metrics svc -n jms -m ttl -u admin
```

This command produces output similar to the following (see data descriptions in Table 18-3 on page 338):

```
-------------------------------------------------
  Msgs      Msg Bytes      Pkts      Pkt Bytes
In   Out    In      Out   In   Out    In      Out
-------------------------------------------------
164  100  120704  73600  282  383  135967  102127
657  100  483552  73600  775  876  498815  149948
```

Physical Destination Metrics

To get metrics information about a physical destination, use the `metrics dst` subcommand:

```
imqcmd metrics dst -t q -n XQueue -m ttl -u admin
```

This command produces output similar to the following (see data descriptions in Table 18-4 on page 340):

```
-----------------------------------------------------------------------------
  Msgs      Msg Bytes         Msg Count         Total Msg Bytes (k)     Largest
In   Out    In      Out    Current  Peak  Avg  Current  Peak     Avg    Msg (k)
-----------------------------------------------------------------------------
200  200  147200  147200     0      200    0      0      143      71        0
300  200  220800  147200    100     200   10     71      143      64        0
300  300  220800  220800     0      200    0      0      143      59        0
```

To get information about a physical destination's consumers, use the following `metrics dst` subcommand:

```
imqcmd metrics dst -t q -n SimpleQueue -m con -u admin
```

This command produces output similar to the following (see data descriptions in Table 18-4 on page 340):

```
------------------------------------------------------------------
  Active Consumers         Backup Consumers         Msg Count
Current  Peak  Avg      Current  Peak     Avg    Current  Peak  Avg
------------------------------------------------------------------
   1       1     0         0       0       0       944    1000  525
```

imqcmd query

The syntax and options of `imqcmd query` are shown in Table 10-6 along with a description of the metrics data provided by the command.

Table 10-6 `imqcmd query` Subcommand Syntax

Subcommand Syntax	Metrics Data Provided

`query bkr`
`[-b` *hostName*`:`*portNumber*`]`
	Information on the current number of messages and message bytes stored in broker memory and persistent store (see "Displaying Broker Information" on page 101).

or	
`query svc -n` *serviceName* `[-b` *hostName*`:`*portNumber*`]`	Information on the current number of allocated threads and number of connections for a specified connection service (see "Displaying Connection Service Information" on page 107).
or	
`query dst -t` *destType* `-n` *destName* `[-b` *hostName*`:`*portNumber*`]`	Information on the current number of producers, active and backup consumers, and messages and message bytes stored in memory and persistent store for a specified destination (see "Displaying Information about Physical Destinations" on page 121).

NOTE Because of the limited metrics data provided by `imqcmd query`, this tool is not represented in the tables presented in Chapter 18, "Metrics Reference" on page 335.

Writing an Application to Monitor Brokers

Message Queue provides a metrics monitoring capability by which the broker can write metrics data into JMS messages, which it then sends to one of a number of metrics topic destinations, depending on the type of metrics information contained in the message.

You can access this metrics information by writing a client application that subscribes to the metrics topic destinations, consumes the messages in these destinations, and processes the metrics information contained in the messages.

There are five metrics topic destinations, whose names are shown in Table 10-7, along with the type of metrics messages delivered to each destination.

Table 10-7 Metrics Topic Destinations

Topic Name	Type of Metrics Messages
mq.metrics.broker	Broker metrics
mq.metrics.jvm	Java Virtual Machine metrics
mq.metrics.destination_list	List of destinations and their types
mq.metrics.destination.queue.*monitoredDestinationName*	Destination metrics for queue of specified name
mq.metrics.destination.topic.*monitoredDestinationName*	Destination metrics for topic of specified name

Setting Up Message-Based Monitoring

This section describes the procedure for using the message-based monitoring capability to gather metrics information. The procedure includes both client development and administration tasks.

➤ To Set Up Message-based Monitoring

1. Write a metrics monitoring client.

   See the *Message Queue Developer's Guide for Java Clients* for instructions on programming clients that subscribe to metrics topic destinations, consume metrics messages, and extract the metrics data from these messages.

2. Configure the broker's Metrics Message Producer by setting broker property values in the `config.properties` file:

   a. Enable metrics message production.

      Set `imq.metrics.topic.enabled=true`

      The default value is `true`.

   b. Set the interval (in seconds) at which metrics messages are generated.

      Set `imq.metrics.topic.interval=`*interval*.

      The default is 60 seconds.

c. Specify whether you want metrics messages to be persistent (that is, whether they will survive a broker failure).

   Set `imq.metrics.topic.persist`.

   The default is `false`.

d. Specify how long you want metrics messages to remain in their respective destinations before being deleted.

   Set `imq.metrics.topic.timetolive`.

   The default value is 300 seconds.

3. Set any access control you desire on metrics topic destinations.

   See the discussion in "Security and Access Considerations," below.

4. Start up your metrics monitoring client.

   When consumers subscribe to a metrics topic, the metrics topic destination will automatically be created. Once a metrics topic has been created, the broker's metrics message producer will begin sending metrics messages to the metrics topic.

Security and Access Considerations

There are two reasons to restrict access to metrics topic destinations:

- Metrics data might include sensitive information about a broker and its resources.
- Excessive numbers of subscriptions to metrics topic destinations might increase broker overhead and negatively affect performance.

Because of these considerations, it is advisable to restrict access to metrics topic destinations.

Monitoring clients are subject to the same authentication and authorization control as any other client. Only users maintained in the Message Queue user repository are allowed to connect to the broker.

You can provide additional protections by restricting access to specific metrics topic destinations through an access control properties file, as described in "Authorizing Users: The Access Control Properties File" on page 142.

For example, the following entries in an `accesscontrol.properties` file will deny access to the mq.metrics.broker metrics topic to everyone except user1 and user 2.

```
topic.mq.metrics.broker.consume.deny.user=*
topic.mq.metrics.broker.consume.allow.user=user1,user2
```

The following entries will only allow users user3 to monitor topic t1.

```
topic.mq.metrics.destination.topic.t1.consume.deny.user=*
topic.mq.metrics.destination.topic.t1.consume.allow.user=user3
```

Depending on the sensitivity of metrics data, you can also connect your metrics monitoring client to a broker using an encrypted connection. For information on using encrypted connections, see "Working With an SSL-Based Service" on page 148.

Metrics Outputs: Metrics Messages

The metrics data outputs you get using the message-based monitoring API is a function of the metrics monitoring client you write. You are limited only by the data provided by the metrics generator in the broker. For a complete list of this data, see "Metrics Reference" on page 335.

Chapter 11

Analyzing and Tuning a Message Service

This chapter covers a number of topics about how to analyze and tune a Message Queue service to optimize the performance of your messaging applications. It includes the following topics:

About Performance

This section provides some background information on performance tuning.

The Performance Tuning Process

The performance you get out of a messaging application depends on the interaction between the application and the Message Queue service. Hence, maximizing performance requires the combined efforts of both the application developer and the administrator.

The process of optimizing performance begins with application design and continues through to tuning the message service after the application has been deployed. The performance tuning process includes the following stages:

- Defining performance requirements for the application
- Designing the application taking into account factors that affect performance (especially trade-offs between reliability and performance)

- Establishing baseline performance measures
- Tuning or reconfiguring the message service to optimize performance

The process outlined above is often iterative. During deployment of the application, a Message Queue administrator evaluates the suitability of the message server for the application's general performance requirements. If the benchmark testing meets these requirements, the administrator can tune the system as described in this chapter. However, if benchmark testing does not meet performance requirements, a redesign of the application might be necessary or the deployment architecture might need to be modified.

Aspects of Performance

In general, performance is a measure of the speed and efficiency with which a message service delivers messages from producer to consumer. However, there are several different aspects of performance that might be important to you, depending on your needs.

Connection Load The number of message producers, or message consumers, or the number of concurrent connections a system can support.

Message throughput The number of messages or message bytes that can be pumped through a messaging system per second.

Latency The time it takes a particular message to be delivered from message producer to message consumer.

Stability The overall availability of the message service or how gracefully it degrades in cases of heavy load or failure.

Efficiency The efficiency of message delivery; a measure of message throughput in relation to the computing resources employed.

These different aspects of performance are generally inter-related. If message throughput is high, that means messages are less likely to be backlogged in the message server, and as a result, latency should be low (a single message can be delivered very quickly). However, latency can depend on many factors: the speed of communication links, message server processing speed, and client processing speed, to name a few.

In any case, there are several different aspects of performance. Which of them are most important to you generally depends on the requirements of a particular application.

Benchmarks

Benchmarking is the process of creating a test suite for your messaging application and of measuring message throughput or other aspects of performance for this test suite.

For example, you could create a test suite by which some number of producing clients, using some number of connections, sessions, and message producers, send persistent or nonpersistent messages of a standard size to some number of queues or topics (all depending on your messaging application design) at some specified rate. Similarly, the test suite includes some number of consuming clients, using some number of connections, sessions, and message consumers (of a particular type) that consume the messages in the test suite's physical destinations using a particular acknowledgment mode.

Using your standard test suite you can measure the time it takes between production and consumption of messages or the average message throughput rate, and you can monitor the system to observe connection thread usage, message storage data, message flow data, and other relevant metrics. You can then ramp up the rate of message production, or the number of message producers, or other variables, until performance is negatively impacted. The maximum throughput you can achieve is a benchmark for your message service configuration.

Using this benchmark, you can modify some of the characteristics of your test suite. By carefully controlling all the factors that might have an impact on performance (see "Application Design Factors that Affect Performance" on page 210), you can note how changing some of these factors affects the benchmark. For example, you can increase the number of connections or the size of messages five-fold or ten-fold, and note the impact on performance.

Conversely, you can keep application-based factors constant and change your broker configuration in some controlled way (for example, change connection properties, thread pool properties, JVM memory limits, limit behaviors, file-based versus JDBC-based persistence, and so forth) and note how these changes affect performance.

This benchmarking of your application provides information that can be valuable when you want to increase the performance of a deployed application by tuning your message service. A benchmark allows the effect of a change or a set of changes to be more accurately predicted.

As a general rule, benchmarks should be run in a controlled test environment and for a long enough period of time for your message service to stabilize. (Performance is negatively impacted at startup by the Just-In-Time compilation that turns Java code into machine code.)

Baseline Use Patterns

Once a messaging application is deployed and running, it is important to establish baseline use patterns. You want to know when peak demand occurs and you want to be able to quantify that demand. For example, demand normally fluctuates by number of end-users, activity levels, time of day, or all of these.

To establish base-line use patterns you need to monitor your message server over an extended period of time, looking at data such as the following:

- Number of connections
- Number of messages stored in the broker (or in particular physical destinations)
- Message flows into and out of a broker (or particular physical destinations)
- Numbers of active consumers

You can also use average and peak values provided in metrics data.

It is important to check these baseline metrics against design expectations. By doing so, you are checking that client code is behaving properly: for example, that connections are not being left open or that consumed messages are not being left unacknowledged. These coding errors consume message server resources and could significantly affect performance.

The base-line use patterns help you determine how to tune your system for optimal performance. For example:

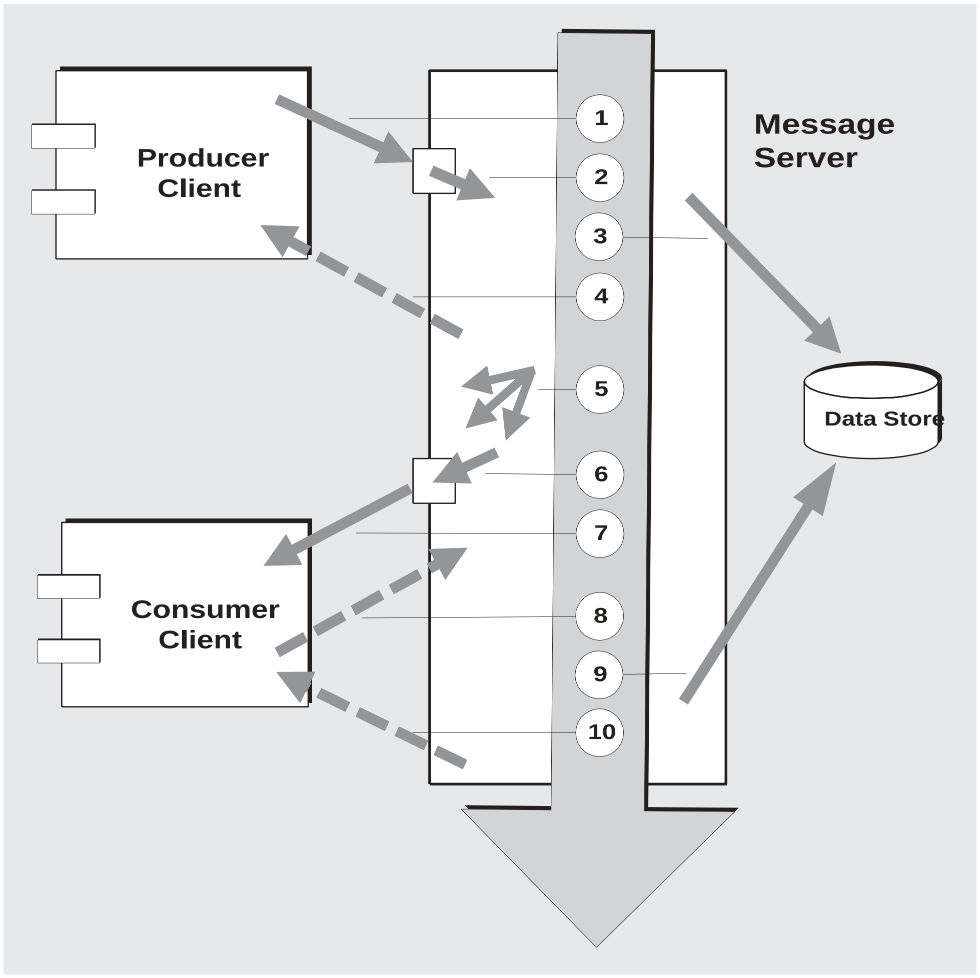
- If one physical destination is used significantly more than others, you might want to set higher message memory limits on that physical destination than on others, or to adjust limit behaviors accordingly.
- If the number of connections needed is significantly greater than allowed by the maximum thread pool size, you might want to increase the thread pool size or adopt a shared thread model.
- If peak message flows are substantially greater than average flows, that might influence the limit behaviors you employ when memory runs low.

In general, the more you know about use patterns, the better you are able to tune your system to those patterns and to plan for future needs.

Factors That Affect Performance

Message latency and message throughput, two of the main performance indicators, generally depend on the time it takes a typical message to complete various steps in the message delivery process. These steps are shown below for the case of a persistent, reliably delivered message. The steps are described following the illustration.

Figure 11-1 Message Delivery Through a Message Queue Service



1. The message is delivered from producing client to message server.
2. The message server reads in the message.
3. The message is placed in persistent storage (for reliability).
4. The message server confirms receipt of the message (for reliability).
5. The message server determines the routing for the message.
6. The message server writes out the message.
7. The message is delivered from message server to consuming client.
8. The consuming client acknowledges receipt of the message (for reliability).
9. The message server processes client acknowledgment (for reliability).
10. The message server confirms that client acknowledgment has been processed.

Since these steps are sequential, any step can be a potential bottleneck in the delivery of messages from producing clients to consuming clients. Most of these steps depend upon physical characteristics of the messaging system: network bandwidth, computer processing speeds, message server architecture, and so forth. Some, however, also depend on characteristics of the messaging application and the level of reliability it requires.

The following subsections discuss the impact of both application design factors and messaging system factors on performance. While application design and messaging system factors closely interact in the delivery of messages, each category is considered separately.

Application Design Factors that Affect Performance

Application design decisions can have a significant effect on overall messaging performance.

The most important factors affecting performance are those that impact the reliability of message delivery. Among these are the following factors:

- Delivery Mode (Persistent/Nonpersistent Messages)
- Use of Transactions
- Acknowledgment Mode
- Durable and Non-durable Subscriptions

Other application design factors impacting performance are the following:

- Use of Selectors (Message Filtering)
- Message Size
- Message Body Type

The sections that follow describe the impact of each of these factors on messaging performance. As a general rule, there is a trade-off between performance and reliability: factors that increase reliability tend to decrease performance.

Table 11-1 shows how the various application design factors generally affect messaging performance. The table shows two scenarios—a high reliability, low performance scenario and a high performance, low reliability scenario—and the choice of application design factors that characterizes each. Between these extremes, there are many choices and trade-offs that affect both reliability and performance.

Table 11-1 Comparison of High Reliability and High Performance Scenarios

Application Design Factor	High Reliability Low Performance Scenario	High Performance Low Reliability Scenario
Delivery mode	Persistent messages	nonpersistent messages
Use of transactions	Transacted sessions	No transactions
acknowledgment mode	`AUTO_ACKNOWLEDGE` or `CLIENT_ACKNOWLEDGE`	`DUPS_OK_ACKNOWLEDGE`
Durable/non-durable subscriptions	Durable subscriptions	Non-durable subscriptions
Use of selectors	Message filtering	No message filtering
Message size	Large number of small messages	Small number of large messages
Message body type	Complex body types	Simple body types

NOTE In the graphs that follow, performance data were generated on a two-CPU, 1002 Mhz, Solaris 8 system, using file-based persistence. The performance test first warmed up the Message Queue broker, allowing the Just-In-Time compiler to optimize the system and the persistent database to be primed.

Once the broker was warmed up, a single producer and single consumer were created and messages were produced for 30 seconds. The time required for the consumer to receive all produced messages was recorded, and a throughput rate (messages per second) was calculated. This scenario was repeated for different combinations of the application design factors shown in Table 11-1.

Delivery Mode (Persistent/Nonpersistent Messages)

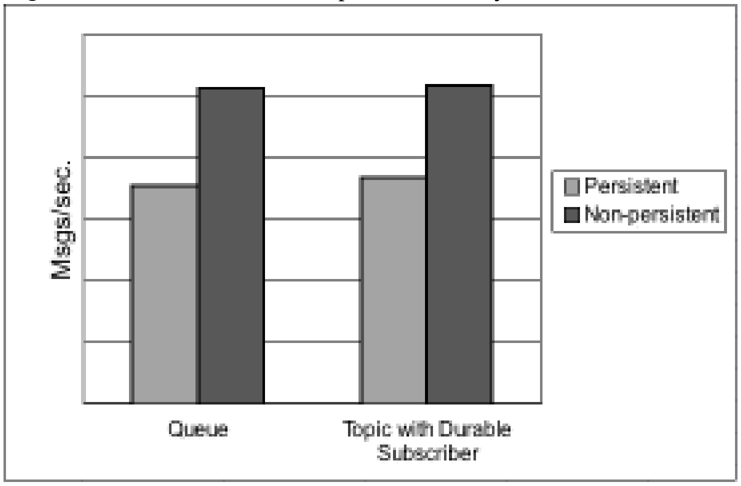
Persistent messages guarantee message delivery in case of message server failure. The broker stores the message in a persistent store until all intended consumers acknowledge they have consumed the message.

Broker processing of persistent messages is slower than for nonpersistent messages for the following reasons:

- A broker must reliably store a persistent message so that it will not be lost should the broker fail.
- The broker must confirm receipt of each persistent message it receives. Delivery to the broker is guaranteed once the method producing the message returns without an exception.
- Depending on the client acknowledgment mode, the broker might need to confirm a consuming client’s acknowledgment of a persistent message.

The differences in performance between the persistent and nonpersistent modes can be significant. Figure 11-2 compares throughput for persistent and nonpersistent messages in two reliable delivery cases: 10k-sized messages delivered both to a queue and to a topic with durable subscriptions. Both cases use the `AUTO_ACKNOWLEDGE` acknowledgment mode.

Figure 11-2 Performance Impact of Delivery Modes



Use of Transactions

A transaction is a guarantee that all messages produced in a transacted session and all messages consumed in a transacted session will be either processed or not processed (rolled back) as a unit.

Message Queue supports both local and distributed transactions.

A message produced or acknowledged in a transacted session is slower than in a non-transacted session for the following reasons:

- Additional information must be stored with each produced message.
- In some situations, messages in a transaction are stored when normally they would not be (for example, a persistent message delivered to a topic destination with no subscriptions would normally be deleted, however, at the time the transaction is begun, information about subscriptions is not available).
- Information on the consumption and acknowledgment of messages within a transaction must be stored and processed when the transaction is committed.

Acknowledgment Mode

One mechanism for ensuring the reliability of JMS message delivery is for a client to acknowledge consumption of messages delivered to it by the Message Queue message server.

If a session is closed without the client acknowledging the message or if the message server fails before the acknowledgment is processed, the broker redelivers that message, setting a `JMSRedelivered` flag.

For a non-transacted session, the client can choose one of three acknowledgment modes, each of which has its own performance characteristics:

- `AUTO_ACKNOWLEDGE`. The system automatically acknowledges a message once the consumer has processed it. This mode guarantees at most one redelivered message after a provider failure.
- `CLIENT_ACKNOWLEDGE`. The application controls the point at which messages are acknowledged. All messages processed in that session since the previous acknowledgment are acknowledged. If the message server fails while processing a set of acknowledgments, one or more messages in that group might be redelivered.
- `DUPS_OK_ACKNOWLEDGE`. This mode instructs the system to acknowledge messages in a lazy manner. Multiple messages can be redelivered after a provider failure.

(Using `CLIENT_ACKNOWLEDGE` mode is similar to using transactions, except there is no guarantee that all acknowledgments will be processed together if a provider fails during processing.)

Acknowledgment mode affects performance for the following reasons:

- Extra control messages between broker and client are required in `AUTO_ACKNOWLEDGE` and `CLIENT_ACKNOWLEDGE` modes. The additional control messages add additional processing overhead and can interfere with JMS payload messages, causing processing delays.
- In `AUTO_ACKNOWLEDGE` and `CLIENT_ACKNOWLEDGE` modes, the client must wait until the broker confirms that it has processed the client's acknowledgment before the client can consume additional messages. (This broker confirmation guarantees that the broker will not inadvertently redeliver these messages.)
- The Message Queue persistent store must be updated with the acknowledgment information for all persistent messages received by consumers, thereby decreasing performance.

Durable and Non-durable Subscriptions

Subscribers to a topic destination fall into two categories, those with durable and non-durable subscriptions.

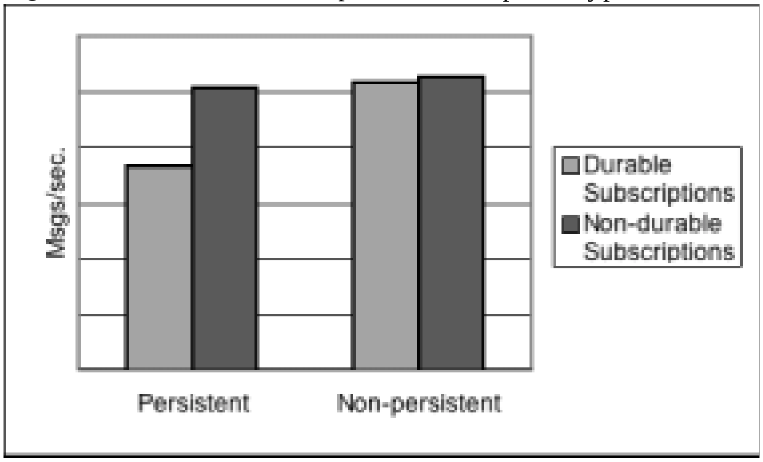
Durable subscriptions provide increased reliability but slower throughput, for the following reasons:

- The Message Queue message server must persistently store the list of messages assigned to each durable subscription so that should a message server fail, the list is available after recovery.
- Persistent messages for durable subscriptions are stored persistently, so that should a message server fail, the messages can still be delivered after recovery, when the corresponding consumer becomes active. By contrast, persistent messages for non-durable subscriptions are not stored persistently (should a message server fail, the corresponding consumer connection is lost and the message would never be delivered).

Figure 11-3 compares throughput for topic destinations with durable and non-durable subscriptions in two cases: persistent and nonpersistent 10k-sized messages. Both cases use `AUTO_ACKNOWLEDGE` acknowledgment mode.

You can see from Figure 11-3 that the performance impact of using durable subscriptions is manifest only in the case of persistent messages; and the impact in that case is because persistent messages are only stored persistently for durable subscriptions, as explained above.

Figure 11-3 Performance Impact of Subscription Types



Use of Selectors (Message Filtering)

Application developers often want to target sets of messages to particular consumers. They can do so either by targeting each set of messages to a unique physical destination or by using a single physical destination and registering one or more selectors for each consumer.

A selector is a string requesting that only messages with property values that match the string are delivered to a particular consumer. For example, the selector `NumberOfOrders >1` delivers only the messages with a `NumberOfOrders` property value of 2 or more.

Registering consumers with selectors lowers performance (as compared to using multiple physical destinations) because additional processing is required to handle each message. When a selector is used, it must be parsed so that it can be matched against future messages. Additionally, the message properties of each message must be retrieved and compared against the selector as each message is routed. However, using selectors provides more flexibility in a messaging application.

Message Size

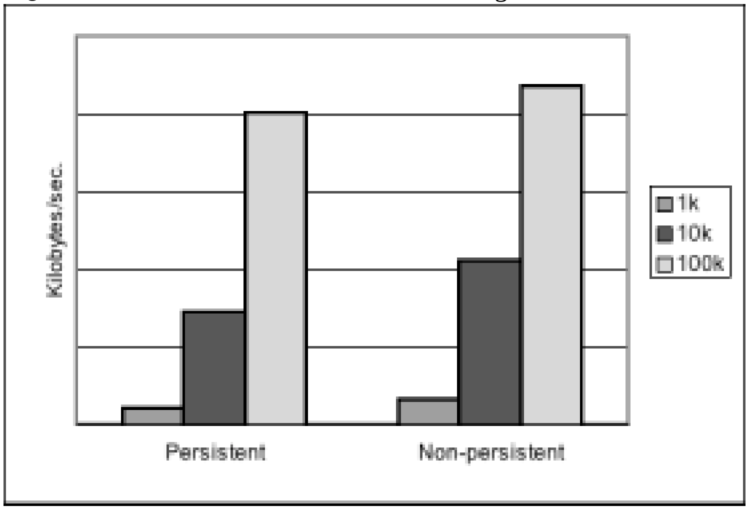
Message size affects performance because more data must be passed from producing client to broker and from broker to consuming client, and because for persistent messages a larger message must be stored.

However, by batching smaller messages into a single message, the routing and processing of individual messages can be minimized, providing an overall performance gain. In this case, information about the state of individual messages is lost.

Figure 11-4 compares throughput in kilobytes per second for 1k, 10k, and 100k-sized messages in two cases: persistent and nonpersistent messages. All cases send messages are to a queue destination and use `AUTO_ACKNOWLEDGE` acknowledgment mode.

Figure 11-4 shows that in both cases there is less overhead in delivering larger messages compared to smaller messages. You can also see that the almost 50% performance gain of nonpersistent messages over persistent messages shown for 1k and 10k-sized messages is not maintained for 100k-sized messages, probably because network bandwidth has become the bottleneck in message throughput for that case.

Figure 11-4 Performance Effect of a Message Size



Message Body Type

JMS supports five message body types, shown below roughly in the order of complexity:

- BytesMessage: Contains a set of bytes in a format determined by the application.
- TextMessage: Is a simple java.lang.String.
- StreamMessage: Contains a stream of Java primitive values.
- MapMessage: Contains a set of name-and-value pairs.
- ObjectMessage: Contains a Java serialized object.

While, in general, the message type is dictated by the needs of an application, the more complicated types (MapMessage and ObjectMessage) carry a performance cost—the expense of serializing and deserializing the data. The performance cost depends on how simple or how complicated the data is.

Message Service Factors that Affect Performance

The performance of a messaging application is affected not only by application design, but also by the message service performing the routing and delivery of messages.

The following sections discuss various message service factors that can affect performance. Understanding the impact of these factors is key to sizing a message service and diagnosing and resolving performance bottlenecks that might arise in a deployed application.

The most important factors affecting performance in a Message Queue service are the following:

- Hardware
- Operating System
- Java Virtual Machine (JVM)
- Connections
- Broker Limits and Behaviors
- Message Server Architecture
- Data Store Performance
- Client Runtime Configuration

The sections below describe the impact of each of these factors on messaging performance.

Hardware

For both the Message Queue message server and client applications, CPU processing speed and available memory are primary determinants of message service performance. Many software limitations can be eliminated by increasing processing power, while adding memory can increase both processing speed and capacity. However, it is generally expensive to overcome bottlenecks simply by upgrading your hardware.

Operating System

Because of the efficiencies of different operating systems, performance can vary, even assuming the same hardware platform. For example, the thread model employed by the operating system can have an important impact on the number of concurrent connections a message server can support. In general, all hardware being equal, Solaris is generally faster than Linux, which is generally faster than Windows.

Java Virtual Machine (JVM)

The message server is a Java process that runs in and is supported by the host JVM. As a result, JVM processing is an important determinant of how fast and efficiently a message server can route and deliver messages.

In particular, the JVM's management of memory resources can be critical. Sufficient memory has to be allocated to the JVM to accommodate increasing memory loads. In addition, the JVM periodically reclaims unused memory, and this memory reclamation can delay message processing. The larger the JVM memory heap, the longer the potential delay that might be experienced during memory reclamation.

Connections

The number and speed of connections between client and broker can affect the number of messages that a message server can handle as well as the speed of message delivery.

Message Server Connection Limits

All access to the message server is by way of connections. Any limit on the number of concurrent connections can affect the number of producing or consuming clients that can concurrently use the message server.

The number of connections to a message server is generally limited by the number of threads available. Message Queue can be configured to support either a dedicated thread model or a shared thread model (see "Thread Pool Management" on page 77).

The dedicated thread model is very fast because each connection has dedicated threads, however the number of connections is limited by the number of threads available (one input thread and one output thread for each connection). The shared thread model places no limit on the number of connections, however there is significant overhead and throughput delays in sharing threads among a number of connections, especially when those connections are busy.

Transport Protocols

Message Queue software allows clients to communicate with the message server using various low-level transport protocols. Message Queue supports the connection services (and corresponding protocols) described in "Connection Services" on page 76.

The choice of protocols is based on application requirements (encrypted, accessible through a firewall), but the choice impacts overall performance.

Figure 11-5 Transport Protocol Speeds

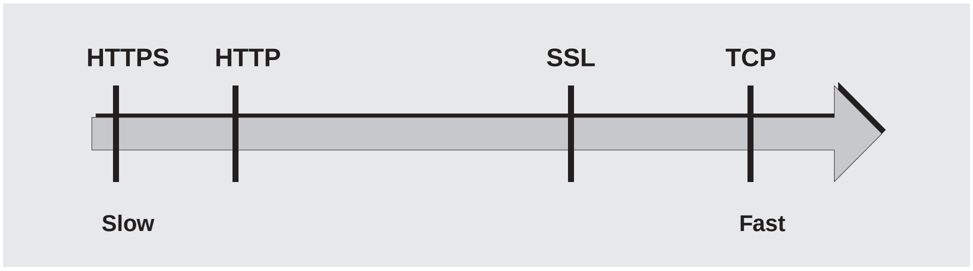


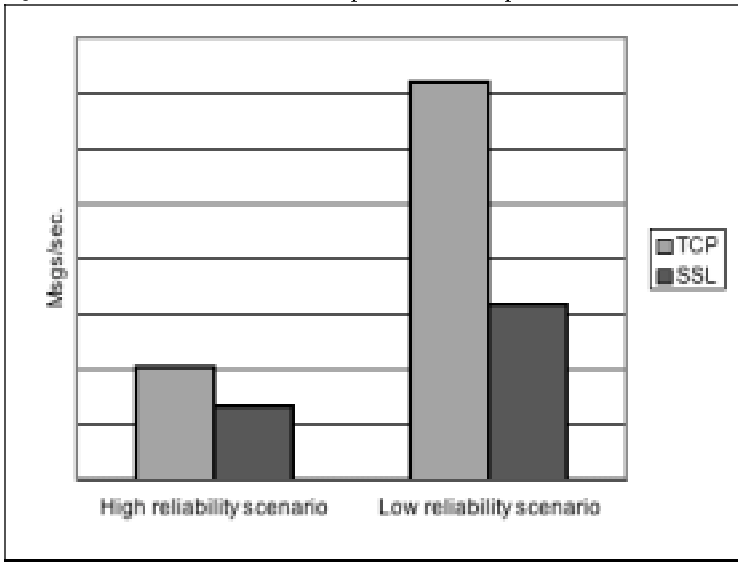
Figure 11-5 reflects the performance characteristics of the various protocol technologies:

- TCP provides the fastest method to communicate with the broker.
- SSL is 50 to 70 percent slower than TCP when it comes to sending and receiving messages (50 percent for persistent messages, closer to 70 percent for nonpersistent messages). Additionally, establishing the initial connection is slower with SSL (it might take several seconds) because the client and broker (or Web Server in the case of HTTPS) need to establish a private key to be used when encrypting the data for transmission. The performance drop is caused by the additional processing required to encrypt and decrypt each low-level TCP packet.

  Figure 11-6 compares throughput for TCP and SSL for two cases: a high reliability scenario (1k persistent messages sent to topic destinations with durable subscriptions and using `AUTO_ACKNOWLEDGE` acknowledgment mode) and a high performance scenario (1k nonpersistent messages sent to topic destinations without durable subscriptions and using `DUPS_OK_ACKNOWLEDGE` acknowledgment mode).

  Figure 11-6 shows that protocol has less impact in the high reliability case. This is probably because the persistence overhead required in the high reliability case is a more important factor in limiting throughput than the protocol speed.

Figure 11-6 Performance Impact of Transport Protocol



- HTTP is slower than either the TCP or SSL. It uses a servlet that runs on a Web server as a proxy between the client and the broker. Performance overhead is involved in encapsulating packets in HTTP requests and in the requirement that messages go through two hops--client to servlet, servlet to broker--to reach the broker.
- HTTPS is slower than HTTP because of the additional overhead required to encrypt the packet between client and servlet and between servlet and broker.

Message Server Architecture

A Message Queue message server can be implemented as a single broker or as multiple interconnected broker instances—a broker cluster.

As the number of clients connected to a broker increases, and as the number of messages being delivered increases, a broker will eventually exceed resource limitations such as file descriptor, thread, and memory limits. One way to accommodate increasing loads is to add more broker instances to a Message Queue message server, distributing client connections and message routing and delivery across multiple brokers.

In general, this scaling works best if clients are evenly distributed across the cluster, especially message producing clients. Because of the overhead involved in delivering messages between the brokers in a cluster, clusters with limited numbers of connections or limited message delivery rates, might exhibit lower performance than a single broker.

You might also use a broker cluster to optimize network bandwidth. For example, you might want to use slower, long distance network links between a set of remote brokers within a cluster, while using higher speed links for connecting clients to their respective broker instances.

For more information on clusters, see Chapter 9, "Working With Broker Clusters."

Broker Limits and Behaviors

The message throughput that a message server might be required to handle is a function of the use patterns of the messaging applications the message server supports. However, the message server is limited in resources: memory, CPU cycles, and so forth. As a result, it would be possible for a message server to become overwhelmed to the point where it becomes unresponsive or unstable.

The Message Queue message server has mechanisms built in for managing memory resources and preventing the broker from running out of memory. These mechanisms include configurable limits on the number of messages or message bytes that can be held by a broker or its individual physical destinations, and a set of behaviors that can be instituted when physical destination limits are reached.

With careful monitoring and tuning, these configurable mechanisms can be used to balance the inflow and outflow of messages so that system overload cannot occur. While these mechanisms consume overhead and can limit message throughput, they nevertheless maintain operational integrity.

Data Store Performance

Message Queue supports both file-based and JDBC-based persistence modules. File-based persistence uses individual files to store persistent data. JDBC-based persistence uses a Java Database Connectivity (JDBC™) interface and requires a JDBC-compliant data store. File-based persistence is generally faster than JDBC-based; however, some users prefer the redundancy and administrative control provided by a JDBC-compliant store.

In the case of file-based persistence, you can maximize reliability by specifying that persistence operations synchronize the in-memory state with the data store. This helps eliminate data loss due to system crashes, but at the expense of performance.

Client Runtime Configuration

The Message Queue client runtime provides client applications with an interface to the Message Queue message service. It supports all the operations needed for clients to send messages to physical destinations and to receive messages from such destinations. The client runtime is configurable (by setting connection factory attribute values), allowing you to control aspects of its behavior, such as connection flow metering, consumer flow limits, and connection flow limits, that can improve performance and message throughput. See "Client Runtime Message Flow Adjustments" on page 229 for more information on these features and the attributes used to configure them.

Adjusting Configuration To Improve Performance

System Adjustments

The following sections describe adjustments you can make to the operating system, JVM, and communication protocols.

Solaris Tuning: CPU Utilization, Paging/Swapping/Disk I/O

See your system documentation for tuning your operating system.

Java Virtual Machine Adjustments

By default, the broker uses a JVM heap size of 192MB. This is often too small for significant message loads and should be increased.

When the broker gets close to exhausting the JVM heap space used by Java objects, it uses various techniques such as flow control and message swapping to free memory. Under extreme circumstances it even closes client connections in order to free the memory and reduce the message inflow. Hence it is desirable to set the maximum JVM heap space high enough to avoid such circumstances.

However, if the maximum Java heap space is set too high, in relation to system physical memory, the broker can continue to grow the Java heap space until the entire system runs out of memory. This can result in diminished performance, unpredictable broker crashes, and / or affect the behavior of other applications and services running on the system. In general, you need to allow enough physical memory for the operating system and other applications to run on the machine.

In general it is a good idea to evaluate the normal and peak system memory footprints, and configure the Java heap size so that it is large enough to provide good performance, but not so large as to risk system memory problems.

To change the minimum and maximum heap size for the broker, use the `-vmargs` command line option when starting the broker. For example:

```
/usr/bin/imqbrokerd -vmargs "-Xms256m -Xmx1024m"
```

This command will set the starting Java heap size to 256MB and the maximum Java heap size to 1GB.

- On Solaris or Linux, if starting the broker via `/etc/rc*` (that is, `/etc/init.d/imq`), specify broker command line arguments in the file `/etc/imq/imqbrokerd.conf` (Solaris) or `/etc/opt/sun/mq/imqbrokerd.conf` (Linux). See the comments in that file for more information.

- On Windows, if starting the broker as a Window's service, specify JVM arguments using the `-vmargs` option to the `imqsvcadmin install` command. See "Service Administrator Utility" in Chapter 13, "Command Line Reference."

In any case, verify settings by checking the broker's log file or using the `imqcmd metrics bkr -m cxn` command.

Tuning Transport Protocols

Once a protocol that meets application needs has been chosen, additional tuning (based on the selected protocol) might improve performance.

A protocol's performance can be modified using the following three broker properties:

- `imq.protocol.`*protocolType*`.nodelay`
- `imq.protocol.`*protocolType*`.inbufsz`
- `imq.protocol.`*protocolType*`.outbufsz`

For TCP and SSL protocols, these properties affect the speed of message delivery between client and broker. For HTTP and HTTPS protocols, these properties affect the speed of message delivery between the Message Queue tunnel servlet (running on a Web server) and the broker. For HTTP / HTTPS protocols there are additional properties that can affect performance (see "HTTP / HTTPS Tuning" on page 227).

The protocol tuning properties are described in the following sections.

nodelay

The `nodelay` property affects Nagle's algorithm (the value of the `TCP_NODELAY` socket-level option on TCP / IP) for the given protocol. Nagle's algorithm is used to improve TCP performance on systems using slow connections such as wide-area networks (WANs).

When the algorithm is used, TCP tries to prevent several small chunks of data from being sent to the remote system (by bundling the data in larger packets). If the data written to the socket does not fill the required buffer size, the protocol delays sending the packet until either the buffer is filled or a specific delay time has elapsed. Once the buffer is full or the time-out has occurred, the packet is sent.

For most messaging applications, performance is best if there is no delay in the sending of packets (Nagle's algorithm is not enabled). This is because most interactions between client and broker are request / response interactions: the client sends a packet of data to the broker and waits for a response. For example, typical interactions include:

- Creating a connection
- Creating a producer or consumer
- Sending a persistent message (the broker confirms receipt of the message)
- Sending a client acknowledgment in an `AUTO_ACKNOWLEDGE` or `CLIENT_ACKNOWLEDGE` session (the broker confirms processing of the acknowledgment)

For these interactions, most packets are smaller than the buffer size. This means that if Nagle's algorithm is used, the broker delays several milliseconds before sending a response to the consumer.

However, Nagle's algorithm may improve performance in situations where connections are slow and broker responses are not required. This would be the case where a client sends a nonpersistent message or where a client acknowledgment is not confirmed by the broker (`DUPS_OK_ACKNOWLEDGE` session).

inbufsz/outbufsz

The `inbufsz` property sets the size of the buffer on the input stream reading data coming in from a socket. Similarly, `outbufsz` sets the buffer size of the output stream used by the broker to write data to the socket.

In general, both parameters should be set to values that are slightly larger than the average packet being received or sent. A good rule of thumb is to set these property values to the size of the average packet plus 1k (rounded to the nearest k).

For example, if the broker is receiving packets with a body size of 1k, the overall size of the packet (message body + header + properties) is about 1200 bytes. An `inbufsz` of 2k (2048 bytes) gives reasonable performance.

Increasing the inbufsz or outbufsz greater than that size may improve performance slightly; however, it increases the memory needed for each connection.

Figure 11-7 shows the consequence of changing `inbufsz` on a 1k packet.

Figure 11-7 Effect of Changing `inbufsz` on a 1k (1024 bytes) Packet

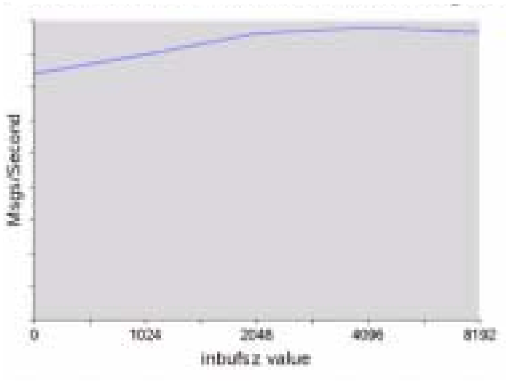
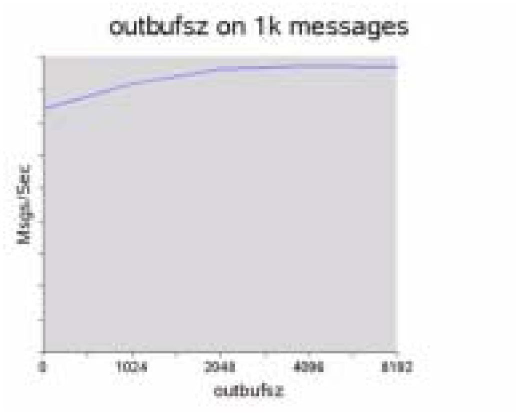


Figure 11-8 shows the consequence of changing `outbufsz` on a 1k packet.

Figure 11-8 Effect of Changing `outbufsz` on a 1k (1024 bytes) Packet



HTTP/HTTPS Tuning

In addition to the general properties discussed in the previous two sections, HTTP / HTTPS performance is limited by how fast a client can make HTTP requests to the Web server hosting the Message Queue tunnel servlet.

A Web server might need to be optimized to handle multiple requests on a single socket. With JDK version 1.4 and later, HTTP connections to a Web server are kept alive (the socket to the Web server remains open) to minimize resources used by the Web server when it processes multiple HTTP requests. If the performance of a client application using JDK version 1.4 is slower than the same application running with an earlier JDK release, you might need to tune the Web server keep-alive configuration parameters to improve performance.

In addition to such Web-server tuning, you can also adjust how often a client polls the Web server. HTTP is a request-based protocol. This means that clients using an HTTP-based protocol periodically need to check the Web server to see if messages are waiting. The `imq.httpjms.http.pullPeriod` broker property (and the corresponding `imq.httpsjms.https.pullPeriod` property) specifies how often the Message Queue client runtime polls the Web server.

If the `pullPeriod` value is `-1` (the default value), the client runtime polls the server as soon as the previous request returns, maximizing the performance of the individual client. As a result, each client connection monopolizes a request thread in the Web server, possibly straining Web server resources.

If the `pullPeriod` value is a positive number, the client runtime periodically sends requests to the Web server to see if there is pending data. In this case, the client does not monopolize a request thread in the Web server. Hence, if large numbers of clients are using the Web server, you might conserve Web server resources by setting the `pullPeriod` to a positive value.

Tuning the File-based Persistent Store

For information on tuning the file-based persistent store, see "Persistence Services" on page 80.

Broker Adjustments

The following sections describe adjustments you can make to broker properties to improve performance.

Memory Management: Increasing Broker Stability Under Load

Memory management can be configured on a destination-by-destination level or on a systemwide level (for all destinations, collectively).

Using Physical Destination Limits

For information on physical destination limits, see Chapter 6, "Managing Physical Destinations."

Using Systemwide Limits

If message producers tend to overrun message consumers, messages can accumulate in the broker. The broker contains a mechanism for throttling back producers and swapping messages out of active memory in low memory conditions, but it is wise to set a hard limit on the total number of messages (and message bytes) that the broker can hold.

Control these limits by setting the `imq.system.max_count` and the `imq.system.max_size` broker properties.

For example:

```
imq.system.max_count=5000
```

The defined value above means that the broker will only hold up to 5000 undelivered / unacknowledged messages. If additional messages are sent, they are rejected by the broker. If a message is persistent then the producer will get an exception when it tries to send the message. If the message is nonpersistent, the broker silently drops the message.

When an exception is returned in sending a message, the client should pause for a moment and retry the send again. (Note that the exception will never be due to the broker's failure to receive a message; the only exceptions raised are those detected by the client on the sending side.)

Multiple Consumer Queue Performance

The efficiency with which multiple queue consumers process messages in a queue destination depends on the following configurable queue destination attributes:

- The number of active consumers (`maxNumActiveConsumers`)
- The maximum number of messages that can be delivered to a consumer in a single batch (`consumerFlowLimit`)

To achieve optimal message throughput there must be a sufficient number of active consumers to keep up with the rate of message production for the queue, and the messages in the queue must be routed and then delivered to the active consumers in such a way as to maximize their rate of consumption. The general mechanism for balancing message delivery among multiple consumers is described in the *Sun Java System Message Queue Technical Overview*.

If messages are accumulating in the queue, it is possible that there is an insufficient number of active consumers to handle the message load. It is also possible that messages are being delivered to the consumers in batch sizes that cause messages to be backing up on the consumers. For example, if the batch size (`consumerFlowLimit`) is too large, one consumer might receive all the messages in a queue while other active consumers receive none. If consumers are very fast, this might not be a problem.

However, if consumers are relatively slow, you want messages to be distributed to them evenly, and therefore you want the batch size to be small. The smaller the batch size, the more overhead is required to deliver messages to consumers. Nevertheless, for slow consumers, there is generally a net performance gain to using small batch sizes.

Client Runtime Message Flow Adjustments

This section discusses flow control behaviors that affect performance (see “Client Runtime Configuration” on page 223). These behaviors are configured as attributes of connection factory administered objects. For information on setting connection factory attributes, see Chapter 8, “Managing Administered Objects.”

Message Flow Metering

Messages sent and received by clients *(payload messages)*, as well as Message Queue control messages, pass over the same client-broker connection. Delays in the delivery of control messages, such as broker acknowledgments, can result if control messages are held up by the delivery of payload messages. To prevent this type of congestion, Message Queue meters the flow of payload messages across a connection.

Payload messages are batched (as specified with the connection factory attribute `imqConnectionFlowCount`) so that only a set number are delivered. After the batch has been delivered, delivery of payload messages is suspended and only pending control messages are delivered. This cycle repeats, as additional batches of payload messages are delivered followed by pending control messages.

The value of `imqConnectionFlowCount` should be kept low if the client is doing operations that require many responses from the broker: for example, if the client is using `CLIENT_ACKNOWLEDGE` or `AUTO_ACKNOWLEDGE` mode, persistent messages, transactions, or queue browsers, or is adding or removing consumers. If, on the other hand, the client has only simple consumers on a connection using `DUPS_OK_ACKNOWLEDGE` mode, you can increase `imqConnectionFlowCount` without compromising performance.

Message Flow Limits

There is a limit to the number of payload messages that the Message Queue client runtime can handle before encountering local resource limitations, such as memory. When this limit is approached, performance suffers. Hence, Message Queue lets you limit the number of messages per consumer (or messages per connection) that can be delivered over a connection and buffered in the client runtime, waiting to be consumed.

Consumer Flow Limiits

When the number of payload messages delivered to the client runtime exceeds the value of `imqConsumerFlowLimit` for any consumer, message delivery for that consumer stops. It is resumed only when the number of unconsumed messages for that consumer drops below the value set with `imqConsumerFlowThreshold`.

The following example illustrates the use of these limits: consider the default settings for topic consumers:

```
imqConsumerFlowLimit=1000
```

```
imqConsumerFlowThreshold=50
```

When the consumer is created, the broker delivers an initial batch of 1000 messages (providing they exist) to this consumer without pausing. After sending 1000 messages, the broker stops delivery until the client runtime asks for more messages. The client runtime holds these messages until the application processes them. The client runtime then allows the application to consume at least 50% (`imqConsumerFlowThreshold`) of the message buffer capacity (i.e. 500 messages) before asking the broker to send the next batch.

In the same situation, if the threshold were 10%, the client runtime would wait for the application to consume at least 900 messages before asking for the next batch.

The next batch size is calculated as follows:

`imqConsumerFlowLimit - (`*current number of pending msgs in buffer*`)`

So if `imqConsumerFlowThreshold` is 50%, the next batch size can fluctuate between 500 and 1000, depending on how fast the application can process the messages.

If the `imqConsumerFlowThreshold` is set too high (close to 100%), the broker will tend to send smaller batches, which can lower message throughput. If the value is set too low (close to 0%), the client may be able to finish processing the remaining buffered messages before the broker delivers the next set, again degrading message throughput. Generally speaking, unless you have specific performance or reliability concerns, you will not need to change the default value of `imqConsumerFlowThreshold` attribute.

The consumer-based flow controls (in particular, `imqConsumerFlowLimit`) are the best way to manage memory in the client runtime. Generally, depending on the client application, you know the number of consumers you need to support on any connection, the size of the messages, and the total amount of memory that is available to the client runtime.

Connection Flow Limiits

In the case of some client applications, however, the number of consumers may be indeterminate, depending on choices made by end users. In those cases, you can still manage memory using connection-level flow limits.

Connection-level flow controls limit the total number of messages buffered for *all* consumers on a connection. If this number exceeds the value of `imqConnectionFlowLimit`, delivery of messages through the connection stops until that total drops below the connection limit. (The `imqConnectionFlowLimit` attribute is enabled only if you set `imqConnectionFlowLimitEnabled` to `true`.)

The number of messages queued up in a session is a function of the number of message consumers using the session and the message load for each consumer. If a client is exhibiting delays in producing or consuming messages, you can normally improve performance by redesigning the application to distribute message producers and consumers among a larger number of sessions or to distribute sessions among a larger number of connections.

Chapter 12

Troubleshooting Problems

This chapter explains how to understand and resolve the following problems:

When problems occur, it is useful to check the version number of the installed Message Queue software. Use the version number to ensure that you are using documentation whose version matches the software version. You also need the version number to report a problem to Sun. To check the version number, issue the following command:

```
imqcmd -v
```

A Client Cannot Establish a Connection

The symptoms of this problem are as follows:

- Client cannot make a new connection.
- Client cannot auto-reconnect on failed connection.

This section explores the following possible causes:

- Client applications are not closing connections, causing the number of connections to exceed resource limitations.
- Broker is not running or there is a network connectivity problem.
- Connection service is inactive or paused.
- Too few threads available for the number of connections required.
- Too few file descriptors for the number of connections required on the Solaris or Linux operating system.
- TCP backlog limits the number of simultaneous new connection requests that can be established.
- Operating system limits the number of concurrent connections.
- Authentication or authorization of the user is failing.

Client applications are not closing connections, causing the number of connections to exceed resource limitations

To confirm this cause of the problem

List all connections to a broker:

```
imqcmd list cxn
```

The output will list all connections and the host from which each connection has been made, revealing an unusual number of open connections for specific clients.

To resolve the problem

Rewrite the offending clients to close unused connections.

Broker is not running or there is a network connectivity problem

To confirm this cause of the problem

- Telnet to the broker's primary port (for example, the default of `7676`) and verify that the broker responds with Port Mapper output.
- Verify that the broker process is running on the host.

To resolve the problem

- Start up the broker.
- Fix the network connectivity problem.

Connection service is inactive or paused

To confirm this cause of the problem

Check the status of all connection services:

```
imqcmd list svc
```

If the status of a connection service is shown as `unknown` or `paused`, clients will not be able to establish a connection using that service.

To resolve the problem

- If the status of a connection service is shown as `unknown`, it is missing from the active service list (`imq.service.active`). In the case of SSL-based services, the service might also be improperly configured, causing the broker to make the following entry in the broker log: `ERROR [B3009]: Unable to start service ssljms: [B4001]: Unable to open protocol tls for ssljms service...` followed by an explanation of the underlying cause of the exception.

  To properly configure SSL services, see "Working With an SSL-Based Service" on page 148.

- If the status of a connection service is shown as `paused`, resume the service (see "Pausing and Resuming a Connection Service" on page 110).

Too few threads available for the number of connections required

To confirm this cause of the problem

Check for the following entry in the broker log:

```
WARNING [B3004]: No threads are available to process a new connection on
service ... Closing the new connection.
```

Also check the number of connections on the connection service and the number of threads currently in use, using one of the following formats:

```
imqcmd query svc -n serviceName
```

```
imqcmd metrics svc -n serviceName -m cxn
```

Each connection requires two threads: one for incoming messages and one for outgoing messages (see "Thread Pool Management" on page 77).

To resolve the problem

- If you are using a dedicated thread pool model (`imq.`*serviceName*`.threadpool_model=dedicated`), the maximum number of connections is half the maximum number of threads in the thread pool. Therefore, to increase the number of connections, increase the size of the thread pool (`imq.`*serviceName*`.max_threads`) or switch to the shared thread pool model.
- If you are using a shared thread pool model (`imq.`*serviceName*`.threadpool_model=shared`), the maximum number of connections is half the product of the following two properties: the connection Monitor limit (`imq.`*serviceName*`.connectionMonitor_limit`) and the maximum number of threads (`imq.`*serviceName*`.max_threads`). Therefore, to increase the number of connections, increase the size of the thread pool or increase the connection monitor limit.
- Ultimately, the number of supportable connections (or the throughput on connections) will reach input/output limits. In such cases, use a multi-broker cluster to distribute connections among the broker instances within the cluster.

Too few file descriptors for the number of connections required on the Solaris or Linux operating system

For more information about this issue, see "Setting the File Descriptor Limit" on page 66.

To confirm this cause of the problem

Check for an entry in the broker log similar to the following: `Too many open files`.

To resolve the problem

Increase the file descriptor limit, as described in the `ulimit` man page.

TCP backlog limits the number of simultaneous new connection requests that can be established

The TCP backlog places a limit on the number of simultaneous connection requests that can be stored in the system backlog (`imq.portmapper.backlog`) before the Port Mapper rejects additional requests. (On Windows operating systems there is a hard-coded backlog limit: 5 for Windows desktops and 200 for Windows servers.)

The rejection of requests because of backlog limits is usually a transient phenomenon, due to an unusually high number of simultaneous connection requests.

To confirm this cause of the problem

Examine the broker log. First, check to see whether the broker is accepting some connections during the same time period that it is rejecting other connections. Next, check for messages that explain rejected connections. If you find such messages, the TCP backlog is probably not the problem, because the broker does not log connection rejections due to the TCP backlog.

If some successful connections are logged, and no connection rejections are logged, the TCP backlog is probably the problem.

To resolve the problem

The following approaches can be used to resolve TCP backlog limitations:

- Program the client to retry the attempted connection after a short interval of time (this normally works because of the transient nature of this problem).
- Increase the value of `imq.portmapper.backlog`.
- Check that clients are not closing and then opening connections too often.

Operating system limits the number of concurrent connections

The Windows operating system license places limits on the number of concurrent remote connections that are supported.

To confirm this cause of the problem

Check that there are plenty of threads available for connections (using `imqcmd query svc`) and check the terms of your Windows license agreement. If you can make connections from a local client, but not from a remote client, operating system limitations might be the cause of the problem.

To resolve the problem

- Upgrade the Windows license to allow more connections.
- Distribute connections among a number of broker instances by setting up a multi-broker cluster.

Authentication or authorization of the user is failing

The authentication can be failing due to an incorrect password, because there is no entry for the user in the user repository, or because the user does not have access permissions for the connection service.

To confirm this cause of the problem

Check entries in the broker log for the `Forbidden` error message. This will indicate an authentication error, but will not indicate the reason for it.

- If you are using a file-based user repository, enter the following command:

  `imqusermgr list -i` *instanceName* `-u` *userName*

- If the output shows a user, the wrong password was probably submitted. If the output shows the following error, there is no entry in the user repository:

  ```
  Error [B3048]: User does not exist in the password file,
  ```

- If you are using an LDAP server user repository, use the appropriate tools to check if there is an entry for the user.
- Check the access control properties file to see if there are restrictions on access to the connection service.

To resolve the problem

- If there is no entry for the user in the user repository, add the user to the user repository (see "Populating and Managing a User Repository" on page 137).
- If the wrong password was used, provide the correct password.
- If the access control properties are improperly set, edit the access control properties file to grant connection service permissions (see "Access Control for Connection Services" on page 145).

Connection Throughput Is Too Slow

The symptoms of this problem are as follows:

- Message throughput does not meet expectations.
- The number of supported connections to a broker is not limited as described in "A Client Cannot Establish a Connection" on page 234, but rather by message input/output rates.

This section explores the following possible causes:

- Network connection or WAN is too slow.
- Connection service protocol is inherently slow compared to TCP.
- Connection service protocol is not optimally tuned.
- Messages are so large they consume too much bandwidth.
- What appears to be slow connection throughput is actually a bottleneck in some other step of the message delivery process.

Network connection or WAN is too slow

To confirm this cause of the problem

Ping the network to see how long it takes for the ping to return, and then consult a network administrator. Also you can send and receive messages using local clients and compare the delivery time with that of remote clients (which use a network link).

To resolve the problem

If the connection is too slow, upgrade the network link.

Connection service protocol is inherently slow compared to TCP

As an example, SSL-based or HTTP-based protocols are slower than TCP (see Figure 11-5 on page 220).

To confirm this cause of the problem

If you are using SSL-based or HTTP-based protocols, try using TCP and compare the delivery times.

To resolve the problem

Application requirements usually dictate the protocols being used, so there is little that you can do, other than to attempt to tune the protocol as described in ("Tuning Transport Protocols" on page 224).

Connection service protocol is not optimally tuned

To confirm this cause of the problem

Try tuning the protocol and see if it makes a difference.

To resolve the problem

Try tuning the protocol as described in ("Tuning Transport Protocols" on page 224).

Messages are so large they consume too much bandwidth

To confirm this cause of the problem

Try running your benchmark with smaller-sized messages.

To resolve the problem

- Have application developers modify the application to use the message compression feature, which is described in the *Message Queue Developer's Guide for Java Clients*.
- Use messages as notifications of data to be sent, but move the data using another protocol.

What appears to be slow connection throughput is actually a bottleneck in some other step of the message delivery process

To confirm this cause of the problem

If none of the items above appear to be the cause of what appears to be slow connection throughput, consult Figure 11-1 on page 209 for other possible bottlenecks and check for symptoms associated with the following problems:

- "Message Production Is Delayed or Slowed" on page 242
- "Messages Are Backlogged" on page 245
- "Message Server Throughput Is Sporadic" on page 250

To resolve the problem

Follow the problem resolution guidelines provided in the problem troubleshooting sections above.

A Client Cannot Create a Message Producer

The symptoms of this problem are as follows:

- A message producer cannot be created for a physical destination; the client receives an exception.

This section explores the following possible causes:

- A physical destination has been configured to allow only a limited number of producers.
- The user is not authorized to create a message producer due to settings in the access control properties file.

A physical destination has been configured to allow only a limited number of producers

One of the ways of avoiding the accumulation of messages on a physical destination is to limit the number of producers (`maxNumProducers`) that it supports.

To confirm this cause of the problem

Check the physical destination (see "Displaying Information about Physical Destinations" on page 121):

```
imqcmd query dst
```

The output will show the current number of producers and the value of `maxNumProducers`. If the two values are the same, the number of producers has reached its configured limit. When a new producer is rejected by the broker, the broker returns a `ResourceAllocationException [C4088]: A JMS destination limit was reached` and makes the following entry in the broker log: `[B4183]: Producer can not be added to destination`.

To resolve the problem

Increase the value of the `maxNumProducers` attribute (see "Updating Physical Destination Properties" on page 123).

The user is not authorized to create a message producer due to settings in the access control properties file

To confirm this cause of the problem

When a new producer is rejected by the broker, the broker returns the following message:

```
JMSSecurityException [C4076]: Client does not have permission to
create producer on destination
```

The broker also makes the following entries in the broker log:

`[B2041]: Producer on destination denied` and `[B4051]: Forbidden guest`.

To resolve the problem

Change the access control properties to allow the user to produce messages (see "Access Control for Physical Destinations" on page 146).

Message Production Is Delayed or Slowed

The symptoms of this problem are as follows:

- When sending persistent messages, the `send` method does not return and the client blocks.
- When sending a persistent message, client receives an exception.
- Producing client slows down.

This section explores the following possible causes:

- The message server is backlogged and has responded by slowing message producers.
- The broker cannot save a persistent message to the data store.
- Broker acknowledgment timeout is too short.
- A producing client is encountering JVM limitations.

The message server is backlogged and has responded by slowing message producers

A backlogged server accumulates messages in broker memory.

When the number of messages or number of message bytes in physical destination memory reaches configured limits, the broker attempts to conserve memory resources in accordance with the specified limit behavior. The following limit behaviors slow down message producers:

- `FLOW_CONTROL`: The broker does not immediately acknowledge receipt of persistent messages (thereby blocking a producing client).
- `REJECT_NEWEST`: The broker rejects new persistent messages.

Similarly, when the number of messages or number of message bytes in brokerwide memory (for all physical destinations) reaches configured limits, the broker will attempt to conserve memory resources by rejecting the newest messages.

Also, when system memory limits are reached because physical destination or brokerwide limits have not been set properly, the broker takes increasingly serious action to prevent memory overload. These actions include throttling back message producers.

To confirm this cause of the problem

When a message is rejected by the broker due to configured message limits, the broker returns the following message:

```
JMSException [C4036]: A server error occurred
```

The broker also makes this entry in the broker log:

```
WARNING [B2011]: Storing of JMS message from IMQconn failed
```

The message is followed by a message indicating the limit that has been reached. If the message limit is on a physical destination, the broker makes an entry like the following:

```
[B4120]: Can not store message on destination destName because
capacity of maxNumMsgs would be exceeded.
```

If the message limit is broker wide, the broker makes an entry like the following:

```
[B4024]: The Maximum Number of messages currrently in the system has
been exceeded, rejecting message.
```

More generally, you can check for message limit conditions before the rejections occur as follows:

- By querying physical destinations and the broker and inspecting their configured message limit settings.
- By monitoring the number of messages or number of message bytes currently in a physical destination or in the broker as a whole, using the appropriate `imqcmd` commands. See Chapter 18, “Metrics Reference” for information about metrics you can monitor, and the commands you use to obtain them.

To resolve the problem

There are a number of approaches to addressing the slowing of producers due to messages becoming backlogged:

- Modify the message limits on a physical destination (or brokerwide) being careful not to exceed memory resources.

  In general, you should manage memory on a destination-by-destination level so that brokerwide message limits are never reached. For more information, see "Broker Adjustments" on page 228.

- Change the limit behaviors on a destination to not slow message production when message limits are reached, but rather to discard messages in memory.

  For example, you can specify the `REMOVE_OLDEST` and `REMOVE_LOW_PRIORITY` limit behaviors, which delete messages that accumulate in memory (see Table 15-1 on page 313).

The broker cannot save a persistent message to the data store

If the broker cannot access a data store or write a persistent message to the data store, the producing client is blocked. This condition can also occur if destination or brokerwide message limits are reached, as described above.

To confirm this cause of the problem

If the broker is unable to write to the data store, it makes one of the following entries in the broker log: `[B2011]: Storing of JMS message from connectionID failed`... or `[B4004]: Failed to persist message messageID`...

To resolve the problem

- In the case of file-based persistence, try increasing the disk space of the file-based data store.

- In the case of a JDBC-compliant data store, check that JDBC-based persistence is properly configured (see Chapter 4, "Configuring a Broker"). If so, consult your database administrator to troubleshoot other database problems.

Broker acknowledgment timeout is too short

Due to slow connections or a lethargic message server (caused by high CPU utilization or scarce memory resources), a broker might require more time to acknowledge receipt of a persistent message than allowed by the value of the connection factory's `imqAckTimeout` attribute.

To confirm this cause of the problem

If the `imqAckTimeout` value is exceeded, the broker returns the following message:

```
JMSException [C4000]: Packet acknowledge failed
```

To resolve the problem

Change the value of the `imqAckTimeout` connection factory attribute (see "Reliability And Flow Control" on page 170).

A producing client is encountering JVM limitations

To confirm this cause of the problem

- Find out whether the client application receives an Out Of Memory error.
- Check the free memory available in the JVM heap using runtime methods such as `freeMemory`, `MaxMemory`, and `totalMemory`.

To resolve the problem

Adjust the JVM (see "Java Virtual Machine Adjustments" on page 223).

Messages Are Backlogged

The symptoms of this problem are as follows:

- The number of messages or message bytes in the broker (or in specific destinations) increases steadily over time.

  To see whether messages are accumulating, check how the number of messages or message bytes in the broker changes over time and compare to configured limits. First check the configured limits:

  ```
  imqcmd query bkr
  ```

  (**Note:** the `imqcmd metrics bkr` subcommand does not display this information.)

  Then check for message accumulation in each destination:

  ```
  imqcmd list dst
  ```

  To see whether messages have exceeded configured destination or brokerwide limits, check the broker log for the following entry: `WARNING [B2011]: Storing of JMS message from…failed`. This entry will be followed by another entry explaining the limit that has been exceeded.

- Message production is delayed or produced messages are rejected by the broker.
- Messages take an unusually long time to reach consumers.

This section explores the following possible causes:

- There are inactive durable subscriptions on a topic destination.
- There are too few consumers available to consume messages in a queue.
- Message consumers are processing too slowly to keep up with message producers.
- Client acknowledgment processing is slowing down message consumption.
- The broker cannot keep up with produced messages.
- Client code defects: consumers are not acknowledging messages.

There are inactive durable subscriptions on a topic destination

If a durable subscription is inactive, messages are stored in a destination until the corresponding consumer becomes active and can consume the messages.

To confirm this cause of the problem

Check the state of durable subscriptions on each topic destination:

```
imqcmd list dur -d destName
```

To resolve the problem

You can take any of the following actions:

- Purge all messages for the offending durable subscriptions (see "Managing Durable Subscriptions" on page 112).
- Specify message limit and limit behavior attributes for the topic (see Table 15-1 on page 313). For example, you can specify the `REMOVE_OLDEST` and `REMOVE_LOW_PRIORITY` limit behaviors, which delete messages that accumulate in memory.
- Purge all messages from the corresponding destinations (see "Purging Physical Destinations" on page 124).
- Limit the time messages can remain in memory. You can rewrite the producing client to set a time-to-live value on each message. You can override any such settings for all producers sharing a connection by setting the `imqOverrideJMSExpiration` and `imqJMSExpiration` connection factory attributes (see "Message Header Overrides" on page 325).

There are too few consumers available to consume messages in a queue

If there are too few active consumers to which messages can be delivered, a queue destination can become backlogged as messages accumulate. This condition can occur for any of the following reasons:

- Too few active consumers exist for the destination.
- Consuming clients have failed to establish connections.
- No active consumers use a selector that matches messages in the queue.

To confirm this cause of the problem

To help determine the reason for unavailable consumers, check the number of active consumers on a destination:

```
imqcmd metrics dst -n destName -t q -m con
```

To resolve the problem

You can take any of the following actions, depending on the reason for unavailable consumers:

- Create more active consumers for the queue, by starting up additional consuming clients.
- Adjust the imq.consumerFlowLimit broker property to optimize queue delivery to multiple consumers (see "Multiple Consumer Queue Performance" on page 229).
- Specify message limit and limit behavior attributes for the queue (see Table 15-1 on page 313). For example, you can specify the `REMOVE_OLDEST` and `REMIOVE_LOW_PRIOROTY` limit behaviors, which delete messages that accumulate in memory.
- Purge all messages from the corresponding destinations (see "Purging Physical Destinations" on page 124).
- Limit the time messages can remain in memory. You can rewrite the producing client to set a time-to-live value on each message, you can override any such setting for all producers sharing a connection by setting the `imqOverrideJMSExpiration` and `imqJMSExpiration` connection factory attributes (see "Message Header Overrides" on page 325).

Message consumers are processing too slowly to keep up with message producers

In this case topic subscribers or queue receivers are consuming messages more slowly than the producers are sending messages. One or more destinations is getting backlogged with messages due to this imbalance.

To confirm this cause of the problem

Check for the rate of flow of messages into and out of the broker:

```
imqcmd metrics bkr -m rts
```

Then check flow rates for each of the individual destinations:

```
imqcmd metrics bkr -t destType -n destName -m rts
```

To resolve the problem

- Optimize consuming client code.
- For queue destinations, increase the number of active consumers (see "Multiple Consumer Queue Performance" on page 229).

Client acknowledgment processing is slowing down message consumption

Two factors affect the processing of client acknowledgments:

- Significant broker resources can be consumed in processing client acknowledgments. As a result, message consumption might be slowed in those acknowledgment modes in which consuming clients block until the broker confirms client acknowledgments.
- JMS payload messages and Message Queue control messages (such as client acknowledgments) share the same connection. As a result, control messages can be held up by JMS payload messages, slowing message consumption.

To confirm this cause of the problem

- Check the flow of messages relative to the flow of packets. If the number of packets per second is out of proportion to the number of messages, client acknowledgments might be a problem.
- Check to see whether the client has received the following message:

  ```
  JMSException [C4000]: Packet acknowledge failed
  ```

To resolve the problem

- Modify the acknowledgment mode used by clients, for example, switch to `DUPS_OK_ACKNOWLEDGE` or `CLIENT_ACKNOWLEDGE`.

- If using `CLIENT_ACKNOWLEDGE` or transacted sessions, group a larger number of messages into a single acknowledgment.
- Adjust consumer and connection flow control parameters (see "Client Runtime Message Flow Adjustments" on page 229).

The broker cannot keep up with produced messages

In this case, messages are flowing into the broker faster than the broker can route and dispatch them to consumers. The sluggishness of the broker can be due to limitations in any or all of the following: CPU, network socket read / write operations, disk read / write operations, memory paging, the persistent store, or JVM memory limits.

To confirm this cause of the problem

Check that none of the other causes of this problem are responsible.

To resolve the problem

- Upgrade the speed of your computer or your data store.
- Use a broker cluster to distribute the load among a number of broker instances.

Client code defects: consumers are not acknowledging messages

Messages are held in a destination until they have been acknowledged by all consumers to which the messages have been sent. If a client is not acknowledging consumed messages, the messages accumulate in the destination without being deleted.

For example, client code might have the following defects:

- Consumers using `CLIENT_ACKNOWLEDGEacknowledgment` or transacted session might not be calling `Session.acknowledge` or `Session.commit` on a regular basis.
- Consumers using `AUTO_ACKNOWLEDGE` sessions might be hanging for some reason.

To confirm this cause of the problem

First check all other possible causes listed in this section. Next, list the destination with the following command:

```
imqcmd list dst
```

Notice whether the number of messages listed under the UnAcked header is the same as the number of messages in the destination. The messages under the UnAcked header were sent to consumers but not acknowledged. If this number is the same as the total number of messages, the broker has sent all the messages and is waiting for acknowledgment.

To resolve the problem

Request the help of application developers in debugging this problem.

Message Server Throughput Is Sporadic

The symptom of this problem is as follows:

- Message throughput sporadically drops, and then resumes normal performance.

This section explores the following possible causes:

- The broker is very low on memory resources.
- JVM memory reclamation (garbage collection) is taking place.
- The JVM is using the Just-In-Time compiler to speed up performance.

The broker is very low on memory resources

Because destination and broker limits were not properly set, the broker takes increasingly serious action to prevent memory overload, and this can cause the broker to become very sluggish until the message backlog is cleared.

To confirm this cause of the problem

Check the broker log for a low memory condition (`[B1089]: In low memory condition, broker is attempting to free up resources`), followed by an entry describing the new memory state and the amount of total memory being used.

Also check the free memory available in the JVM heap:

```
imqcmd metrics bkr -m cxn
```

Free memory is low when the value of total JVM memory is close to the maximum JVM memory value.

To resolve the problem

- Adjust the JVM (see "Java Virtual Machine Adjustments" on page 223).
- Increase system swap space.

JVM memory reclamation (garbage collection) is taking place

Memory reclamation periodically sweeps through the system to free up memory. When this occurs, all threads are blocked. The larger the amount of memory to be freed up and the larger the JVM heap size, the larger the delay due to memory reclamation.

To confirm this cause of the problem

Monitor CPU usage on your computer. CPU usage drops when memory reclamation is taking place.

Also start your broker using the following command line options:

```
-vmargs -verbose:gc
```

Standard output indicates the time that memory reclamation takes place.

To resolve the problem

In multiple CPU computers, set the memory reclamation to take place in parallel:

```
-XX:+UseParallelGC=true
```

The JVM is using the Just-In-Time compiler to speed up performance

To confirm this cause of the problem

Check that none of the other causes of this problem are responsible.

To resolve the problem

Let the system run for a while; performance should improve.

Messages Are Not Reaching Consumers

The symptom of this problem is as follows:

- Messages sent by producers are not received by consumers.

This section explores the following possible causes:

- Limit behaviors are causing messages to be deleted on the broker.
- Message time-out value is expiring.
- Clocks are not synchronized.
- Consuming client failed to start message delivery on a connection.

Limit behaviors are causing messages to be deleted on the broker

When the number of messages or number of message bytes in destination memory reach configured limits, the broker attempts to conserve memory resources. Three of the configurable behaviors taken by the broker when these limits are reached will cause messages to be lost:

- `REMOVE_OLDEST`: deleting the oldest messages.
- `REMOVE_LOW_PRIORITY`: deleting the lowest priority messages according to age of the messages.
- `REJECT_NEWEST`: rejecting new persistent messages.

As the number of messages or number of message bytes in broker memory reach configured limits, the broker attempts to conserve memory resources by rejecting the newest messages.

To confirm this cause of the problem

Check the dead message queue, as described under "The Dead Message Queue Contains Messages" on page 255. Specifically, use the instructions under "The number of messages, or their sizes, exceed destination limits" on page 256. Look for the `REMOVE_OLDEST` or `REMOVE_LOW_PRIORITY` reason.

To resolve the problem

Increase the destination limits. For example:

```
imqcmd update dst -n MyDest -o maxNumMsgs=1000
```

Message time-out value is expiring

The broker deletes messages whose time-out value has expired. If a destination gets sufficiently backlogged with messages, messages whose time-to-live value is too short might be deleted.

To confirm this cause of the problem

Check the dead message queue to see whether messages are timing out.

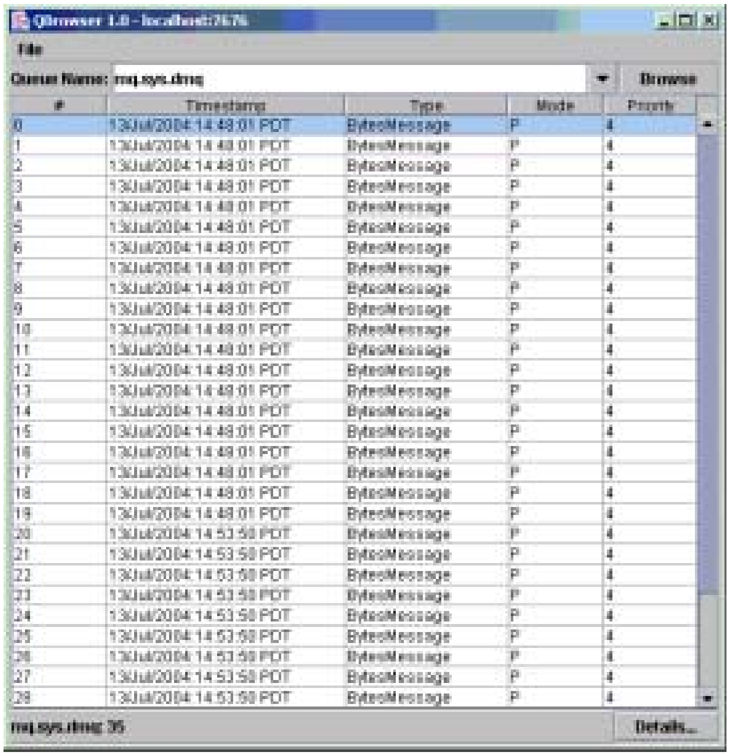
Use the QBrowser demo application to look at the DMQ contents. The QBrowser demo is in an operating system-specific location; for the location, see Appendix A, "Platform-Specific Locations of Message Queue Data" and look in the tables for "Example Applications and Locations."

This is an example of invocation on Windows:

```
cd \MessageQueue3\demo\applications\qbrowser java QBrowser
```

When the QBrowser main window appears, select the queue name `mq.sys.dmq` and then click Browse. A list like the following appears.

Figure 12-1 QBrowser Window



QBrowser 1.0 - localhost:7676

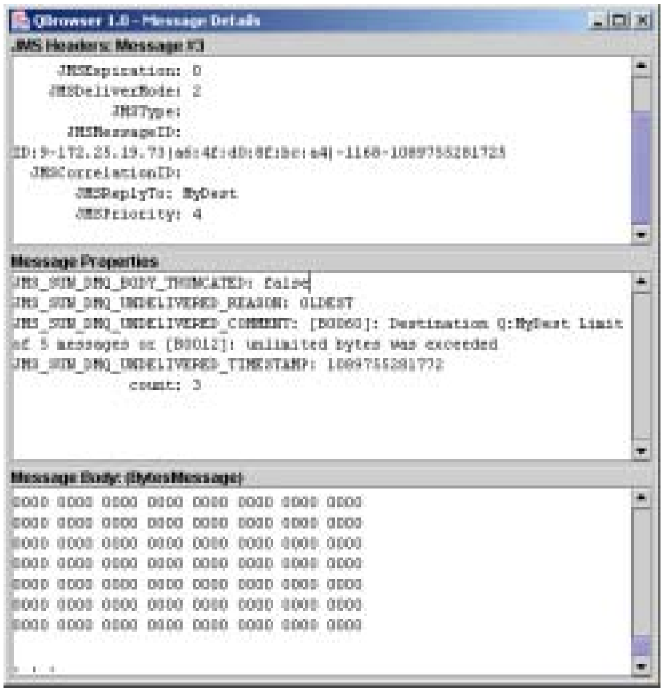
Queue Name: mq.sys.dmq

#	Timestamp	Type	Mode	Priority
0	13/Jul/2004 14:48:01 PDT	BytesMessage	P	4
1	13/Jul/2004 14:48:01 PDT	BytesMessage	P	4
2	13/Jul/2004 14:48:01 PDT	BytesMessage	P	4
3	13/Jul/2004 14:48:01 PDT	BytesMessage	P	4
4	13/Jul/2004 14:48:01 PDT	BytesMessage	P	4
5	13/Jul/2004 14:48:01 PDT	BytesMessage	P	4
6	13/Jul/2004 14:48:01 PDT	BytesMessage	P	4
7	13/Jul/2004 14:48:01 PDT	BytesMessage	P	4
8	13/Jul/2004 14:48:01 PDT	BytesMessage	P	4
9	13/Jul/2004 14:48:01 PDT	BytesMessage	P	4
10	13/Jul/2004 14:48:01 PDT	BytesMessage	P	4
11	13/Jul/2004 14:48:01 PDT	BytesMessage	P	4
12	13/Jul/2004 14:48:01 PDT	BytesMessage	P	4
13	13/Jul/2004 14:48:01 PDT	BytesMessage	P	4
14	13/Jul/2004 14:48:01 PDT	BytesMessage	P	4
15	13/Jul/2004 14:48:01 PDT	BytesMessage	P	4
16	13/Jul/2004 14:48:01 PDT	BytesMessage	P	4
17	13/Jul/2004 14:48:01 PDT	BytesMessage	P	4
18	13/Jul/2004 14:48:01 PDT	BytesMessage	P	4
19	13/Jul/2004 14:48:01 PDT	BytesMessage	P	4
20	13/Jul/2004 14:53:50 PDT	BytesMessage	P	4
21	13/Jul/2004 14:53:50 PDT	BytesMessage	P	4
22	13/Jul/2004 14:53:50 PDT	BytesMessage	P	4
23	13/Jul/2004 14:53:50 PDT	BytesMessage	P	4
24	13/Jul/2004 14:53:50 PDT	BytesMessage	P	4
25	13/Jul/2004 14:53:50 PDT	BytesMessage	P	4
26	13/Jul/2004 14:53:50 PDT	BytesMessage	P	4
27	13/Jul/2004 14:53:50 PDT	BytesMessage	P	4
28	13/Jul/2004 14:53:50 PDT	BytesMessage	P	4

mq.sys.dmq: 35

Double click a message to display details about that message.

Figure 12-2 QBrowser Message Details



```
     JMSExpiration: 0
   JMSDeliverMode: 2
          JMSType:
     JMSMessageID:
ID:9-172.23.19.73|a6:4f:d0:8f:bc:a4|-1168-1089755281723
  JMSCorrelationID:
       JMSReplyTo: MyDest
      JMSPriority: 4
```

```
JMS_SUN_DMQ_BODY_TRUNCATED: false
JMS_SUN_DMQ_UNDELIVERED_REASON: OLDEST
JMS_SUN_DMQ_UNDELIVERED_COMMENT: [B0060]: Destination Q:MyDest Limit
of 5 messages or [B0012]: unlimited bytes was exceeded
JMS_SUN_DMQ_UNDELIVERED_TIMESTAMP: 1089755281772
              count: 3
```

```
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000

. . .
```

Note whether the `JMS_SUN_DMQ_UNDELIVERED_REASON` property for messages has the value `EXPIRED`.

To resolve the problem

Contact the application developers and have them increase the time-to-live value.

Clocks are not synchronized

If clocks are not synchronized, broker calculations of message lifetimes can be wrong, causing messages to exceed their expiration times and be deleted.

To confirm this cause of the problem

In the broker log file, look for any of the following messages: B2102, B2103, B2104. These messages all report that possible clock skew was detected.

To resolve this problem

Check that you are running a time synchronization program, as described in "Preparing System Resources" on page 65.

Consuming client failed to start message delivery on a connection

Messages cannot be delivered until client code establishes a connection and starts message delivery on the connection.

To confirm this cause of the problem

Check that client code establishes a connection and starts message delivery.

To resolve the problem

Rewrite the client code to establish a connection and start message delivery.

The Dead Message Queue Contains Messages

The symptom of this problem is as follows:

- When you list destinations, you see that the dead message queue contains messages. For example, issue a command like the following.

  ```
  imqcmd list dst
  ```

  After you supply a user name and password, output like the following appears:

  ```
  Listing all the destinations on the broker specified by:
  --------------------------------
  Host         Primary Port
  --------------------------------
  localhost    7676
  --------------------------------------------------------------------
     Name     Type    State   Producers  Consumers Msgs
                                           Total Count  UnAck  Avg Size
  --------------------------------------------------------------------
  MyDest      Queue  RUNNING  0          0          5      0      1177.0
  mq.sys.dmq  Queue  RUNNING   0          0          35     0      1422.0
  Successfully listed destinations.
  ```

  In this example, the dead message queue, `mq.sys.dmq`, contains 35 messages.

This section explores the following possible causes:

- The number of messages, or their sizes, exceed destination limits.
- The broker clock and producer clock are not synchronized.
- Consumers are not receiving the messages before messages time out.

- There are too many producers for the number of consumers.
- Producers are faster than consumers.
- A consumer is too slow.
- Clients are not committing messages.
- Durable consumers are inactive.
- An unexpected broker error occurred.

The number of messages, or their sizes, exceed destination limits

To confirm this cause of the problem

Use the QBrowser demo application to look at the contents of the dead message queue. The QBrowser demo is in an operating system-specific location; for the location, see Appendix A, "Platform-Specific Locations of Message Queue Data" and look in the tables for "Example Applications and Locations."

This is an example of invocation on Windows:

```
cd \MessageQueue3\demo\applications\qbrowser java QBrowser
```

When the QBrowser main window appears, select the queue name `mq.sys.dmq` and then click Browse. A list like the one shown in Figure 12-1 on page 253 appears.

Double click any message to display details about that message. The window shown in Figure 12-2 on page 254 appears.

Note the values for the following message properties:

- `JMS_SUN_DMQ_UNDELIVERED_REASON`
- `JMS_SUN_DMQ_UNDELIVERED_COMMENT`
- `JMS_SUN_DMQ_UNDELIVERED_TIMESTAMP`

Under JMS Headers, note the value for `JMSDestination` to determine the destination whose messages are becoming dead.

To resolve this problem

Increase the destination limits. For example:

```
imqcmd update dst -n MyDest -o maxNumMsgs=1000
```

The broker clock and producer clock are not synchronized

To confirm this cause of the problem:

Using the QBrowser application, view the message details for messages in the dead message queue. Check the value for `JMS_SUN_DMQ_UNDELIVERED_REASON`, looking for messages with the reason `EXPIRED`.

In the broker log file, look for any of the following messages: B2102, B2103, B2104. These messages all report that possible clock skew was detected.

To resolve this problem

Check that you are running a time synchronization program, as described in "Preparing System Resources" on page 65.

Consumers are not receiving the messages before messages time out

To verify this cause of the problem

Using the QBrowser application, view the message details for messages in the dead message queue. Check the value for `JMS_SUN_DMQ_UNDELIVERED_REASON, looking for` messages with the reason `EXPIRED`.

Check to see whether there any consumers on the destination. For example:

```
imqcmd query dst -t q -n MyDest
```

Check the value listed for Current Number of Active Consumers. If there are active consumers, one of the following is true:

- A consumer's connection is paused.
- The message timeout is too short for the speed at which the consumer executes.

To resolve the problem

Request that application developers increase message time-to-live values.

There are too many producers for the number of consumers

To confirm this cause of the problem

Using the QBrowser application, view the message details for messages in the dead message queue. Check the value for `JMS_SUN_DMQ_UNDELIVERED_REASON`.

If the reason is `REMOVE_OLDEST` or `REMOVE_LOW_PRIORITY`, use the `imqcmd query dst` command to check the number of producers and consumers on the destination. If the number of producers exceeds the number of consumers, production rate might be overwhelming consumption rate.

To resolve the problem

Add more consumer clients or set the destination to use the `FLOW_CONTROL` limit behavior. The `FLOW_CONTROL` limit behavior uses consumption rate to control production rate.

Start the flow control behavior by using a command such as the following example:

```
imqcmd update dst -n myDst -t q -o consumerFlowLimit=FLOW_CONTROL
```

Producers are faster than consumers

To confirm this cause of the problem

To determine whether slow consumers are causing producers to slow down, set the destination limit behavior to `FLOW_CONTROL`. The `FLOW_CONTROL` limit behavior uses consumption rate to control production rate.

Start the flow control behavior by using a command such as the following example:

```
imqcmd update dst -n myDst -t q -o consumerFlowLimit=FLOW_CONTROL
```

Use metrics to examine the destination input and output, by issuing a command like the following example:

```
imqcmd metrics dst -n myDst -t q -m rts
```

In the metrics output, examine the following values:

- `Msgs/sec Out`

  This value shows how many messages per second the broker is removing. The broker removes messages when all consumers acknowledge receiving them, so the metric reflects consumption rate.

- `Msgs/sec In`

  This value shows how many messages per second the broker is receiving from producers. The metric reflects production rate.

Because flow control aligns production to consumption, note whether production slows or stops. If the rate slows or stops, there is a discrepancy between the processing speed of producers and consumers.

You can also check the number of unacknowledged (UnAcked) sent messages, by using the `imqcmd list dst` command. If the number of unacknowledged messages is less than the size of the destination. the destination has additional capacity and is being held back by client flow control.

To resolve the problem

If production rate is consistently faster than consumption rate, consider using flow control regularly, to keep the system aligned.

In addition, using the subsequent sections, consider and attempt to resolve each of the following possible factors:

- A consumer is too slow.
- Clients are not committing messages.
- Consumers are failing to acknowledge messages.
- Durable consumers are inactive.
- An unexpected broker error occurred.

A consumer is too slow

To confirm this cause of the problem

Use metrics to determine the rate of production and consumption, as described under "Producers are faster than consumers" on page 258.

To resolve the problem

Try one or more of the following:

- Set the destinations to use the `FLOW_CONTROL` limit behavior. Use a command like the following:

  ```
  imqcmd update dst -n myDst -t q -o consumerFlowLimit=FLOW_CONTROL
  ```

  Use of flow control slows production to the rate of consumption and prevents the accumulation of messages on the broker. Producer applications hold messages until the destination can process them in a timely manner, with less risk of expiration.

- Find out from application developers whether producers send messages at a steady rate, or in periodic bursts.

  If an application sends bursts of messages, follow the instructions in the next item to increase destination limits.

- Increase destination limits based on number of messages or number of bytes, or both.

  To change the number of messages on a destination, enter a command that has the following format:

  ```
  imqcmd update dst -n destName -t {q/t} -o maxNumMsgs=number
  ```

To change the size of a destination, enter a command that has the following format:

```
imqcmd update dst -n destName -t {q/t} -o maxTotalMsgBytes=number
```

Be aware that raising limits increases the amount of memory that the broker uses. If limits are too high, the broker could run out of memory and become unable to process messages.

- Consider whether you can accept loss of messages during levels of high production load.

Clients are not committing messages

To confirm this cause of the problem

Check with application developers to find out whether the application uses transactions. If the application uses transactions, list the active transactions as follows:

```
imqcmd list txn
```

This is an example of the command output:

```
----------------------------------------------------------------------
Transaction ID      State    User name  # Msgs/# Acks   Creation time
----------------------------------------------------------------------
6800151593984248832  STARTED  guest       3/2        7/19/04 11:03:08 AM
```

Note the numbers of messages and number of acknowledgments.

If the number of messages is high, producers may be sending individual messages but failing to commit transactions. Until the broker receives a commit, it cannot route and deliver the messages for that transaction.

If the number of acknowledgments is high, consumers may be sending acknowledgments for individual messages but failing to commit transactions. Until the broker receives a commit, it cannot remove the acknowledgments for that transaction.

To resolve this problem

Contact application developers to fix the coding error.

Consumers are failing to acknowledge messages

To confirm this cause of the problem

Contact application developers to determine whether the application uses system-based acknowledgment or client-based acknowledgment. If the application uses system-based acknowledgment, skip this section.

If the application uses client-based acknowledgment (the `CLIENT_ACKNOWLEDGE type`), first decrease the number of messages stored on the client. Use a command like the following:

```
imqcmd update dst -n myDst -t q -o consumerFlowLimit=1
```

Next, you will determine whether the broker is buffering messages because a consumer is slow, or whether the consumer processes messages quickly but does not acknowledge them.

List the destination, using the following command:

```
imqcmd list dst
```

After you supply a user name and password, output like the following appears:

```
Listing all the destinations on the broker specified by:
---------------------------------
Host         Primary Port
---------------------------------
localhost    7676
----------------------------------------------------------------------
   Name     Type    State   Producers  Consumers Msgs
                                         Total Count  UnAck  Avg Size
-----------------------------------------------------------------------
MyDest      Queue   RUNNING   0          0       5    200    1177.0
mq.sys.dmq  Queue   RUNNING   0          0      35      0    1422.0
Successfully listed destinations.
```

The `UnAck` number represents messages that the broker has sent and for which it is waiting for acknowledgment. If the `UnAck` number is high or increasing, you know that the broker is sending messages, so it is not waiting for a slow consumer. You also know that the consumer is not acknowledging the messages.

To resolve the problem

Contact application developers to fix the coding error.

Durable consumers are inactive

To confirm this cause of the problem

Look at the topic's durable subscribers, using the following command format:

```
imqcmd list dur -d topicName
```

To resolve the problem

- Purge the durable consumers using the `imqcmd purge dur` command.
- Restart the consumer applications.

An unexpected broker error occurred

To confirm this cause of the problem

Use QBrowser to examine a message, as described under "Producers are faster than consumers" on page 258.

If the value for `JMS_SUN_DMQ_UNDELIVERED_REASON` is `ERROR`, a broker error occurred.

To resolve the problem

- Examine the broker log file to find the associated error.
- Contact Sun Technical Support to report the broker problem.

Part III

Reference

Chapter 13

Command Line Reference

This chapter provides reference information on the use of the Message Queue command line administration utilities. It consists of the following sections:

Command Line Syntax

Message Queue command line utilities are shell commands. The name of the utility is a command and its subcommands or options are arguments passed to that command. There is no need for separate commands to start or quit the utility.

All the command line utilities share the following command syntax:

utilityName [*subcommand*] [*commandArgument*] [[-*optionName* [-*optionArgument*]]...]

where *utilityName* is one of the following:

- `imqbrokerd` (Broker utility)
- `imqcmd` (Command utility)

- `imqobjmgr` (Object Manager utility)
- `imqdbmgr` (Database Manager utility)
- `imqusermgr` (User Manager utility)
- `imqsvcadmin` (Service Administrator utility)
- `imqkeytool` (Key Tool utility)

Subcommands and command-level arguments, if any, must precede all options and their arguments; the options themselves may appear in any order. All subcommands, command arguments, options, and option arguments are separated with spaces. If the value of an option argument contains a space, the entire value must be enclosed in quotation marks. (It is generally safest to enclose any attribute-value pair in quotation marks.)

The following command, which starts the default broker, is an example of a command line with no subcommand clause:

```
imqbrokerd
```

Here is a fuller example:

```
imqcmd destroy dst -t q -n myQueue -u admin -f -s
```

This command destroys a queue destination (destination type `q`) named `myQueue`. Authentication is performed on the user name `admin`; the command will prompt for a password. The command will be performed without prompting for confirmation (`-f` option) and in silent mode, without displaying any output (`-s` option).

Broker Utility

The Broker utility (`imqbrokerd`) starts a broker. Command line options override values in the broker configuration files, but only for the current broker session.

Table 13-1 shows the options to the `imqbrokerd` command and the configuration properties, if any, overridden by each option.

Table 13-1 Broker Utility Options

Option	Properties Overridden	Description
`-name` *instanceName*	`imq.instancename`	Instance name of broker Multiple broker instances running on the same host must have different instance names. Default value: `imqbroker`
`-port` *portNumber*	`imq.portmapper.port`	Port number for broker's Port Mapper Message Queue clients use this port number to connect to the broker. Multiple broker instances running on the same host must have different Port Mapper port numbers. Default value: `7676`
`-cluster` *broker1* `[[`*, broker2*`]…]`	`imq.cluster.brokerlist`	Connect brokers into cluster[1] The specified brokers are merged with the list in the `imq.cluster.brokerlist` property. Each broker argument has one of the forms *hostName*`:`*portNumber* *hostName* `:`*portNumber* If *hostName* is omitted, the default value is `localhost`; if *portNumber* is omitted, the default value is `7676`.
`-D`*property*`=`*value*	Corresponding property in instance configuration file	Set configuration property See Chapter 14, "Broker Properties Reference," for information about broker configuration properties. **Caution:** Be careful to check the spelling and formatting of properties set with this option. Incorrect values will be ignored without notification or warning.
`-reset props`	None	Reset configuration properties Replaces the broker's existing instance configuration file (`config.properties`) with an empty file; all properties assume their default values.

Table 13-1 Broker Utility Options *(Continued)*

Option	Properties Overridden	Description
`-reset store`	None	Reset persistent data store Clears all persistent data from the data store (including persistent messages, durable subscriptions, and transaction information), allowing you to start the broker instance with a clean slate. To prevent the persistent store from being reset on subsequent restarts, restart the broker instance without the `-reset` option. To clear only persistent messages or durable subscriptions, use `-reset messages` or `-reset durables` instead.
`-reset messages`	None	Clear persistent messages from data store
`-reset durables`	None	Clear durable subscriptions from data store
`-backup` *fileName*	None	Back up configuration change record to file[1] See "Managing the Configuration Change Record" on page 187 for more information.
`-restore` *fileName*	None	Restore configuration change record from backup file[1] The backup file must have been previously created using the `-backup` option. See "Managing the Configuration Change Record" on page 187 for more information.
`-remove instance`	None	Remove broker instance[2] Deletes the instance configuration file, log files, persistent store, and other files and directories associated with the instance.
`-password` *keyPassword*	`imq.keystore.password`	Password for SSL certificate key store[3]
`-dbuser` *userName*	`imq.persist.jdbc.user`	User name for JDBC-based persistent data store
`-dbpassword` *dbPassword*	`imq.persist.jdbc.password`	Password for JDBC-based persistent data store[3]
`-ldappassword` *ldapPassword*	`imq.user_repository.ldap.password`	Password for LDAP user repository[3]

Table 13-1 Broker Utility Options *(Continued)*

Option	Properties Overridden	Description

`-passfile` *filePath*	`imq.passfile.enabled`
	`imq.passfile.dirpath`
	`imq.passfile.name`
		Location of password file
		
		Sets the broker's `imq.passfile.enabled` property to `true`, `imq.passfile.dirpath` to the path containing the password file, and `imq.passfile.name` to the file name itself.
		
		See "Using a Password File" on page 158 for more information.

`-shared`	`imq.jms.threadpool_model`	Use shared thread pool model to implement `jms` connection service Execution threads will be shared among connections to increase the number of connections supported. Sets the broker's `imq.jms.threadpool_model` property to `shared`.
`-javahome` *path*	None	Location of alternative Java runtime Default: Use runtime installed on system or bundled with Message Queue.
`-vmargs` *arg1* [[*arg2*] …]	None	Pass arguments to Java virtual machine Arguments are separated with spaces. To pass more than one argument, or an argument containing a space, enclose the argument list in quotation marks. VM arguments can be passed only from the command line; there is no associated configuration property in the instance configuration file.
`-license` [*licenseName*]	None	License to load, if different from default for installed edition of Message Queue product: `pe` Platform Edition with basic features `try` Platform Edition with enterprise features (90-day trial) `unl` Enterprise Edition If no license name is specified, this option lists all licenses installed on the system.

Table 13-1 Broker Utility Options *(Continued)*

Option	Properties Overridden	Description
`-upgrade-store-nobackup`	None	Automatically remove old data store on upgrade to Message Queue 3.5 or 3.5 SP*x* from an incompatible version[2] See the *Message Queue Installation Guide* for more information.
`-force`	None	Perform action without user confirmation This option applies only to the `-remove instance` and `-upgrade-store-nobackup` options, which normally require confirmation.

`-loglevel` *level*	`imq.broker.log.level`	Logging level:
		`NONE`
		`ERROR`
		`WARNING`
		`INFO`
		Default value: `INFO`

`-metrics` *interval*	`imq.metrics.interval`	Logging interval for broker metrics, in seconds
`-tty`	`imq.log.console.output`	Log all messages to console Sets the broker's `imq.log.console.output` property to `ALL`. If not specified, only error and warning messages will be logged.
`-s | -silent`	`imq.log.console.output`	Silent mode (no logging to console) Sets the broker's `imq.log.console.output` property to `NONE`.
`-version`	None	Display version information[4]
`-h | -help`	None	Display usage help[4]

1. This option applies only to broker clusters.
2. This option requires user confirmation unless `-force` is also specified.
3. This option is being deprecated and will eventually be removed. Instead, either omit the password entirely (so that the command will prompt for it interactively) or use the `-passfile` option to specify a password file containing the password.
4. Any other options specified on the command line are ignored .

Command Utility

The Command utility (imqcmd) is used for managing brokers, connection services, connections, physical destinations, durable subscriptions, and transactions.

All imqcmd commands must include a subcommand (except those using the -v or -h option to display product version information or usage help). The possible subcommands are listed here and described in detail in the corresponding sections below. In all cases, if the subcommand accepts a broker address (-b option) and no host name or port number is specified, the values localhost and 7676 are assumed by default.

Broker Management	
`shutdown bkr`	Shut down broker
`restart bkr`	Restart broker
`pause bkr`	Pause broker
`resume bkr`	Resume broker
`update bkr`	Set broker properties
`reload cls`	Reload cluster configuration
`query bkr`	List broker property values
`metrics bkr`	Display broker metrics
Connection Service Management	
`pause svc`	Pause connection service
`resume svc`	Resume connection service
`update svc`	Set connection service properties
`list svc`	List connection services available on broker
`query svc`	List connection service property values
`metrics svc`	Display connection service metrics
Connection Management	
`list cxn`	List connections on broker
`query cxn`	Display connection information

Physical Destination Management	
`create dst`	Create physical destination
`destroy dst`	Destroy physical destination
`pause dst`	Pause message delivery for physical destination
`resume dst`	Resume message delivery for physical destination
`update dst`	Set physical destination properties
`purge dst`	Purge all messages from physical destination
`compact dst`	Compact physical destination
`list dst`	List physical destinations
`query dst`	List physical destination property values
`metrics dst`	Display physical destination metrics
Durable Subscription Management	
`destroy dur`	Destroy durable subscription
`purge dur`	Purge all messages for durable subscription
`list dur`	List durable subscriptions for topic
Transaction Management	
`commit txn`	Commit transaction
`rollback txn`	Roll back transaction
`list txn`	List transactions being tracked by broker
`query txn`	Display transaction information

Broker Management

The Command utility cannot be used to start a broker; use the Broker utility (`imqbrokerd`) instead. Once the broker is started, you can use the `imqcmd` subcommands listed in Table 13-2 to manage and control it.

Table 13-2 Command Utility Subcommands for Broker Management

Syntax	Description
`shutdown bkr [-b` *hostName*:*portNumber*`]`	Shut down broker
`restart bkr [-b` *hostName*:*portNumber*`]`	Restart broker Shuts down the broker and then restarts it using the same options specified when it was originally started.

Table 13-2 Command Utility Subcommands for Broker Management *(Continued)*

Syntax	Description
`pause bkr [-b` *hostName*`:`*portNumber*`]`	Pause broker See "Pausing a Broker" on page 103 for more information.
`resume bkr [-b` *hostName*`:`*portNumber*`]`	Resume broker

`update bkr [-b` *hostName*`:`*portNumber*`]`
`-o` *property1*=*value1*
`[[-o` *property2*=*value2*`] … ]`
	Set broker properties
	See Chapter 14, "Broker Properties Reference," for information on broker properties.

`reload cls`	Reload cluster configuration[1] Forces all persistent information to be brought up to date.
`query bkr -b` *hostName*`:`*portNumber*	List broker property values Also lists all running brokers connected to the specified broker in a cluster.

`metrics bkr [-b` *hostName*`:`*portNumber*`]`
`[-m` *metricType*`]`
`[-int` *interval*`]`
`[-msp` *numSamples*`]`
	Display broker metrics
	The `-m` option specifies the type of metrics to display:
	`ttl` Messages and packets flowing into and out of the broker
	`rts` Rate of flow of messages and packets into and out of the broker per second
	`cxn` Connections, virtual memory heap, and threads
	Default value: `ttl`.
	The `-int` option specifies the interval, in seconds, at which to display metrics. Default value: 5.
	The `-msp` option specifies the number of samples to display. Default value: unlimited (infinite).

1. This option applies only to broker clusters.

Connection Service Management

Table 13-3 lists the `imqcmd` subcommands for managing connection services.

Table 13-3 Command Utility Subcommands for Connection Service Management

Syntax	Description
`pause svc -n` *serviceName* `[-b` *hostName*`:`*portNumber*`]`	Pause connection service The `admin` connection service cannot be paused.

Table 13-3 Command Utility Subcommands for Connection Service Management

Syntax	Description
`resume svc -n` *serviceName* `[-b` *hostName*`:`*portNumber*`]`	Resume connection service

`update svc -n` *serviceName*
`[-b` *hostName*`:`*portNumber*`]`
`-o` *property1*=*value1*
`[[-o` *property2*=*value2*`] … ]`
	Set connection service properties
	See "Connection Properties" on page 285 for information on connection service properties.

`list svc [-b` *hostName*`:`*portNumber*`]`	List connection services available on broker
`query svc -n` *serviceName* `[-b` *hostName*`:`*portNumber*`]`	List connection service property values

`metrics svc -n` *serviceName*
`[-b` *hostName*`:`*portNumber*`]`
`[-m` *metricType*`]`
`[-int` *interval*`]`
`[-msp` *numSamples*`]`
	Display connection service metrics
	The `-m` option specifies the type of metrics to display:
	`ttl` Messages and packets flowing into and out of the broker by way of the specified connection service
	`rts` Rate of flow of messages and packets into and out of the broker per second by way of the specified connection service
	`cxn` Connections, virtual memory heap, and threads
	Default value: `ttl`.
	The `-int` option specifies the interval, in seconds, at which to display metrics. Default value: `5`.
	The `-msp` option specifies the number of samples to display. Default value: unlimited (infinite).

Connection Management

Table 13-4 lists the `imqcmd` subcommands for managing connections.

Table 13-4 Command Utility Subcommands for Connection Service Management

Syntax	Description

`list cxn [-svn` *serviceName*`]`
`[-b` *hostName*`:`*portNumber*`]`
	List connections on broker
	Lists all connections on the broker to the specified connection service. If no connection service is specified, all connections are listed.

`query cxn -n` *connectionID* `[-b` *hostName*`:`*portNumber*`]`	Display connection information

Physical Destination Management

Table 13-5 lists the `imqcmd` subcommands for managing physical destinations. In all cases, the `-t` (destination type) option can take either of two values:

`q`	Queue destination
`t`	Topic destination

Table 13-5 Command Utility Subcommands for Physical Destination Management

Syntax	Description

`create dst -t` *destType* `-n` *destName*
 `[-o` *property1=value1*`]`
 `[[-o` *property2=value2*`] … ]`
	Create physical destination[1]
	
	 The destination name *destName* may contain only alphanumeric characters (no spaces) and must begin with an alphabetic character or the underscore (_) or dollar sign (**$**) character. It may not begin with the characters `mq`.

`destroy dst -t` *destType* `-n` *destName*	Destroy physical destination[1] This operation cannot be applied to a system-created destination, such as a dead message queue.

`pause dst [-t` *destType* `-n` *destName*`]`
 `[-pst` *pauseType*`]`
	Pause message delivery for physical destination
	
	 Pauses message delivery for the physical destination specified by the `-t` and `-n` options. If these options are not specified, all destinations are paused.
	
	 The `pst` option specifies the type of message delivery to be paused:
	
	 `CONSUMERS` Pause delivery to message consumers
	 `PRODUCERS` Pause delivery to message producers
	 `ALL` Pause all message delivery
	
	 Default value: `ALL`

`resume dst [-t` *destType* `-n` *destName*`]`	Resume message delivery for physical destination Resumes message delivery for the physical destination specified by the `-t` and `-n` options. If these options are not specified, all destinations are resumed.
`update dst -t` *destType* `-n` *destName* `-o` *property1=value1* `[[-o` *property2=value2*`] … ]`	Set physical destination properties See Chapter 15, "Physical Destination Property Reference," for information on physical destination properties.
`purge dst -t` *destType* `-n` *destName*	Purge all messages from physical destination

Table 13-5 Command Utility Subcommands for Physical Destination Management

Syntax	Description
`compact dst [-t` *destType* `-n` *destName*`]`	Compact physical destination Compacts the file-based persistent data store for the physical destination specified by the `-t` and `-n` options. If these options are not specified, all destinations are compacted. A destination must be paused before it can be compacted.

`list dst [-t` *destType*`]`
`[-tmp]`
	List physical destinations
	
	Lists all physical destinations of the type specified by the `-t` option. If no destination type is specified, both queue and topic destinations are listed. If the `-tmp` option is specified, temporary destinations are listed as well.

`query dst -t` *destType* `-n` *destName*	List physical destination property values

`metrics dst -t` *destType* `-n` *destName*
`[-m` *metricType*`]`
`[-int` *interval*`]`
`[-msp` *numSamples*`]`
	Display physical destination metrics
	
	The `-m` option specifies the type of metrics to display:
	`ttl` Messages and packets flowing into and out of the destination and residing in memory
	`rts` Rate of flow of messages and packets into and out of the broker per second, along with other rate information
	`con` Metrics related to message consumers
	`dsk` Disk usage
	
	Default value: `ttl`.
	
	The `-int` option specifies the interval, in seconds, at which to display metrics. Default value: `5`.
	
	The `-msp` option specifies the number of samples to display. Default value: unlimited (infinite).

1. This operation cannot be performed in a broker cluster whose master broker is temporarily unavailable.

Durable Subscription Management

Table 13-6 lists the `imqcmd` subcommands for managing durable subscriptions.

Table 13-6 Command Utility Subcommands for Durable Subscription Management

Syntax	Description
`destroy dur -c` *clientID* `-n` *subscriberName*	Destroy durable subscription[1]
`purge dur -c` *clientID* `-n` *subscriberName*	Purge all messages for durable subscription
`list dur -d` *topicName*	List durable subscriptions for topic

1. This operation cannot be performed in a broker cluster whose master broker is temporarily unavailable.

Transaction Management

Table 13-7 lists the `imqcmd` subcommands for managing transactions.

Table 13-7 Command Utility Subcommands for Transaction Management

Syntax	Description
`commit txn -n` *transactionID*	Commit transaction
`rollback txn -n` *transactionID*	Roll back transaction
`list txn`	List transactions being tracked by broker
`query txn -n` *transactionID*	Display transaction information

General Command Utility Options

The additional options listed in Table 13-8 can be applied to any subcommand of the `imqcmd` command.

Table 13-8 General Command Utility Options

Option	Description
`-secure`	Use secure connection to broker with `ssladmin` connection service

Table 13-8 General Command Utility Options *(Continued)*

Option	Description
`-u` *userName*	User name for authentication If this option is omitted, the Command utility will prompt for it interactively.
`-p` *password*	Password for authentication[1]
`-passfile` *path*	Location of password file See "Using a Password File" on page 158 for more information.
`-rtm` *timeoutInterval*	Initial timeout interval, in seconds This is the initial length of time that the Command utility will wait for a reply from the broker before retrying a request. Each subsequent retry will use a timeout interval that is a multiple of this initial interval. Default value: `10`.
`-rtr` *numRetries*	Number of retries to attempt after a broker request times out Default value: `5`.
`-javahome` *path*	Location of alternative Java runtime Default: Use runtime installed on system or bundled with Message Queue.
`-f`	Perform action without user confirmation
`-s`	Silent mode (no output displayed)
`-v`	Display version information[2,3]
`-h`	Display usage help[2,3]
`-H`	Display expanded usage help, including attribute list and examples[2,3]

1. This option is being deprecated and will eventually be removed. Instead, either omit the password entirely (so that the command will prompt for it interactively) or use the `-passfile` option to specify a password file containing the password.
2. Any other options specified on the command line are ignored .
3. A user name and password are not needed with this option.

Object Manager Utility

The Object Manager utility (`imqobjmgr`) creates and manages Message Queue administered objects. Table 13-9 lists the available subcommands.

Table 13-9 Object Manager Subcommands

Subcommand	Description
`add`	Add administered object to object store
`delete`	Delete administered object from object store
`list`	List administered objects in object store
`query`	Display administered object information
`update`	Modify administered object

Table 13-10 lists the options to the `imqobjmgr` command.

Table 13-10 Object Manager Options

Option	Description
`-l` *lookupName*	JNDI lookup name of administered object
`-j` *attribute*=*value*	Attributes of JNDI object store (see "Object Stores" on page 161)
`-t` *objectType*	Type of administered object: `q` Queue destination `t` Topic destination `cf` Connection factory `qf` Queue connection factory `tf` Topic connection factory `xcf` Connection factory for distributed transactions `xqf` Queue connection factory for distributed transactions `xtf` Topic connection factory for distributed transactions `e` SOAP endpoint (see *Message Queue Developer's Guide for Java Clients*)
`-o` *attribute*=*value*	Attributes of administered object (see "Administered Object Attributes" on page 164 and Chapter 16, "Administered Object Attribute Reference")
`-r` *readOnlyState*	Is administered object read-only? If `true`, client cannot modify object's attributes. Default value: `false`.
`-i` *fileName*	Name of command file containing all or part of subcommand clause

Table 13-10 Object Manager Options *(Continued)*

Option	Description
`-pre`	Preview results without performing command This option is useful for checking the values of default attributes.
`-javahome` *path*	Location of alternative Java runtime Default: Use runtime installed on system or bundled with Message Queue.
`-f`	Perform action without user confirmation
`-s`	Silent mode (no output displayed)
`-v`	Display version information[1]
`-h`	Display usage help[1]
`-H`	Display expanded usage help, including attribute list and examples[1]

1. Any other options specified on the command line are ignored .

Database Manager Utility

The Database Manager utility (`imqdbmgr`) sets up the database schema for a JDBC-based persistent data store. You can also use it to delete Message Queue database tables that have become corrupted or to change the data store. Table 13-11 lists the available subcommands.

Table 13-11 Database Manager Subcommands

Subcommand	Description
`create all`	Create new database and persistent store schema Used on embedded database systems. The broker property `imq.persist.jdbc.createdburl` must be specified.
`create tbl`	Create persistent store schema for existing database Used on external database systems.
`delete tbl`	Delete Message Queue database tables from current persistent store
`delete oldtbl`	Delete Message Queue database tables from earlier-version persistent store Used after the persistent store has been automatically migrated to the current version of Message Queue.

Table 13-11 Database Manager Subcommands *(Continued)*

Subcommand	Description
`recreate tbl`	Re-create persistent store schema Deletes all existing Message Queue database tables from the current persistent store and then re-creates the schema.
`reset lck`	Reset persistent store lock Resets the lock so that the persistent store database can be used by other processes.

Table 13-12 lists the options to the `imqdbmgr` command.

Table 13-12 Database Manager Options

Option	Description
`-b` *instanceName*	Instance name of broker
`-D`*property*=*value*	Set broker configuration property See "Persistence Properties" on page 292 for information about persistence-related broker configuration properties. **Caution:** Be careful to check the spelling and formatting of properties set with this option. Incorrect values will be ignored without notification or warning.
`-u` *name*	User name for authentication
`-p` *password*	Password for authentication[1]
`-passfile` *path*	Location of password file See "Using a Password File" on page 158 for more information.
`-v`	Display version information[2]
`-h`	Display usage help[2]

1. This option is being deprecated and will eventually be removed. Instead, either omit the password entirely (so that the command will prompt for it interactively) or use the `-passfile` option to specify a password file containing the password.
2. Any other options specified on the command line are ignored .

User Manager Utility

The User Manager utility (`imqusermgr`) is used for populating or editing a flat-file user repository. The utility must be run on the same host where the broker is installed; if a broker-specific user repository does not yet exist, you must first start up the corresponding broker instance in order to create it. You will also need the appropriate permissions to write to the repository: on the Solaris or Linux platforms, this means you must be either the root user or the user who originally created the broker instance.

Table 13-13 lists the subcommands available with the `imqusermgr` command. In all cases, the `-i` option specifies the instance name of the broker to whose user repository the command applies; if not specified, the default name `imqbroker` is assumed.

Table 13-13 User Manager Subcommands

Syntax	Description

`add [-i` *instanceName*`]` `-u` *userName* `-p` *password* `[-g` *group*`]`	Add user and password to repository
	The optional `-g` option specifies a group to which to assign this user:
	`admin`
	`user`
	`anonymous`

`delete [-i` *instanceName*`]` `-u` *userName*	Delete user from repository

`update [-i` *instanceName*`]` `-u` *userName* `-p` *password* `[-a` *activeState*`]`
`update [-i` *instanceName*`]` `-u` *userName* `-a` *activeState* `[-p` *password*`]`
	Set user's password or active state (or both)
	The `-a` option takes a boolean value specifying whether to make the user active (`true`) or inacftive (`false`). Default value: `true`.

`list [-i` *instanceName*`]` `[-u` *userName*`]`	Display user information If no user name is specified, all users in the repository are listed.

In addition, the options listed in Table 13-14 can be applied to any subcommand of the `imqusermgr` command.

Table 13-14 General User Manager Options

Option	Description
`-f`	Performs action without user Perform action without user confirmation.
`-s`	Silent mode (no output displayed)
`-v`	Display version information[1]
`-h`	Display usage help[1]

1. Any other options specified on the command line are ignored .

Service Administrator Utility

The Service Administrator utility (`imqsvcadmin`) utility installs a broker as a Windows service. Table 13-15 lists the available subcommands.

Table 13-15 Service Administrator Subcommands

Subcommand	Description
`install`	Install service
`remove`	Remove service
`query`	Display startup options Startup options can include whether the service is started manually or automatically, its location, the location of the Java runtime, and the values of arguments passed to the broker on startup.

Table 13-16 lists the options to the `imqsvcadmin` command.

Table 13-16 Service Administrator Options

Option	Description
`-javahome` *path*	Location of alternative Java runtime Default: Use runtime installed on system or bundled with Message Queue.
`-jrehome` *path*	Location of alternative Java Runtime Environment (JRE)

Table 13-16 Service Administrator Options *(Continued)*

Option	Description
`-vmargs` *arg1* [[*arg2*] …]	Additional arguments to pass to Java Virtual Machine running broker service[1] Example: `imqsvcadmin install -vmargs "-Xms16m -Xmx128m"`
`-args` *arg1* [[*arg2*] …]	Additional command line arguments to pass to broker service[1] Example: `imqsvcadmin install -args "-passfile d:\imqpassfile"` See "Broker Utility" on page 266 for information about broker command line arguments.
`-h`	Display usage help[2]

1. These arguments can also be specified in the Start Parameters field under the General tab in the service's Properties window (reached via the Services tool in the Windows Administrative Tools control panel).
2. Any other options specified on the command line are ignored .

Any information you specify using the `-javahome`, `-vmargs`, and `-args` options is stored in the Windows registry under the keys `JREHome`, `JVMArgs`, and `ServiceArgs` in the path

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\iMQ_Broker\Parameters
```

Key Tool Utility

The Key Tool utility (`imqkeytool`) generates a self-signed certificate for the broker, which can be used for the `ssljms`, `ssladmin`, or `cluster` connection service. The syntax is

```
imqkeytool -broker
```

On UNIX systems, you may need to run the utility from the `superuser` (`root`) account.

Chapter 14

Broker Properties Reference

This chapter provides reference information about configuration properties for a message broker. It consists of the following sections:

Connection Properties

Table 14-1 lists the broker properties related to connection services.

Table 14-1 Broker Connection Properties

Property	Type	Default	Description
`imq.service.activelist`	String	`jms,admin`	List of connection services, separated by commas, to be activated at broker startup
`imq.hostname`	String	All available IP addresses	Default host name or IP address for all connection services
`imq.portmapper.hostname`	String	None	Host name or IP address of Port Mapper If specified, overrides `imq.hostname`

Table 14-1 Broker Connection Properties *(Continued)*

Property	Type	Default	Description
`imq.portmapper.port`[1]	Integer	`7676`	Port number of Port Mapper **Note:** If multiple broker instances are running on the same host, each must be assigned a unique Port Mapper port.
`imq.`*serviceName*`.`*protocolType*`.hostname`	String	None	Host name or IP address for connection service[2] If specified, overrides `imq.hostname` for the designated connection service
`imq.`*serviceName*`.`*protocolType*`.port`	Integer	`0`	Port number for connection service[2] A value of `0` specifies that the port number should be allocated dynamically by the Port Mapper.
`imq.portmapper.backlog`	Integer	`50`	Maximum number of pending Port Mapper requests in operating system backlog
`imq.`*serviceName*`.threadpool_model`	String	`dedicated`	Threading model for thread pool management: `dedicated` Two dedicated threads per connection, one for incoming and one for outgoing messages `shared`[3] Connections processed by shared thread when sending or receiving messages The dedicated model limits the number of connections that can be supported, but provides higher performance; the shared model increases the number of possible connections, but at the cost of lower performance because of the additional overhead needed for thread management.

`imq.`*serviceName*`.min_threads`	Integer	`jms`: `10`
		`ssljms`: `10`
		`httpjms`: `10`
		`httpsjms`: `10`
		`admin`: `4`
		`ssladmin`: `4`
			Minimum number of threads maintained in connection service's thread pool
			
			When the number of available threads exceeds this threshold, threads will be shut down as they become free until the minimum is reached.
			
			The default value varies by connection service, as shown.

Table 14-1 Broker Connection Properties *(Continued)*

Property	Type	Default	Description

`imq.`*serviceName*`.max_threads`	Integer	`jms: 1000`
		`ssljms: 500`
		`httpjms: 500`
		`httpsjms: 500`
		`admin: 10`
		`ssladmin: 10`
			The number of threads beyond which no new threads are added to the thread pool for use by the named connection service. The number must be greater than zero and greater in value than the value of `min_threads`.
			The default value varies by connection service, as shown.

`imq.shared.connectionMonitor_limit`	Integer	Solaris: `512` Linux: `512` Windows: `64`	Maximum number of connections monitored by a distributor thread[4] The system allocates enough distributor threads to monitor all connections. The smaller the value of this property, the faster threads can be assigned to active connections. A value of `-1` denotes an unlimited number of connections per thread. The default value varies by operating-system platform, as shown.
`imq.ping.interval`	Integer	`120`	Interval, in seconds, at which to test connection between client and broker A value of `0` or `-1` disables periodic testing of the connection.

1. Can be used with `imqcmd update bkr` command
2. `jms`, `ssljms`, `admin`, and `ssladmin` services only; see Appendix C, "HTTP/HTTPS Support," for information on configuring the `httpjms` and `httpsjms` services
3. `jms` and `admin` services only
4. Shared threading model only

Routing Properties

Table 14-2 lists the broker properties related to routing services. Properties that configure the automatic creation of destinations are listed in Table 14-3.

Table 14-2 Broker Routing Properties

Property	Type	Default	Description
`imq.system.max_count`[1]	Integer	`-1`	Maximum number of messages held by broker A value of `-1` denotes an unlimited message count.

Table 14-2 Broker Routing Properties *(Continued)*

Property	Type	Default	Description
`imq.system.max_size`[1]	String	`-1`	Maximum total size of messages held by broker The value may be expressed in bytes, kilobytes, or megabytes, using the following suffixes: `b` Bytes `k` Killobytes (1024 bytes) `m` Megabytes (1024 x 1024 = 1,048,576 bytes) An unsuffixed value is expressed in bytes; a value of `-1` denotes an unlimited message capacity. Examples: `1600` 1600 bytes `1600b` 1600 bytes `16k` 16 kilobytes (= 16,384 bytes) `16m` 16 megabytes (= 16,777,216 bytes) `-1` No limit
`imq.message.max_size`[1]	String	`70m`	Maximum size of a single message body The syntax is the same as for `imq.system.max_size` (see above).
`imq.message.expiration.interval`	Integer	`60`	Interval, in seconds, at which expired messages are reclaimed

`imq.`*resourceState*`.threshold`	Integer	`green`: `0`
		`yellow`: `80`
		`orange`: `90`
		`red`: `98`
			Percent utilization at which memory resource state is triggered (where *resourceState* is `green`, `yellow`, `orange`, or `red`)

`imq.`*resourceState*`.count`	Integer	`green`: `5000`
		`yellow`: `500`
		`orange`: `50`
		`red`: `0`
			Maximum number of incoming messages allowed in a batch before checking whether memory resource state threshold has been reached (where *resourceState* is `green`, `yellow`, `orange`, or `red`)
			
			This limit throttles back message producers as system memory becomes increasingly scarce.

`imq.destination.DMQ.truncateBody`[1]	Boolean	`false`	Remove message body before storing in dead message queue? If `true`, only the message header and property data will be saved.

Table 14-2 Broker Routing Properties *(Continued)*

Property	Type	Default	Description
`imq.transaction.autorollback`	Boolean	`false`	Automatically roll back distributed transactions left in prepared state at broker startup? If `false`, you must manually commit or roll back transactions using the Command utility (`imqcmd`).

1. Can be used with `imqcmd update bkr` command

Table 14-3 Broker Properties for Auto-Created Destinations

Property	Type	Default	Description
`imq.autocreate.queue`[1,2]	Boolean	`true`	Allow auto-creation of queue destinations?
`imq.autocreate.topic`[3]	Boolean	`true`	Allow auto-creation of topic destinations?
`imq.autocreate.destination.maxNumMsgs`	Integer	`100000`	Maximum number of unconsumed messages A value of `-1` denotes an unliimited number of messages.
`imq.autocreate.destination.maxBytesPerMsg`	String	`10k`	Maximum size, in bytes, of any single message The value may be expressed in bytes, kilobytes, or megabytes, using the following suffixes: `b` Bytes `k` Killobytes (1024 bytes) `m` Megabytes (1024 x 1024 = 1,048,576 bytes) An unsuffixed value is expressed in bytes; a value of `-1` denotes an unlimited message size. Examples: `1600` 1600 bytes `1600b` 1600 bytes `16k` 16 kilobytes (= 16,384 bytes) `16m` 16 megabytes (= 16,777,216 bytes) `-1` No limit

Table 14-3 Broker Properties for Auto-Created Destinations *(Continued)*

Property	Type	Default	Description
`imq.autocreate.destination.maxTotalMsgBytes`	String	`10m`	Maximum total memory, in bytes, for unconsumed messages The syntax is the same as for `imq.autocreate.destination.maxBytesPerMsg` (see above).

`imq.autocreate.destination.limitBehavior`	String	`REJECT_NEWEST`	Broker behavior when memory-limit threshold reached:
			
			`FLOW_CONTROL`
			Slow down producers
			
			`REMOVE_OLDEST`
			Throw out oldest messages
			
			`REMOVE_LOW_PRIORITY`
			Throw out lowest-priority messages according to age; no notification to producing client
			
			`REJECT_NEWEST`
			Reject newest messages; notify producing client with an exception only if message is persistent
			
			If the value is `REMOVE_OLDEST` or `REMOVE_LOW_PRIORITY` and the `imq.autocreate.destination.useDMQ` property is `true`, excess messages are moved to the dead message queue.

`imq.autocreate.destination.maxNumProducers`	Integer	`100`	Maximum number of message producers for destination When this limit is reached, no new producers can be created. A value of `-1` denotes an unliimited number of producers.
`imq.autocreate.queue.maxNumActiveConsumers`[2]	Integer	`1`	Maximum number of active message consumers in load-balanced delivery from queue destination A value of `-1` denotes an unliimited number of consumers.

Table 14-3 Broker Properties for Auto-Created Destinations *(Continued)*

Property	Type	Default	Description
`imq.autocreate.queue.maxNumBackupConsumers`[2]	Integer	`0`	Maximum number of backup message consumers in load-balanced delivery from queue destination A value of `-1` denotes an unliimited number of consumers.
`imq.autocreate.queue.consumerFlowLimit`[2]	Integer	`1000`	Maximum number of messages delivered to queue consumer in a single batch In load-balanced queue delivery, this is the initial number of queued messages routed to active consumers before load balancing begins. A destination consumer can override this limit by specifying a lower value on a connection. A value of `-1` denotes an unliimited number of consumers.
`imq.autocreate.topic.consumerFlowLimit`[3]	Integer	`1000`	Maximum number of messages delivered to topic consumer in a single batch A value of `-1` denotes an unliimited number of consumers.
`imq.autocreate.destination.isLocalOnly`	Boolean	`false`	Local delivery only? This property applies only to destinations in broker clusters, and cannot be changed once the destination has been created. If `true`, the destination is not replicated on other brokers and is limited to delivering messages only to local consumers (those connected to the broker on which the destination is created).

Table 14-3 Broker Properties for Auto-Created Destinations *(Continued)*

Property	Type	Default	Description
`imq.autocreate.queue.localDeliveryPreferred`[2]	Boolean	`false`	Local delivery preferred? This property applies only to load-balanced queue delivery in broker clusters. If `true`, messages will be delivered to remote consumers only if there are no consumers on the local broker; the destination must not be restricted to local-only delivery (`imq.autocreate.destination.isLocalOnly` must be `false`).
`imq.autocreate.destination.useDMQ`	Boolean	`true`	Send dead messages to dead message queue? If `false`, dead messages will simply be discarded.

1. Can be used with `imqcmd update bkr` command
2. Queue destinations only
3. Topic destinations only

Persistence Properties

Message Queue supports both file-based and JDBC-based models for persistent data storage. The broker property `imq.persist.store` (Table 14-4) specifies which model to use. The following sections describe the broker configuration properties for the two models.

Table 14-4 Global Broker Persistence Property

Property	Type	Default	Description
`imq.persist.store`	String	`file`	Model for persistent data storage: `file` File-based persistence `jdbc` JDBC-based persistence

File-Based Persistence

Table 14-5 lists the broker properties related to file-based persistence.

Table 14-5 Broker Properties for File-Based Persistence

Property	Type	Default	Description
`imq.persist.file.message.max_record_size`	String	`1m`	Maximum-size message to add to message storage file Any message exceeding this size will be stored in a separate file of its own. The value may be expressed in bytes, kilobytes, or megabytes, using the following suffixes: `b` Bytes `k` Killobytes (1024 bytes) `m` Megabytes (1024 x 1024 = 1,048,576 bytes) An unsuffixed value is expressed in bytes. Examples: `1600` 1600 bytes `1600b` 1600 bytes `16k` 16 kilobytes (= 16,384 bytes) `16m` 16 megabytes (= 16,777,216 bytes)
`imq.persist.file.destination.message.filepool.limit`	Integer	`100`	Maximum number of free files available for reuse in destination file pool Free files in excess of this limit will be deleted. The broker will create and delete additional files in excess of the limit as needed. The higher the limit, the faster the broker can process persistent data.

Table 14-5 Broker Properties for File-Based Persistence *(Continued)*

Property	Type	Default	Description
`imq.persist.file.message.filepool.cleanratio`	Integer	`0`	Percentage of files in free file pools to be maintained in a clean (empty) state The higher this value, the less disk space is required for the file pool, but the more overhead is needed to clean files during operation.
`imq.persist.file.message.cleanup`	Boolean	`false`	Clean up files in free file pools on shutdown? Setting this property to `true` saves disk space for the file store, but slows broker shutdown.
`imq.persist.file.sync.enabled`	Boolean	`false`	Synchronize in-memory state with physical storage device? Setting this property to `true` eliminates data loss due to system crashes, but at a cost in performance. **Note:** If running Sun Cluster and the Sun Cluster Data Service for Message Queue, set this property to `true` for brokers on all cluster nodes.

JDBC-Based Persistence

Table 14-6 lists the broker properties related to JDBC-based persistence. Examples shown are for the PointBase® family of database products from DataMirror Mobile Solutions, Inc.

Table 14-6 Broker Properties for JDBC-Based Persistence

Property	Description	Example
`imq.persist.jdbc.brokerid`	*(Optional)* Broker instance identifier The identifier must be an alphanumeric string whose length does not exceed *n* - 12, where *n* is the maximum table name length allowed by the database. This identifier is appended to database table names to make them unique in the case where more than one broker instance is using the same database as a persistent data store. It is usually unnecessary for an embedded database, which stores data for only one broker instance.	Not required for PointBase embedded version
`imq.persist.jdbc.driver`	Java class name of JDBC driver for connecting to database	`com.pointbase.jdbc.jdbcUniversalDriver`

`imq.persist.jdbc.opendburl`	URL for opening connection to existing database	`jdbc:pointbase:embedded:`*dbName*`;`
		`database.home=`
		`.../instances/`*instanceName*`/dbstore`

`imq.persist.jdbc.createdburl`	*(Optional)* URL for creating new database
	
	This property is needed only if the database will be created using the Message Queue Database Manager utility (`imqdbmgr`).
		`jdbc:pointbase:embedded:`*dbName*`;`
		`new,database.home=`
		`.../instances/`*instanceName*`/dbstore`

`imq.persist.jdbc.closedburl`	*(Optional)* URL for closing database connection	Not required for PointBase

Table 14-6 Broker Properties for JDBC-Based Persistence *(Continued)*

Property	Description	Example
`imq.persist.jdbc.user`	*(Optional)* User name for opening database connection, if required. For security reasons, the value can be specified instead using command line options `imqbrokerd -dbuser` and `imqdbmgr -u`.	
`imq.persist.jdbc.needpassword`	*(Optional)* Does database require a password for broker access? If `true`, the `imqbrokerd` and `imqdbmgr` commands will prompt for a password, unless you use the `-passfile` option to specify a password file containing the password.	
`imq.persist.jdbc.password`	*(Optional)* Password for opening database connection This property should be specified only in a password file.	

`imq.persist.jdbc.table.IMQSV35`	SQL command to create version table	`CREATE TABLE ${`*name*`}`
		`(STOREVERSION INTEGER NOT NULL,`
		`BROKERID VARCHAR(100))`

`imq.persist.jdbc.table.IMQCCREC35`	SQL command to create configuration change record table	`CREATE TABLE ${`*name*`}`
		`(RECORDTIME BIGINT NOT NULL,`
		`RECORD BLOB(10k))`

`imq.persist.jdbc.table.IMQDEST35`	SQL command to create destination table	`CREATE TABLE ${`*name*`}`
		`(DID VARCHAR(100) NOT NULL,`
		`DEST BLOB(10k),`
		primaryKey`(DID))`

`imq.persist.jdbc.table.IMQINT35`	SQL command to create interest table	`CREATE TABLE ${name}`
		`(CUID BIGINT NOT NULL,`
		`INTEREST BLOB(10k),`
		primaryKey`(CUID))`

Table 14-6 Broker Properties for JDBC-Based Persistence *(Continued)*

Property	Description	Example

`imq.persist.jdbc.table.IMQMSG35`	SQL command to create message table
	
	The default maximum length for the `MSG` column is 1 megabyte (`1m`). If you expect to have messages larger than this, set the length accordingly. If the tables have already been created, you must recreate them to change the maximum message length.
		`CREATE TABLE ${`*name*`}`
		`(MID VARCHAR(100) NOT NULL,`
		`DID VARCHAR(100),`
		`MSGSIZE BIGINT,`
		`MSG BLOB(1m),`
		primaryKey`(MID))`

`imq.persist.jdbc.table.IMQPROPS35`	SQL command to create property table	`CREATE TABLE ${`*name*`}`
		`(PROPNAME VARCHAR(100) NOT NULL,`
		`PROPVALUE BLOB(10k),`
		primaryKey`(PROPNAME))`

`imq.persist.jdbc.table.IMQILIST35`	SQL command to create interest state table	`CREATE TABLE ${`*name*`}`
		`(MID VARCHAR(100) NOT NULL,`
		`CUID BIGINT,`
		`DID VARCHAR(100),`
		`STATE INTEGER,`
		primaryKey`(MID, CUID))`

`imq.persist.jdbc.table.IMQTXN35`	SQL command to create transaction table	`CREATE TABLE ${`*name*`}`
		`(TUID BIGINT NOT NULL,`
		`STATE INTEGER,`
		`TSTATEOBJ BLOB(10K),`
		primaryKey`(TUID))`

`imq.persist.jdbc.table.IMQTACK35`	SQL command to create transaction acknowledgment table	`CREATE TABLE ${`*name*`}`
		`(TUID BIGINT NOT NULL,`
		`TXNACK BLOB(10k))`

Security Properties

Table 14-7 lists the broker properties related to security services.

Table 14-7 Broker Security Properties

Property	Type	Default	Description
`imq.accesscontrol.enabled`	Boolean	`true`	Use access control? If `true`, the system will check the access control properties file to verify that an authenticated user is authorized to use a connection service or to perform specific operations with respect to specific destinations.
imq.*serviceName*`.accesscontrol.enabled`	Boolean	None	Use access control for connection service? If specified, overrides `imq.accesscontrol.enabled` for the designated connection service. If `true`, the system will check the access control properties file to verify that an authenticated user is authorized to use the designated connection service or to perform specific operations with respect to specific destinations.
`imq.accesscontrol.file.filename`	String	`accesscontrol.properties`	Name of access control properties file The file name specifies a path relative to the access control directory (see Appendix A).

Table 14-7 Broker Security Properties *(Continued)*

Property	Type	Default	Description
`imq.`*serviceName*`.accesscontrol.file.filename`	String	None	Name of access control properties file for connection service If specified, overrides `imq.accesscontrol.file.filename` for the designated connection service. The file name specifies a path relative to the access control directory (see Appendix A).
`imq.authentication.type`	String	`digest`	Password encoding method: `basic` Base-64 `digest` MD5
`imq.`*serviceName*`.authentication.type`	String	None	Password encoding method for connection service: `basic` Base-64 `digest` MD5 If specified, overrides `imq.authentication.type` for the designated connection service.
`imq.authentication.basic.user_repository`	String	`file`	Type of user repository for base-64 authentication: `file` File-based `ldap` LDAP
`imq.authentication.client.response.timeout`	Integer	`180`	Interval, in seconds, to wait for client response to authentication requests
`imq.passfile.enabled`	Boolean	`false`	Obtain passwords from password file?
`imq.passfile.dirpath`	String	See Appendix A	Path to directory containing password file
`imq.passfile.name`	String	`passfile`	Name of password file

Table 14-7 Broker Security Properties *(Continued)*

Property	Type	Default	Description
`imq.imqcmd.password`	String	None	Password for administrative user The Command utility (`imqcmd`) uses this password to authenticate the user before executing a command.
`imq.user_repository.ldap.server`	String	None	Host name and port number for LDAP server The value is of the form *hostName*`:`*port* where *hostName* is the fully qualified DNS name of the host running the LDAP server and *port* is the port number used by the server. To specify a list of failover servers, use the following syntax: *host1*`:`*port1* `ldap://`*host2*`:`*port2* `ldap://`*host3*`:`*port3* … Entries in the list are separated by spaces. Note that each failover server address is prefixed with `ldap://`. Use this format even if you use SSL and have set the property `imq.user_repository.ldap.ssl.enabled` to `true`. You need not specify `ldaps` in the address.

Table 14-7 Broker Security Properties *(Continued)*

Property	Type	Default	Description
`imq.user_repository.ldap.principal`	String	None	Distinguished name for binding to LDAP user repository Not needed if the LDAP server allows anonymous searches.
`imq.user_repository.ldap.password`	String	None	Password for binding to LDAP user repository Not needed if the LDAP server allows anonymous searches. This property should be specified only in password files.
`imq.user_repository.ldap.`*propertyName*	To come	To come	To come
`imq.user_repository.ldap.base`	String	None	Directory base for LDAP user entries
`imq.user_repository.ldap.uidattr`	String	None	Provider-specific attribute identifier for LDAP user name
`imq.user_repository.ldap.usrfilter`	String	None	*(Optional)* JNDI filter for LDAP user searches
`imq.user_repository.ldap.grpsearch`	Boolean	`false`	Enable LDAP group searches? **Note:** Message Queue does not support nested groups.
`imq.user_repository.ldap.grpbase`	String	None	Directory base for LDAP group entries
`imq.user_repository.ldap.gidattr`	String	None	Provider-specific attribute identifier for LDAP group name
`imq.user_repository.ldap.memattr`	String	None	Provider-specific attribute identifier for user names in LDAP group

Table 14-7 Broker Security Properties *(Continued)*

Property	Type	Default	Description
`imq.user_repository.ldap.grpfilter`	String	None	*(Optional)* JNDI filter for LDAP group searches
`imq.user_repository.ldap.timeout`	Integer	`280`	Time limit for LDAP searches, in seconds
`imq.user_repository.ldap.ssl.enabled`	Boolean	`false`	Use SSL when communicating with LDAP server?
`imq.keystore.file.dirpath`	String	See Appendix A	Path to directory containing key store file
`imq.keystore.file.name`	String	`keystore`	Name of key store file
`imq.keystore.password`	String	None	Password for key store file This property should be specified only in a password file.
`imq.audit.enabled`	Boolean	`false`	Start audit logging to broker log file? This option applies to Message Queue Enterprise Edition only.

Monitoring Properties

Table 14-8 lists the broker properties related to monitoring services.

Table 14-8 Broker Monitoring Properties

Property	Type	Default	Description

`imq.log.level`[1]	String	`INFO`	Logging level
			
			Specifies the categories of logging information that can be written to an output channel. Possible values, from high to low:
			`ERROR`
			`WARNING`
			`INFO`
			
			Each level includes those above it (for example, `WARNING` includes `ERROR`).

`imq.destination.logDeadMsgs`[1]	Boolean	`false`	Log information about dead messages? If `true`, the following events will be logged: • A destination is full, having reached its maximum size or message count. • The broker discards a message for a reason other than an administrative command or delivery acknowledgment. • The broker moves a message to the dead message queue.

`imq.log.console.stream`	String	`ERR`	Destination for console output:
			`OUT` `stdout`
			`ERR` `stderr`

Table 14-8 Broker Monitoring Properties *(Continued)*

Property	Type	Default	Description

`imq.log.console.output`	String	`ERROR|WARNING`	Categories of logging information to write to console:
			`NONE`
			`ERROR`
			`WARNING`
			`INFO`
			`ALL`
			The `ERROR`, `WARNING`, AND `INFO` categories do *not* include those above them, so each must be specified explicitly if desired. Any combination of categories can be specified, separated by vertical bars (|).

`imq.log.file.dirpath`	String	See Appendix A	Path to directory containing log file
`imq.log.file.filename`	String	`log.txt`	Name of log file

`imq.log.file.output`	String	`ALL`	Categories of logging information to write to log file:
			`NONE`
			`ERROR`
			`WARNING`
			`INFO`
			`ALL`
			The `ERROR`, `WARNING`, AND `INFO` categories do *not* include those above them, so each must be specified explicitly if desired. Any combination of categories can be specified, separated by vertical bars (|).

`imq.log.file.rolloverbytes`[1]	Integer	`-1`	File length, in bytes, at which output rolls over to a new log file A value of `-1` denotes an unlimited number of bytes (no rollover based on file length).
`imq.log.file.rolloversecs`[1]	Integer	`604800` (one week)	Age of file, in seconds, at which output rolls over to a new log file A value of `-1` denotes an unlimited number of seconds (no rollover based on file age).

Table 14-8 Broker Monitoring Properties *(Continued)*

Property	Type	Default	Description
imq.log.syslog.output[2]	String	ERROR	Categories of logging information to write to syslogd(1M): NONE ERROR WARNING INFO ALL The ERROR, WARNING, AND INFO categories do *not* include those above them, so each must be specified explicitly if desired. Any combination of categories can be specified, separated by vertical bars (|).
imq.log.syslog.facility[2]	String	LOG_DAEMON	syslog facility for logging messages Possible values mirror those listed on the syslog(3C) man page. Appropriate values for use with Message Queue include: LOG_USER LOG_DAEMON LOG_LOCAL0 LOG_LOCAL1 LOG_LOCAL2 LOG_LOCAL3 LOG_LOCAL4 LOG_LOCAL5 LOG_LOCAL6 LOG_LOCAL7
imq.log.syslog.identity[2]	String	imqbrokerd_${imq.*instanceName*}	Identity string to be prefixed to all messages logged to syslog
imq.log.syslog.logpid[2]	Boolean	true	Log broker process ID with message?
imq.log.syslog.logconsole[2]	Boolean	false	Write messages to system console if they cannot be sent to syslog?

Table 14-8 Broker Monitoring Properties *(Continued)*

Property	Type	Default	Description

`imq.log.timezone`	String	Local time zone	Time zone for log time stamps.
			Possible values are the same as those used by the method `java.util.TimeZone.getTimeZone`. Examples:
			`GMT`
			`GMT-8:00`
			`America/LosAngeles`
			`Europe/Rome`
			`Asia/Tokyo`

`imq.metrics.enabled`	Boolean	`true`	Enable writing of metrics information to Logger? Does not affect the production of metrics messages (controlled by `imq.metrics.topic.enabled`).
`imq.metrics.interval`	Integer	`-1`	Time interval, in seconds, at which to write metrics information to Logger Does not affect the time interval for production of metrics messages (controlled by `imq.metrics.topic.interval`). A value of `-1` denotes an indefinite interval (never write metrics information to the Logger).
`imq.metrics.topic.enabled`	Boolean	`true`	Enable production of metrics messages to metric topic destinations? If `false`, an attempt to subscribe to a metric topic destination will throw a client-side exception.
`imq.metrics.topic.interval`	Integer	`60`	Time interval, in seconds, at which to produce metrics messages to metric topic destinations
`imq.metrics.topic.persist`	Boolean	`false`	Are metrics messages sent to metric topic destinations persistent?
`imq.metrics.topic.timetolive`	Integer	`300`	Lifetime, in seconds, of metrics messages sent to metric topic destinations

1. Can be used with `imqcmd update bkr` command
2. Solaris platform only

Cluster Configuration Properties

Table 14-9 lists the configuration properties related to broker clusters.

Table 14-9 Broker Properties for Cluster Configuration

Property	Type	Default	Description
`imq.cluster.brokerlist`[1]	String	None	List of broker addresses The list consists of one or more addresses, separated by commas. Each address specifies the host name and Port Mapper port number of a broker in the cluster, in the form *hostName*`:`*portNumber*. Example: `host1:3000,host2:8000,ctrlhost`
`imq.cluster.hostname`[2]	String	None	Host name or IP address for `cluster` connection service If specified, overrides `imq.hostname` (see Table 14-1 on page 285) for the `cluster` connection service
`imq.cluster.port`[2]	Integer	`0`	Port number for `cluster` connection service A value of `0` specifies that the port number should be allocated dynamically by the Port Mapper.
`imq.cluster.transport`[1]	String	`tcp`	Network transport protocol for `cluster` connection service For secure, encrypted message delivery between brokers, set this property to `ssl`.

`imq.cluster.url`[1,3]	String	None	URL of `cluster` configuration file, if any
			Examples:
			`http://webserver/imq/cluster.properties`
			(for a file on a Web server)
			`file:/net/mfsserver/imq/cluster.properties`
			(for a file on a shared drive)

`imq.cluster.masterbroker`[1]	String	None	Host name and port number of cluster's master broker, if any The value has the form *hostName*`:`*portNumber*, where *hostName* is the host name of the master broker and *portNumber* is its Port Mapper port number. Example: `ctrlhost:7676`

1. Must have the same value for all brokers in a cluster
2. Can be specified independently for each broker in a cluster
3. Can be used with `imqcmd update bkr` command

Alphabetical List of Broker Properties

Table 14-10 is an alphabetical list of broker configuration properties, with cross-references to the relevant tables in this chapter.

Table 14-10 Alphabetical List of Broker Properties

Property	Table
`imq.accesscontrol.enabled`	Table 14-7 on page 298
`imq.accesscontrol.file.filename`	Table 14-7 on page 298
`imq.audit.enabled`	Table 14-7 on page 298
`imq.authentication.basic.user_repository`	Table 14-7 on page 298
`imq.authentication.client.response.timeout`	Table 14-7 on page 298
`imq.authentication.type`	Table 14-7 on page 298
`imq.autocreate.destination.isLocalOnly`	Table 14-3 on page 289
`imq.autocreate.destination.limitBehavior`	Table 14-3 on page 289
`imq.autocreate.destination.maxBytesPerMsg`	Table 14-3 on page 289
`imq.autocreate.destination.maxNumMsgs`	Table 14-3 on page 289
`imq.autocreate.destination.maxNumProducers`	Table 14-3 on page 289
`imq.autocreate.destination.maxTotalMsgBytes`	Table 14-3 on page 289
`imq.autocreate.destination.useDMQ`	Table 14-3 on page 289
`imq.autocreate.queue`	Table 14-3 on page 289
`imq.autocreate.queue.consumerFlowLimit`	Table 14-3 on page 289
`imq.autocreate.queue.localDeliveryPreferred`	Table 14-3 on page 289
`imq.autocreate.queue.maxNumActiveConsumers`	Table 14-3 on page 289
`imq.autocreate.queue.maxNumBackupConsumers`	Table 14-3 on page 289
`imq.autocreate.topic`	Table 14-3 on page 289
`imq.autocreate.topic.consumerFlowLimit`	Table 14-3 on page 289
`imq.cluster.brokerlist`	Table 14-9 on page 307
`imq.cluster.hostname`	Table 14-9 on page 307
`imq.cluster.masterbroker`	Table 14-9 on page 307
`imq.cluster.port`	Table 14-9 on page 307
`imq.cluster.transport`	Table 14-9 on page 307
`imq.cluster.url`	Table 14-9 on page 307

Table 14-10 Alphabetical List of Broker Properties *(Continued)*

Property	Table
`imq.destination.DMQ.truncateBody`	Table 14-2 on page 287
`imq.destination.logDeadMsgs`	Table 14-8 on page 303
`imq.hostname`	Table 14-1 on page 285
`imq.imqcmd.password`	Table 14-7 on page 298
`imq.keystore.file.dirpath`	Table 14-7 on page 298
`imq.keystore.file.name`	Table 14-7 on page 298
`imq.keystore.password`	Table 14-7 on page 298
`imq.keystore.`*propertyName*	Table 14-7 on page 298
`imq.log.console.output`	Table 14-8 on page 303
`imq.log.console.stream`	Table 14-8 on page 303
`imq.log.file.dirpath`	Table 14-8 on page 303
`imq.log.file.filename`	Table 14-8 on page 303
`imq.log.file.output`	Table 14-8 on page 303
`imq.log.file.rolloverbytes`	Table 14-8 on page 303
`imq.log.file.rolloversecs`	Table 14-8 on page 303
`imq.log.level`	Table 14-8 on page 303
`imq.log.syslog.facility`	Table 14-8 on page 303
`imq.log.syslog.identity`	Table 14-8 on page 303
`imq.log.syslog.logconsole`	Table 14-8 on page 303
`imq.log.syslog.logpid`	Table 14-8 on page 303
`imq.log.syslog.output`	Table 14-8 on page 303
`imq.log.timezone`	Table 14-8 on page 303
`imq.message.expiration.interval`	Table 14-2 on page 287
`imq.message.max_size`	Table 14-2 on page 287
`imq.metrics.enabled`	Table 14-8 on page 303
`imq.metrics.interval`	Table 14-8 on page 303
`imq.metrics.topic.enabled`	Table 14-8 on page 303
`imq.metrics.topic.interval`	Table 14-8 on page 303
`imq.metrics.topic.persist`	Table 14-8 on page 303
`imq.metrics.topic.timetolive`	Table 14-8 on page 303

Table 14-10 Alphabetical List of Broker Properties *(Continued)*

Property	Table
`imq.passfile.dirpath`	Table 14-7 on page 298
`imq.passfile.enabled`	Table 14-7 on page 298
`imq.passfile.name`	Table 14-7 on page 298
`imq.persist.file.destination.message.filepool.limit`	Table 14-5 on page 293
`imq.persist.file.message.cleanup`	Table 14-5 on page 293
`imq.persist.file.message.filepool.cleanratio`	Table 14-5 on page 293
`imq.persist.file.message.max_record_size`	Table 14-5 on page 293
`imq.persist.file.sync.enabled`	Table 14-5 on page 293
`imq.persist.jdbc.brokerid`	Table 14-6 on page 295
`imq.persist.jdbc.closedburl`	Table 14-6 on page 295
`imq.persist.jdbc.createdburl`	Table 14-6 on page 295
`imq.persist.jdbc.driver`	Table 14-6 on page 295
`imq.persist.jdbc.needpassword`	Table 14-6 on page 295
`imq.persist.jdbc.opendburl`	Table 14-6 on page 295
`imq.persist.jdbc.password`	Table 14-6 on page 295
`imq.persist.jdbc.table.IMQCCREC35`	Table 14-6 on page 295
`imq.persist.jdbc.table.IMQDEST35`	Table 14-6 on page 295
`imq.persist.jdbc.table.IMQILIST35`	Table 14-6 on page 295
`imq.persist.jdbc.table.IMQINT35`	Table 14-6 on page 295
`imq.persist.jdbc.table.IMQMSG35`	Table 14-6 on page 295
`imq.persist.jdbc.table.IMQPROPS35`	Table 14-6 on page 295
`imq.persist.jdbc.table.IMQSV35`	Table 14-6 on page 295
`imq.persist.jdbc.table.IMQTACK35`	Table 14-6 on page 295
`imq.persist.jdbc.table.IMQTXN35`	Table 14-6 on page 295
`imq.persist.jdbc.user`	Table 14-6 on page 295
`imq.persist.store`	Table 14-4 on page 292
`imq.ping.interval`	Table 14-1 on page 285
`imq.portmapper.backlog`	Table 14-1 on page 285
`imq.portmapper.hostname`	Table 14-1 on page 285
`imq.portmapper.port`	Table 14-1 on page 285

Table 14-10 Alphabetical List of Broker Properties *(Continued)*

Property	Table
`imq.`*resourceState*`.count`	Table 14-2 on page 287
`imq.`*resourceState*`.threshold`	Table 14-2 on page 287
`imq.service.activelist`	Table 14-1 on page 285
`imq.`*serviceName*`.accesscontrol.enabled`	Table 14-7 on page 298
`imq.`*serviceName*`.accesscontrol.file.filename`	Table 14-7 on page 298
`imq.`*serviceName*`.authentication.type`	Table 14-7 on page 298
`imq.`*serviceName*`.max_threads`	Table 14-1 on page 285
`imq.`*serviceName*`.min_threads`	Table 14-1 on page 285
`imq.`*serviceName*`.`*protocolType*`.hostname`	Table 14-1 on page 285
`imq.`*serviceName*`.`*protocolType*`.port`	Table 14-1 on page 285
`imq.`*serviceName*`.threadpool_model`	Table 14-1 on page 285
`imq.shared.connectionMonitor_limit`	Table 14-1 on page 285
`imq.system.max_count`	Table 14-2 on page 287
`imq.system.max_size`	Table 14-2 on page 287
`imq.transaction.autorollback`	Table 14-2 on page 287
`imq.user_repository.ldap.base`	Table 14-7 on page 298
`imq.user_repository.ldap.gidattr`	Table 14-7 on page 298
`imq.user_repository.ldap.grpbase`	Table 14-7 on page 298
`imq.user_repository.ldap.grpfilter`	Table 14-7 on page 298
`imq.user_repository.ldap.grpsearch`	Table 14-7 on page 298
`imq.user_repository.ldap.memattr`	Table 14-7 on page 298
`imq.user_repository.ldap.password`	Table 14-7 on page 298
`imq.user_repository.ldap.principal`	Table 14-7 on page 298
`imq.user_repository.ldap.`*propertyName*	Table 14-7 on page 298
`imq.user_repository.ldap.server`	Table 14-7 on page 298
`imq.user_repository.ldap.ssl.enabled`	Table 14-7 on page 298
`imq.user_repository.ldap.timeout`	Table 14-7 on page 298
`imq.user_repository.ldap.uidattr`	Table 14-7 on page 298
`imq.user_repository.ldap.usrfilter`	Table 14-7 on page 298

Chapter 15

Physical Destination Property Reference

This chapter provides reference information about configuration properties for physical destinations. These properties can be set when creating or updating a physical destination. For auto-created destinations, you set default values in the broker's instance configuration file (see Table 14-3 on page 289).

Table 15-1 Physical Destination Properties

Property	Type	Default	Description
`maxNumMsgs`[1]	Integer	`-1`	Maximum number of unconsumed messages A value of `-1` denotes an unliimited number of messages. For the dead message queue, the default value is `1000`.
`maxBytesPerMsg`	String	`-1`	Maximum size, in bytes, of any single message Rejection of a persistent message is reported to the producing client with an exception; no notification is sent for nonpersistent messages. The value may be expressed in bytes, kilobytes, or megabytes, using the following suffixes: `b` Bytes `k` Killobytes (1024 bytes) `m` Megabytes (1024 x 1024 = 1,048,576 bytes) An unsuffixed value is expressed in bytes; a value of `-1` denotes an unlimited message size. Examples: `1600` 1600 bytes `1600b` 1600 bytes `16k` 16 kilobytes (= 16,384 bytes) `16m` 16 megabytes (= 16,777,216 bytes) `-1` No limit

Table 15-1 Physical Destination Properties *(Continued)*

Property	Type	Default	Description
maxTotalMsgBytes[1]	String	-1	Maximum total memory, in bytes, for unconsumed messages
			The syntax is the same as for maxBytesPerMsg (see above).
			For the dead message queue, the default value is 10m.
limitBehavior	String	REJECT_NEWEST	Broker behavior when memory-limit threshold reached:
			FLOW_CONTROL Slow down producers
			REMOVE_OLDEST Throw out oldest messages
			REMOVE_LOW_PRIORITY Throw out lowest-priority messages according to age; no notification to producing client
			REJECT_NEWEST Reject newest messages; notify producing client with an exception only if message is persistent
			If the value is REMOVE_OLDEST or REMOVE_LOW_PRIORITY and the useDMQ property is true, excess messages are moved to the dead message queue. For the dead message queue itself, the default limit behavior is REMOVE_OLDEST and cannot be set to FLOW_CONTROL.
maxNumProducers[2]	Integer	-1	Maximum number of message producers for destination
			When this limit is reached, no new producers can be created. A value of -1 denotes an unliimited number of producers.
maxNumActiveConsumers[3]	Integer	1	Maximum number of active message consumers in load-balanced delivery from queue destination
			A value of -1 denotes an unliimited number of consumers. In Sun Java System Message Queue Platform Edition, the value is limited to 2.
maxNumBackupConsumers[3]	Integer	0	Maximum number of backup message consumers in load-balanced delivery from queue destination
			A value of -1 denotes an unliimited number of consumers. In Sun Java System Message Queue Platform Edition, the value is limited to 1.

Table 15-1 Physical Destination Properties *(Continued)*

Property	Type	Default	Description
`consumerFlowLimit`	Integer	`1000`	Maximum number of messages delivered to consumer in a single batch In load-balanced queue delivery, this is the initial number of queued messages routed to active consumers before load balancing begins. A destination consumer can override this limit by specifying a lower value on a connection. A value of `-1` denotes an unliimited number of consumers.
`isLocalOnly`[2]	Boolean	`false`	Local delivery only? This property applies only to destinations in broker clusters, and cannot be changed once the destination has been created. If `true`, the destination is not replicated on other brokers and is limited to delivering messages only to local consumers (those connected to the broker on which the destination is created).
`localDeliveryPreferred`[2,3]	Boolean	`false`	Local delivery preferred? This property applies only to load-balanced queue delivery in broker clusters. If `true`, messages will be delivered to remote consumers only if there are no consumers on the local broker; the destination must not be restricted to local-only delivery (`isLocalOnly` must be `false`).
`useDMQ`[2]	Boolean	`true`	Send dead messages to dead message queue? If `false`, dead messages will simply be discarded.

1. In a cluster environment, applies to each individual instance of a destination rather than collectively to all instances in the cluster
2. Does not apply to dead message queue
3. Queue destinations only

Chapter 16

Administered Object Attribute Reference

This chapter provides reference information about the attributes of administered objects. It consists of the following sections:

Connection Factory Attributes

The attributes of a connection factory object are grouped into categories described in the following sections below:

Connection Handling

Table 16-1 lists the connection factory attributes for connection handling.

Table 16-1 Connection Factory Attributes for Connection Handling

Attribute	Type	Default	Description
`imqAddressList`	String	An existing Message Queue 3.0 address, if any; if none, the first entry in Table 16-2 on page 320	List of broker addresses The list consists of one or more message server addresses, separated by commas. Each address specifies (or implies) the host name, port number, and connection service for a broker instance to which the client can connect. Address syntax varies depending on the connection service and port assignment method; see below for details.
`imqAddressListBehavior`	String	`PRIORITY`	Order in which to attempt connection to server addresses: `PRIORITY` Order specified in address list `RANDOM` Random order **NOTE:** If many clients share the same connection factory, specify random connection order to prevent them from all attempting to connect to the same address.
`imqAddressListIterations`	Integer	`5`	Number of times to iterate through address list attempting to establish or reestablish a connection A value of `-1` denotes an unlimited number of iterations.
`imqPingInterval`	Integer	`30`	Interval, in seconds, at which to test connection between client and broker A value of `0` or `-1` disables periodic testing of the connection.
`imqReconnectEnabled`	Boolean	`false`	Attempt to reestablish a lost connection?
`imqReconnectAttempts`	Integer	`0`	Number of times to attempt connection (or reconnection) to each address in address list before moving on to next A value of `-1` denotes an unliimited number of connection attempts; attempt repeatedly to connect to first address until successful.

Table 16-1 Connection Factory Attributes for Connection Handling *(Continued)*

Attribute	Type	Default	Description
`imqReconnectInterval`	Long integer	`3000`	Interval, in milliseconds, between reconnection attempts This value applies both for successive attempts on a given address and for successive addresses in the list. **NOTE:** Too small a value may give the broker insufficient recovery time; too large a value may cause unacceptable connection delays.
`imqSSLIsHostTrusted`	Boolean	`true`	Accept broker's self-signed authentication certificate? **NOTE:** To use signed certificates from a certificate authority, set this attribute to `false`.

The value of the `imqAddressList` attribute is a comma-separated string specifying one or more message server addresses to which to connect. The general syntax for each address is as follows:

scheme`://`*address*

where *scheme* identifies one of the addressing schemes shown in the first column of Table 16-2 and *address* denotes the server address itself. The exact syntax for specifying the address depends on the addressing scheme, as shown in the last column of the table.

Table 16-2 Message Server Addressing Schemes

Scheme	Service	Syntax	Description
`mq`	`jms` or `ssljms`	[*hostName*][:*portNumber*][/*serviceName*]	Assign port dynamically for `jms` or `ssljms` connection service. The address list entry specifies the host name and port number for the Message Queue Port Mapper. The Port Mapper itself dynamically assigns a port to be used for the connection. Default values: *hostName* = `localhost` *portNumber* = `7676` *serviceName* = `jms` For the `ssljms` connection service, all variables must be specified explicitly.
`mqtcp`	`jms`	*hostName*:*portNumber*/`jms`	Connect to specified port using `jms` connection service. Bypasses the Port Mapper and makes a TCP connection directly to the specified host name and port number.
`mqssl`	`ssljms`	*hostName*:*portNumber*/`ssljms`	Connect to specified port using `ssljms` connection service. Bypasses the Port Mapper and makes a secure SSL connection directly to the specified host name and port number.

`http`	`httpjms`	`http://`*hostName*:*portNumber*/*contextRoot*/`tunnel`
		
		If multiple broker instances use the same tunnel servlet, the following syntax connects to a specific broker instance rather than a randomly selected one:
		
		`http://`*hostName*:*portNumber*/*contextRoot*/`tunnel?`
		`ServerName=`*hostName*:*instanceName*
			Connect to specified port using `httpjms` connection service.
			
			Makes an HTTP connection to a Message Queue tunnel servlet at the specified URL. The broker must be configured to access the HTTP tunnel servlet.

Table 16-2 Message Server Addressing Schemes *(Continued)*

Scheme	Service	Syntax	Description
`https`	`httpsjms`	`https://`*hostName*`:`*portNumber*`/`*contextRoot*`/tunnel` If multiple broker instances use the same tunnel servlet, the following syntax connects to a specific broker instance rather than a randomly selected one: `https://`*hostName*`:`*portNumber*`/`*contextRoot*`/tunnel?ServerName=`*hostName*`:`*instanceName*	Connect to specified port using `httpsjms` connection service. Makes a secure HTTPS connection to a Message Queue tunnel servlet at the specified URL. The broker must be configured to access the HTTPS tunnel servlet.

Table 16-3 shows examples of the various address formats.

Table 16-3 Message Server Address Examples

Service	Broker Host	Port	Example Address
Not specified	Not specified	Not specified	No address (`mq://localHost:7676/jms`)
Not specified	Specified host	Not specified	`myBkrHost` (`mq://myBkrHost:7676/jms`)
Not specified	Not specified	Specified Port Mapper port	`1012` (`mq://localHost:1012/jms`)
`ssljms`	Local host	Standard Port Mapper port	`mq://localHost:7676/ssljms`
`ssljms`	Specified host	Standard Port Mapper port	`mq://myBkrHost:7676/ssljms`
`ssljms`	Specified host	Specified Port Mapper port	`mq://myBkrHost:1012/ssljms`
`jms`	Local host	Specified service port	`mqtcp://localhost:1032/jms`
`ssljms`	Specified host	Specified service port	`mqssl://myBkrHost:1034/ssljms`
`httpjms`	Not applicable	Not applicable	`http://websrvr1:8085/imq/tunnel`
`httpsjms`	Not applicable	Not applicable	`https://websrvr2:8090/imq/tunnel`

Client Identification

Table 16-4 lists the connection factory attributes for client identification.

Table 16-4 Connection Factory Attributes for Client Identification

Attribute	Type	Default	Description
`imqDefaultUsername`	String	`guest`	Default user name for authenticating with broker
`imqDefaultPassword`	String	`guest`	Default password for authenticating with broker
`imqConfiguredClientID`	String	`null`	Administratively configured client identifier
`imqDisableSetClientID`	Boolean	`false`	Prevent client from changing client identifier using `setClientID` method?

Reliability and Flow Control

Table 16-5 lists the connection factory attributes for reliability and flow control.

Table 16-5 Connection Factory Attributes for Reliability and Flow Control

Attribute	Type	Default	Description
`imqAckTimeout`	String	`0`	Maximum time, in milliseconds, to wait for broker acknowledgment before throwing an exception A value of `0` denotes no timeout (wait indefinitely). **NOTE:** In some situations, too low a value can cause premature timeout: for example, initial authentication of a user against an LDAP user repository using a secure (SSL) connection can take more than 30 seconds.
`imqConnectionFlowCount`	Integer	`100`	Number of payload messages in a metered batch Delivery of payload messages to the client is temporarily suspended after this number of messages, allowing any accumulated control messages to be delivered. Payload message delivery is resumed on notification by the client runtime, and continues until the count is again reached. A value of `0` disables metering of message delivery and may cause Message Queue control messages to be blocked by heavy payload message traffic.
`imqConnectionFlowLimitEnabled`	Boolean	`false`	Limit message flow at connection level?

Table 16-5 Connection Factory Attributes for Reliability and Flow Control *(Continued)*

Attribute	Type	Default	Description
`imqConnectionFlowLimit`	Integer	`1000`	Maximum number of messages per connection to deliver and buffer for consumption Message delivery on a connection stops when the number of unconsumed payload messages pending (subject to flow metering governed by `imqConnectionFlowCount`) exceeds this limit. Delivery resumes only when the number of pending messages falls below the limit. This prevents the client from being overwhelmed with pending messages that might cause it to run out of memory. This attribute is ignored if `imqConnectionFlowLimitEnabled` is `false`.
`imqConsumerFlowLimit`	Integer	`100`	Maximum number of messages per consumer to deliver and buffer for consumption Message delivery to a given consumer stops when the number of unconsumed payload messages pending for that consumer exceeds this limit. Delivery resumes only when the number of pending messages for the consumer falls below the percentage specified by `imqConsumerFlowThreshold`. This canbe used to improve load balancing among multiple consumers and prevent any single consumer from starving others on the same connection. This limit can be overridden by a lower value set for a queue's own `consumerFlowLimit` attribute (see Chapter 15, "Physical Destination Property Reference"). Note also that message delivery to all consumers on a connection is subject to the overall limit specified by `imqConnectionFlowLimit`.
`imqConsumerFlowThreshold`	Integer	`50`	Number of messages per consumer buffered in the client runtime, as a percentage of `imqConsumerFlowLimit`, below which to resume message delivery

Queue Browser and Server Sessions

Table 16-6 lists the connection factory attributes for queue browsing and server sessions.

Table 16-6 Connection Factory Attributes for Queue Browser and Server Sessions

Attribute	Type	Default	Description
`imqQueueBrowserMaxMessagesPerRetrieve`	Integer	`1000`	Maximum number of messages to retrieve at one time when browsing contents of a queue destination
`imqQueueBrowserRetrieveTimeout`	Long integer	`60000`	Maximum time, in milliseconds, to wait to retrieve messages, when browsing contents of a queue destination, before throwing an exception
`imqLoadMaxToServerSession`	Boolean	`true`	Load up to maximum number of messages into a server session? If `false`, the client will load only a single message at a time. This attribute applies only to JMS application server facilities.

Setting Standard Message Properties

The connection factory attributes listed in Table 16-7 control whether the Message Queue client runtime sets certain standard message properties defined in the *Java Message Service Specification*.

Table 16-7 Connection Factory Attributes for Standard Message Properties

Property	Type	Default	Description
`imqSetJMSXUserID`	Boolean	`false`	Set `JMSXUserID` property (identity of user sending message) for produced messages?
`imqSetJMSXAppID`	Boolean	`false`	Set `JMSXAppID` property (identity of application sending message) for produced messages?
`imqSetJMSXProducerTXID`	Boolean	`false`	Set `JMSXProducerTXID` property (transaction identifier of transaction within which message was produced) for produced messages?
`imqSetJMSXConsumerTXID`	Boolean	`false`	Set `JMSXConsumerTXID` property (transaction identifier of transaction within which message was consumed) for consumed messages?

Table 16-7 Connection Factory Attributes for Standard Message Properties *(Continued)*

Property	Type	Default	Description
`imqSetJMSXRcvTimestamp`	Boolean	`false`	Set `JMSXRcvTimestamp` property (time message delivered to consumer) for consumed messages?

Message Header Overrides

Table 16-8 lists the connection factory attributes for overriding JMS message header fields.

Table 16-8 Connection Factory Attributes for Message Header Overrides

Attribute	Type	Default	Description
`imqOverrideJMSDeliveryMode`	Boolean	`false`	Allow client-set delivery mode to be overridden?
`imqJMSDeliveryMode`	Integer	`2`	Overriding value of delivery mode: `1` Nonpersistent `2` Persistent
`imqOverrideJMSExpiration`	Boolean	`false`	Allow client-set expiration time to be overridden?
`imqJMSExpiration`	Long integer	`0`	Overriding value of expiration time, in milliseconds A value of `0` denotes an unlimited expiration time (message never expires).
`imqOverrideJMSPriority`	Boolean	`false`	Allow client-set priority level to be overridden?
`imqJMSPriority`	Integer	`4` (normal)	Overriding value of priority level (`0` to `9`)
`imqOverrideJMSHeadersToTemporaryDestinations`	Boolean	`false`	Apply overrides to temporary destinations?

Destination Attributes

Table 16-9 lists the attributes that can be set for a destination administered object.

Table 16-9 Destination Attributes

Attribute	Type	Default	Description
`imqDestinationName`	String	`Untitled_Destination_Object`	Name of physical destination The destination name may contain only alphanumeric characters (no spaces) and must begin with an alphabetic character or the underscore (_) or dollar sign ($) character. It may not begin with the characters `mq`.
`imqDestinationDescription`	String	None	Descriptive string for destination

SOAP Endpoint Attributes

Table 16-10 lists the attributes used to configure endpoint URLs for applications that use the Simple Object Access Protocol (SOAP); see the *Message Queue Developer's Guide for Java Clients* for more information.

Table 16-10 SOAP Endpoint Attributes

Attribute	Type	Default	Description
`imqSOAPEndpointList`	String	None	List of one of more URLs representing SOAP endpoints to which to send messages, separated by spaces Each URL should be associated with a servlet that can receive and process SOAP messages. Example: `http://www.serv1/ http://www.serv2/` If the list specifies more than one URL, messages are broadcast to all of them.
`imqEndpointName`	String	`Untitled_Endpoint_Object`	Name of SOAP endpoint
`imqEndpointDescription`	String	None	Descriptive string for SOAP endpoint. Example: `My endpoints for broadcast`

Chapter 17

JMS Resource Adapter Property Reference

The Message Queue JMS Resource Adapter (JMS RA) enables you to integrate Sun Java System Message Queue with any J2EE 1.4 application server, by means of the standard J2EE connector architecture (JCA). When the Message Queue JMS Resource Adapter is plugged into an application server, an application deployed in that application server can use Message Queue to send and receive JMS messages.

The Message Queue JMS Resource Adapter exposes its configuration properties through three JavaBean components:

- `ResourceAdapter` configuration affects the behavior of the Resource Adapter as a whole.
- `ManagedConnectionFactory` configuration affects connections created by the Resource Adapter for use by message-driven beans (MDBs).
- `ActivationSpec` configuration affects message endpoints that represent message-driven beans MDBs in their interactions with the messaging system.

To set property values for these entities, you use the tools that your application server provides for configuration and deployment of the Resource Adapter and for deployment of MDBs.

This chapter lists and describes the configuration properties of the Message Queue JMS Resource Adapter. It contains the following sections:

- "ResourceAdapter JavaBean" on page 328
- "ManagedConnectionFactory JavaBean" on page 329
- "ActivationSpec JavaBean" on page 331

ResourceAdapter JavaBean

The `ResourceAdapter` configuration configures the default JMS Resource Adapter behavior. Table 17-1 lists and describes the properties with which you can configure this JavaBean. A footnote marks each required property.

Table 17-1 Resource Adapter Properties

Property	Default	Description
`addressList`[1]	`mq://localhost:7676/jms`	The connection that the Resource Adapter makes to the Message Queue service, specified using the message service address format. The Resource Adapter supplies the default value. This property name, `addressList`, is specific to Sun Java System Message Queue, but has the same meaning as the standard property `connectionURL`. Sun Java System Message Queue provides both property names. You must set either `connectionURL` or `addressList`; they are equivalent.
`addressListBehavior`	`PRIORITY`	A string specifying how the Resource Adapter connects to the Message Queue service. The value is `PRIORITY` or `RANDOM`. A `PRIORITY` connection selects a Message Queue broker by choosing the first specified in the address list (`addressList`). A `RANDOM` connection selects a Message Queue broker randomly from the address list. Reconnection after a connection failure is the same for `PRIORITY` and `RANDOM`. A reconnection attempt starts with the broker whose connection failed. If that attempt is unsuccessful, the Resource Adapter proceeds sequentially through the active address list.
`addressListIterations`	`1`	The number of times to iterate through the address list. This value applies to the initial connection and to subsequent reconnection attempts.
`connectionURL`	`mq://localhost:7676/jms`	The connection that the Resource Adapter makes to the Message Queue service, specified using the message service address format. Equivalent to the `addressList` property; see description above for further details.
`userName`[1]	`guest`	The default user name with which the Resource Adapter connects to the Message Queue service. The Resource Adapter supplies the default value.

Table 17-1 Resource Adapter Properties *(Continued)*

Property	Default	Description
`password`[1]	`guest`	The default password with which the Resource Adapter connects to the Message Queue service. The Resource Adapter supplies the default value.
`reconnectAttempts`	`6`	The number of times to attempt reconnection to a single entry in the address list. This property is used when `reconnectEnabled` is set to `true`.
`reconnectEnabled`	`false`	A boolean value specifying whether to attempt reconnection after a connection failure. The behavior of a reconnection attempt is governed by the values for `reconnectInterval` and `reconnectAttempts`.
`reconnectInterval`	`30000`	The interval between reconnection attempts, in milliseconds. This property is used when `reconnectEnabled` is set to `true`.

1. This property is required.

ManagedConnectionFactory JavaBean

A managed connection factory provides and defines the connections that the Resource Adapter provides to a message-driven bean. If you set an attribute for which the `ResourceAdapter` JavaBean has an analogous property, the setting supersedes the analogous value specified for the `ResourceAdapter` bean.

Table 17-2 lists and describes the configurable attributes of a managed connection factory provided by the Message Queue Resource Adapter.

Table 17-2 Managed Connection Factory Attributes

Attribute	Default	Description
`addressList`	None	A list of connections derived from this managed connection factory. The format of this attribute adheres to the Message Service `addressList`, as described in Table 17-1 on page 328. If this value is not set, connections use the `addressList` value specified for the `ResourceAdapter` JavaBean and described in that table.

Table 17-2 Managed Connection Factory Attributes *(Continued)*

Attribute	Default	Description
`addressListBehavior`	`PRIORITY`	A string specifying how the Resource Adapter connects to the Message Queue service. The value is `PRIORITY` or `RANDOM`. A `PRIORITY` connection selects a Message Queue broker by choosing the first specified in the address list (`addressList`). A `RANDOM` connection selects a Message Queue broker randomly from the address list. Reconnection after a connection failure is the same for `PRIORITY` and `RANDOM`. A reconnection attempt starts with the broker whose connection failed. If that is unsuccessful, the connection attempts proceed sequentially through the active address list.
`addressListIterations`	`1`	The number of times to iterate through the address list. This value applies to the initial connection and to subsequent reconnection attempts.
`clientID`	None	The client identifier to use for connections derived from this managed connection factory.
`password`	`guest`	*(Optional)* The password for connections. If this value is not set, connections use the password specified for the `ResourceAdapter` JavaBean, as described in Table 17-1 on page 328.
`reconnectAttempts`	`6`	The number of times to attempt reconnection to a single entry in the address list.
`reconnectEnabled`	`false`	A boolean value specifying whether to attempt reconnection after failure of a connection or a new connection attempt. The reconnection attempt is governed by the `reconnectInterval` and `reconnectAttempts` attributes.
`reconnectInterval`	`30000`	The minimum number of milliseconds to wait between attempts to reconnect to the Message Queue service.
`userName`	`guest`	*(Optional)* The user name for connections. If this value is not set, connections use the user name specified for the `ResourceAdapter` JavaBean, as described in Table 17-1 on page 328.

ActivationSpec JavaBean

`ActivationSpec` JavaBean properties are used by the application server when it instructs the Resource Adapter to activate a message endpoint and associate the message endpoint with a message-driven bean.

Table 17-3 lists and describes the configurable properties for a message endpoint activation specification. The table indicates the properties that are specific to the Message Queue Resource Adapter and the properties that are specific to the Enterprise JavaBean 2.1 standard or J2EE Connector Architecture (J2EE CA) 1.5 standard.

Table 17-3 Activation Specification Properties

Property	Default	Description
`acknowledgeMode`	`Auto-acknowledge`	*(Optional)* The JMS session acknowledgment mode to use for the consumer. This is a standard EJB 2.1 and J2EE CA 1.5 property. The value can be `Auto-acknowledge` or `Dups-ok-acknowledge`.
`addressList`	Inherited from `addressList` in the `ResourceAdapter` JavaBean configuration	*(Optional)* The specification of the connection made by the Resource Adapter on behalf of the message endpoint. This property is specific to the Message Queue JMS Resource Adapter. The valid values must conform to the message service connection address syntax.
`clientId`	None	The JMS client ID to be used by the JMS connection created for this consumer. You must set this property if you set the `subscriptionDurability` property to `Durable`. This is a standard EJB 2.1 and J2EE CA 1.5 property.
`customAcknowledgeMode`	None	A string specifying the mode for MDB message consumption. The valid values for this property are `No_acknowledge` or null. You can use `No_acknowledge` mode only for a non-transacted, non-durable topic subscription. If you use this setting with a transacted subscription or a durable subscription, subscription activation fails.

Table 17-3 Activation Specification Properties *(Continued)*

Property	Default	Description
`destination`	None	The name of the destination from which this MDB consumes messages. This is a required property. It is a standard EJB 2.1 and J2EE CA1.5 property. The value must be set to the value of the `destinationName` property for a Message Queue destination administered object.
`destinationType`	None	The type of destination specified by the `destination` property. Valid values are `javax.jms.Queue` or `javax.jms.Topic`. This is a required property. It is a standard EJB 2.1 and J2EE CA1.5 property.
`endpointExceptionRedelivery Attempts`	`6`	The number of times to redeliver a message to the MDB when the MDB throws an exception during message delivery.
`messageSelector`	None	*(Optional)* A JMS message selector to use for filtering the messages delivered to the consumer. The value is of type String. This is a standard EJB 2.1 and J2EE CA 1.5 property.
`sendUndeliverableMsgsToDMQ`	`true`	A boolean value specifying whether to place a message in the dead message queue when the MDB throws a runtime exception and the number of redelivery attempts exceeds the value of `endpointExceptionRedeliveryAttempts`. If `false`, the Message Queue broker will attempt redelivery of the message to any valid consumer, including the same MDB.
`subscriptionDurability`	`NonDurable`	A string specifying whether a consumer for a topic destination is durable or nondurable. The value can be `NonDurable` or `Durable`. This property is optional for nondurable subscriptions and required for durable subscriptions. If you set this value to `Durable`, you must also set the properties `clientID` and `subscriptionName`. This is a standard EJB 2.1 and J2EE CA1.5 property and is valid only if the `destinationType` property is set to `avax.jms.Topic`.

Table 17-3 Activation Specification Properties *(Continued)*

Property	Default	Description
`subscriptionName`	None	A string to use to name durable subscriptions. You must set this property if you set `subscriptionDurability` property to `Durable`. This is a standard EJB 2.1 and J2EE CA 1.5 property.

Chapter 18

Metrics Reference

This chapter lists and describes metrics produced by the Message Queue product. This chapter contains the following sections:

JVM Metrics

Table 18-1 lists and describes the metrics data that the broker generates for the broker process JVM heap. For each metric, the table shows which metrics monitoring tools provide it.

Table 18-1 JVM Metrics

Metric Quantity	**Description**	**imqcmd `metrics bkr` (*metricType*)**	**Log File**	**Metrics Message (*metrics topic*)**[2]
JVM heap: free memory	Amount of free memory available for use in the JVM heap	Yes (`cxn`)	Yes	Yes (…`jvm`)
JVM heap: total memory	Current JVM heap size	Yes (`cxn`)	Yes	Yes (…`jvm`)
JVM heap: max memory	Maximum to which the JVM heap size can grow.	No	Yes[1]	Yes (…`jvm`)

1. Shown only at broker startup.
2. For metrics topic destination names, see Table 10-7 on page 202.

Brokerwide Metrics

Table 18-2 lists and describes the data the broker reports regarding brokerwide metrics information. It also shows which of the data can be obtained using the different metrics monitoring tools.

Table 18-2 Brokerwide Metrics

Metric Quantity	**Description**	**imqcmd `metrics bkr` (*metricType*)**	**Log File**	**Metrics Message (*metrics topic*)**[1]
Connection Data				
Num connections	Number of currently open connections to the broker	Yes (`cxn`)	Yes	Yes (…broker)
Num threads	Total number of threads currently in use for all connection services	Yes (`cxn`)	Yes	No
Min threads	Number of threads, which once reached, are maintained in the thread pool for use by connection services	Yes (`cxn`)	Yes	No
Max threads	Number of threads, beyond which no new threads are added to the thread pool for use by connection services	Yes (`cxn`)	Yes	No
Stored Messages Data				
Num messages	Number of payload messages currently stored in broker memory and persistent store	No Use `query bkr`	No	Yes (…broker)
Total message bytes	Number of payload messages bytes currently stored in broker memory and persistent store	No Use `query bkr`	No	Yes (…broker)
Message Flow Data				
Num messages in	Number of payload messages that have flowed into the broker since it was last started	Yes (`ttl`)	Yes	Yes (…broker)
Message bytes in	Number of payload message bytes that have flowed into the broker since it was last started	Yes (`ttl`)	Yes	Yes (…broker)
Num packets in	Number of packets that have flowed into the broker since it was last started; includes both payload messages and control messages	Yes (`ttl`)	Yes	Yes (…broker)

Table 18-2 Brokerwide Metrics *(Continued)*

Metric Quantity	Description	imqcmd `metrics bkr` (*metricType*)	Log File	Metrics Message (*metrics topic*)[1]
Packet bytes in	Number of packet bytes that have flowed into the broker since it was last started; includes both payload messages and control messages	Yes (`ttl`)	Yes	Yes (…broker)
Num messages out	Number of payload messages that have flowed out of the broker since it was last started.	Yes (`ttl`)	Yes	Yes (…broker)
Message bytes out	Number of payload message bytes that have flowed out of the broker since it was last started	Yes (`ttl`)	Yes	Yes (…broker)
Num packets out	Number of packets that have flowed out of the broker since it was last started; includes both payload messages and control messages	Yes (`ttl`)	Yes	Yes (…broker)
Packet bytes out	Number of packet bytes that have flowed out of the broker since it was last started; includes both payload messages and control messages	Yes (`ttl`)	Yes	Yes (…broker)
Rate messages in	Current rate of flow of payload messages into the broker	Yes (`rts`)	Yes	No
Rate message bytes in	Current rate of flow of payload message bytes into the broker	Yes (`rts`)	Yes	No
Rate packets in	Current rate of flow of packets into the broker; includes both payload messages and control messages	Yes (`rts`)	Yes	No
Rate packet bytes in	Current rate of flow of packet bytes into the broker; includes both payload messages and control messages	Yes (`rts`)	Yes	No
Rate messages out	Current rate of flow of payload messages out of the broker	Yes (`rts`)	Yes	No
Rate message bytes out	Current rate of flow of payload message bytes out of the broker	Yes (`rts`)	Yes	No
Rate packets out	Current rate of flow of packets out of the broker; includes both payload messages and control messages	Yes (`rts`)	Yes	No
Rate packet bytes out	Current rate of flow of packet bytes out of the broker; includes both payload messages and control messages	Yes (`rts`)	Yes	No

Table 18-2 Brokerwide Metrics *(Continued)*

Metric Quantity	Description	imqcmd `metrics bkr` (*metricType*)	Log File	Metrics Message (*metrics topic*)[1]
Destinations Data				
Num destinations	Number of physical destination in the broker	No	No	Yes (…broker)

1. For metrics topic destination names, see Table 10-7 on page 202.

Connection Service Metrics

Table 18-3 lists and describes the metrics data the broker reports for individual connection services. It also shows which of the data can be obtained using the different metrics monitoring tools.

Table 18-3 Connection Service Metrics

Metric Quantity	Description	imqcmd `metrics svc` (*metricType*)	Log File	Metrics Message (*metrics topic*)
Connection Data				
Num connections	Number of currently open connections	Yes (cxn) Also `query svc`	No	No
Num threads	Number of threads currently in use	Yes (cxn) Also `query svc`	No	No
Min threads	Number of threads, which once reached, are maintained in the thread pool for use by connection services, totaled across all connection services	Yes (cxn)	No	No
Max threads	Number of threads, beyond which no new threads are added to the thread pool for use by connection services, totaled across all connection services	Yes (cxn)	No	No
Message Flow Data				
Num messages in	Number of payload messages that have flowed into the connection service since the broker was last started	Yes (`ttl`)	No	No

Table 18-3 Connection Service Metrics *(Continued)*

Metric Quantity	**Description**	**imqcmd `metrics svc` (*metricType*)**	**Log File**	**Metrics Message (*metrics topic*)**
Message bytes in	Number of payload message bytes that have flowed into the connection service since the broker was last started	Yes (`ttl`)	No	No
Num packets in	Number of packets that have flowed into the connection service since the broker was last started; includes both payload messages and control messages	Yes (`ttl`)	No	No
Packet bytes in	Number packet bytes that have flowed into the connection service since the broker was last started; includes both payload messages and control messages	Yes (`ttl`)	No	No
Num messages out	Number of payload messages that have flowed out of the connection service since the broker was last started	Yes (`ttl`)	No	No
Message bytes out	Number of payload message bytes that have flowed out of the connection service since the broker was last started	Yes (`ttl`)	No	No
Num packets out	Number of packets that have flowed out of the connection service since the broker was last started; includes both payload messages and control messages	Yes (`ttl`)	No	No
Packet bytes out	Number packet bytes that have flowed out of the connection service since the broker was last started; includes both payload messages and control messages	Yes (`ttl`)	No	No
Rate messages in	Current rate of flow of payload messages into the broker through the connection service	Yes (`rts`)	No	No
Rate message bytes in	Current rate of flow of payload message bytes into the connection service	Yes (`rts`)	No	No
Rate packets in	Current rate of flow of packets into the connection service; includes both payload messages and control messages	Yes (`rts`)	No	No
Rate packet bytes in	Current rate of flow of packet bytes into the connection service; includes both payload messages and control messages	Yes (`rts`)	No	No
Rate messages out	Current rate of flow of payload messages out of the connection service	Yes (`rts`)	No	No

Table 18-3 Connection Service Metrics *(Continued)*

Metric Quantity	Description	imqcmd `metrics svc` (*metricType*)	Log File	Metrics Message (*metrics topic*)
Rate message bytes out	Current rate of flow of payload message bytes out of the connection service	Yes (`rts`)	No	No
Rate packets out	Current rate of flow of packets out of the connection service; includes both payload messages and control messages	Yes (`rts`)	No	No
Rate packet bytes out	Current rate of flow of packet bytes out of the connection service; includes both payload messages and control messages	Yes (`rts`)	No	No

Destination Metrics

Table 18-4 lists and describes the metrics data the broker reports for individual destinations. It also shows which of the data can be obtained using the different metrics monitoring tools.

Table 18-4 Destination Metrics

Metric Quantity	Description	imqcmd `metrics dst` (*metricType*)	Log File	Metrics Message (*metrics topic*)[1]
Consumer Data				
Num consumers	Current number of consumers For a topic, this value includes non-durable subscriptions, active durable subscriptions, and inactive durable subscriptions. For a queue, this value includes active consumers and backup consumers.	Yes (con)	No	Yes (…*destName*)
Avg num consumers	Average number of consumers since the broker was last started	Yes (con)	No	Yes (…*destName*)
Peak num consumers	Peak number of consumers since the broker was last started	Yes (con)	No	Yes (…*destName*)
Num active consumers	Current number of active consumers	Yes (con)	No	Yes (…*destName*)
Avg num active consumers	Average number of active consumers since the broker was last started	Yes (con)	No	Yes (…*destName*)

Table 18-4 Destination Metrics *(Continued)*

Metric Quantity	Description	imqcmd `metrics dst` (*metricType*)	Log File	Metrics Message (*metrics topic*)[1]
Peak num active consumers	Peak number of active consumers since the broker was last started	Yes (`con`)	No	Yes (…*destName*)
Num backup consumers	Current number of backup consumers (applies only to queues)	Yes (`con`)	No	Yes (…*destName*)
Avg num backup consumers	Average number of backup consumers since the broker was last started (applies only to queues)	Yes (`con`)	No	Yes (…*destName*)
Peak num backup consumers	Peak number of backup consumers since the broker was last started (applies only to queues)	Yes (`con`)	No	Yes (…*destName*)
Stored Messages Data				
Num messages	Number of payload messages currently stored in destination memory and persistent store	Yes (`con`) (`ttl`) (`rts`) Also `query dst`	No	Yes (…*destName*)
Avg num messages	Average number of payload messages stored in destination memory and persistent store since the broker was last started	Yes (`con`) (`ttl`) (`rts`)	No	Yes (…*destName*)
Peak num messages	Peak number of payload messages stored in destination memory and persistent store since the broker was last started	Yes (`con`) (`ttl`) (`rts`)	No	Yes (…*destName*)
Total message bytes	Number of payload message bytes currently stored in destination memory and persistent store	Yes (`ttl`) (`rts`) Also `query dst`	No	Yes (…*destName*)
Avg total message bytes	Average number of payload message bytes stored in destination memory and persistent store since the broker was last started	Yes (`ttl`) (`rts`)	No	Yes (…*destName*)
Peak total message bytes	Peak number of payload message bytes stored in destination memory and persistent store since the broker was last started	Yes (`ttl`) (`rts`)	No	Yes (…*destName*)

Table 18-4 Destination Metrics *(Continued)*

Metric Quantity	Description	imqcmd `metrics dst` (*metricType*)	Log File	Metrics Message (*metrics topic*)[1]
Peak message bytes	Peak number of payload message bytes in a single message received by the destination since the broker was last started	Yes (`ttl`) (`rts`)	No	Yes (…*destName*)
Message Flow Data				
Num messages in	Number of payload messages that have flowed into this destination since the broker was last started	Yes (`ttl`)	No	Yes (…*destName*)
Msg bytes in	Number of payload message bytes that have flowed into this destination since the broker was last started	Yes (`ttl`)	No	Yes (…*destName*)
Num messages out	Number of payload messages that have flowed out of this destination since the broker was last started	Yes (`ttl`)	No	Yes (…*destName*)
Msg bytes out	Number of payload message bytes that have flowed out of this destination since the broker was last started	Yes (`ttl`)	No	Yes (…*destName*)
Rate num messages in	Current rate of flow of payload messages into the destination	Yes (`rts`)	No	No
Rate num messages out	Current rate of flow of payload messages out of the destinatio	Yes (`rts`)	No	No
Rate msg bytes in	Current rate of flow of payload message bytes into the destination	Yes (`rts`)	No	No
Rate Msg bytes out	Current rate of flow of payload message bytes out of the destination	Yes (`rts`)	No	No
Disk Utilization Data				
Disk reserved	Disk space, in bytes, used by all message records (active and free) in the destination file-based store	Yes (`dsk`)	No	Yes (…*destName*)
Disk used	Disk space, in bytes, used by active message records in destination file-based store	Yes (`dsk`)	No	Yes (…*destName*)
Disk utilization ratio	Ratio of used disk space to reserved disk space. The higher the ratio, the more the disk space is being used to hold active messages	Yes (`dsk`)	No	Yes (…*destName*)

1. For metrics topic destination names, see Table 10-7 on page 202.

Part IV

Appendixes

Appendix A

Platform-Specific Locations of Message Queue Data

Sun Java System Message Queue data is stored in different locations on different operating system platforms. The tables that follow show the location of various types of Message Queue data on the following platforms:

- "Solaris" on page 348
- "Linux" on page 349
- "Windows" on page 350

In the tables, *instanceName* denotes the name of the broker instance with which the data is associated.

Solaris

Table A-1 shows the location of Message Queue data on the Solaris operating system. If you are using Message Queue on Solaris with the standalone version of Sun Java System Application Server, the directory structure is like that described under "Windows" on page 350.

Table A-1 Message Queue Data Locations on Solaris Platform

Data Category	**Location**
Broker instance configuration properties	`/var/imq/instances/`*instanceName*`/props/config.properties`
Broker configuration file templates	`/usr/share/lib/imq/props/broker/`
Persistent store (messages, destinations, durable subscriptions, transactions)	`/var/imq/instances/`*instanceName*`/fs350` or a JDBC-accessible data store
Broker instance log file directory (default location)	`/var/imq/instances/`*instanceName*`/log/`
Administered objects (object store)	Local directory of your choice or an LDAP server
Security: user repository	`/var/imq/instances/`*instanceName*`/etc/passwd` or an LDAP server
Security: access control file (default location)	`/var/imq/instances/`*instanceName*`/etc/accesscontrol.properties`
Security: password file directory (default location)	`/var/imq/instances/`*instanceName*`/etc/`
Security: example password file	`/etc/imq/passfile.sample`
Security: broker's key store file location	`/etc/imq/`
JavaDoc API documentation	`/usr/share/javadoc/imq/index.html`
Example applications and configurations	`/usr/demo/imq/`
Java archive (`.jar`), Web archive (`.war`), and Resource Adapter archive (`.rar`) files	`/usr/share/lib/`

Linux

Table A-2 shows the location of Message Queue data on the Linux operating system.

Table A-2 Message Queue Data Locations on Linux Platform

Data Category	Location
Broker instance configuration properties	`/var/opt/sun/mq/instances/`*instanceName*`/props/config.properties`
Broker configuration file templates	`/opt/sun/mq/private/share/lib/props/`
Persistent store (messages, destinations, durable subscriptions, transactions)	`/var/opt/sun/mq/instances/`*instanceName*`/fs350/` or a JDBC-accessible data store
Broker instance log file directory (default location)	`/var/opt/sun/mq/instances/`*instanceName*`/log/`
Administered objects (object store)	Local directory of your choice or an LDAP server
Security: user repository	`/var/opt/sun/mq/instances/`*instanceName*`/etc/passwd` or an LDAP server
Security: access control file (default location)	`/var/opt/sun/mq/instances/`*instanceName*`/etc/accesscontrol.properties`
Security: password file directory (default location)	`/var/opt/sun/mq/instances/`*instanceName*`/etc/`
Security: example password file	`/etc/opt/sun/mq/passfile.sample`
Security: broker's key store file location	`/etc/opt/sun/mq/`
JavaDoc API documentation	`/opt/sun/mq/javadoc/index.html`
Example applications and configurations	`/opt/sun/mq/examples/`
Java archive (`.jar`), Web archive (`.war`), and Resource Adapter archive (`.rar`) files	`/opt/sun/mq/share/lib/`
Shared library (`.so`) files	`/opt/sun/mq/lib/`

Windows

Table A-3 shows the location of Message Queue data on the Windows operating system. The table also applies to the Solaris platform when Message Queue is bundled with the standalone version of Sun Java System Application Server. That version of Application Server is bundled with neither Solaris nor Sun Java Enterprise System. Use the pathnames in Table A-3, but change the direction of the slash characters from the Windows backslash (\) to the Solaris forward slash (/). See "Directory Variable Conventions" on page 23 for definitions of the `IMQ_HOME` and `IMQ_VARHOME` directory variables.

Table A-3 Message Queue Data Locations on Windows Platform

Data Category	Location
Broker instance configuration properties	`IMQ_VARHOME\instances\`*instanceName*`\props\config.properties`
Broker configuration file templates	`IMQ_HOME\lib\props\broker\`
Persistent store (messages, destinations, durable subscriptions, transactions)	`IMQ_VARHOME\instances\`*instanceName*`\fs350\` or a JDBC-accessible data store
Broker instance log file directory (default location)	`IMQ_VARHOME\instances\`*instanceName*`\log\`
Administered objects (object store)	Local directory of your choice or an LDAP server
Security: user repository	`IMQ_VARHOME\instances\`*instanceName*`\etc\passwd` or an LDAP server
Security: access control file (default location)	`IMQ_VARHOME\instances\`*instanceName*`\etc\accesscontrol.properties`
Security: password file directory (default location)	`IMQ_HOME\etc\`
Security: example password file	`IMQ_HOME\etc\passfile.sample`
Security: broker's key store file location	`IMQ_HOME\etc\`
JavaDoc API documentation	`IMQ_HOME\javadoc\index.html`
Example applications and configurations	`IMQ_HOME\demo\`
Java archive (`.jar`), Web archive (`.war`), and Resource Adapter archive (`.rar`) files	`IMQ_HOME\lib\`

Appendix B

Stability of Message Queue Interfaces

Sun Java System Message Queue uses many interfaces that can help administrators automate tasks. This appendix classifies the interfaces according to their stability. The more stable an interface is, the less likely it is to change in subsequent versions of the product.

Any interface that is not listed in this appendix is private and not for customer use.

Table B-1 describes the stability classification scheme.

Table B-1 Interface Stability Classification Scheme

Classification	Description
Private	Not for direct use by customers. May change or be removed in any release.
Evolving	For use by customers. Subject to incompatible change at a major (e.g. 3.0, 4.0) or minor (e.g. 3.1, 3.2) release. The changes will be made carefully and slowly. Reasonable efforts will be made to ensure that all changes are compatible but that is not guaranteed.
Stable	For use by customers. Subject to incompatible change at a major (e.g 3.0, 4.0) release only.
Standard	For use by customers. These interfaces are defined by a formal standard, and controlled by a standards organization. Incompatible changes to these interfaces are rare.
Unstable	For use by customers. Subject to incompatible change at a major (e.g. 3.0, 4.0) or minor (e.g. 3.1, 3.2) release. Customers are advised that these interfaces may be removed or changed substantially and in an incompatible way in a future release. It is recommended that customers not create explicit dependencies on unstable interfaces.

Table B-2 lists the interfaces and their classifications.

Table B-2 Stability of Message Queue Interfaces

Interface	Classification
Command Line Interfaces	
`imqbrokerd` command line interface	Evolving
`imqadmin` command line interface	Unstable
`imqcmd` command line interface	Evolving
`imqdbmgr` command line interface	Unstable
`imqkeytool` command line interface	Evolving
`imqobjmgr` command line interface	Evolving
`imqusermgr` command line interface	Unstable
Output from `imqbrokerd`, `imqadmin`, `imqcmd`, `imqdbmgr`, `imqkeytool`, `imqobjmgr`, `imqusermgr`	Unstable
Commands	
`imqobjmgr` command file	Evolving
`imqbrokerd` command	Stable
`imqadmin` command	Unstable
`imqcmd` command	Stable
`imqdbmgr` command	Unstable
`imqkeytool` command	Stable
`imqobjmgr` command	Stable
`imqusermgr` command	Unstable
APIs	
JMS API (`javax.jms`)	Standard
JAXM API (`javax.xml`)	Standard
C-API	Evolving
C-API environment variables	Unstable
Message-based monitoring API	Evolving
Administered Object API (`com.sun.messaging`)	Evolving
.jar and .war Files	
`imq.jar` location and name	Stable
`jms.jar` location and name	Evolving

Table B-2 Stability of Message Queue Interfaces *(Continued)*

Interface	Classification
imqbroker.jar location and name	Private
imqutil.jar location and name	Private
imqadmin.jar location and name	Private
imqservlet.jar location and name	Evolving
imqhttp.war location and name	Evolving
imqhttps.war location and name	Evolving
imqjmsra.rar location and name	Evolving
imqxm.jar location and name	Evolving
jaxm-api.jar location and name	Evolving
saaj-api.jar location and name	Evolving
saaj-impl.jar location and name	Evolving
activation.jar location and name	Evolving
mail.jar location and name	Evolving
dom4j.jar location and name	Private
fscontext.jar location and name	Unstable
Files	
Broker log file location and content format	Unstable
password file	Unstable
accesscontrol.properties file	Unstable
System Destinations	
mq.sys.dmq destination	Stable
mq.metrics.* destinations	Evolving
Configuration Properties	
Message Queue JMS Resource Adapter configuration properties	Evolving
Message Queue JMS Resource Adapter JavaBean and ActivationSpec configuration properties	Evolving

Table B-2 Stability of Message Queue Interfaces *(Continued)*

Interface	Classification
Message Properties and Formats	
Dead message queue message property, `JMSXDeliveryCount`	Standard
Dead message queue message properties, `JMS_SUN_*`	Evolving
Message Queue client message properties: `JMS_SUN_*`	Evolving
JMS message format for metrics or monitoring messages	Evolving
Miscellaneous	
Message Queue JMS Resource Adapter package, `com.sun.messaging.jms.ra`	Evolving
JDBC schema for storage of persistent messages	Evolving

Appendix C

HTTP/HTTPS Support

Message Queue, Enterprise Edition includes support for a Java client to communicate with the broker by means of an HTTP or secure HTTP (HTTPS) transport, rather than a direct TCP connection. HTTP / HTTPS support is not available for C clients.

This appendix describes the architecture used to enable this support and explains the setup work needed to allow clients to use HTTP-based connections for Message Queue messaging. It has the following sections:

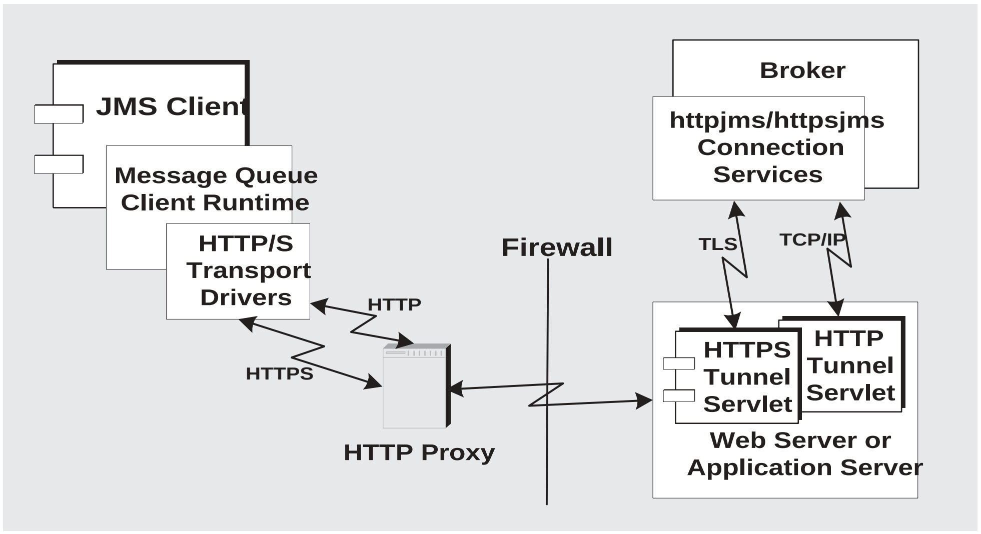
HTTP/HTTPS Support Architecture

Message Queue messaging can run on top of HTTP / HTTPS connections. Because HTTP / HTTPS connections are normally allowed through firewalls, this allows client applications to be separated from a broker by a firewall.

Figure C-1 on page 356 shows the main components involved in providing HTTP / HTTPS support.

- On the client side, an HTTP or HTTPS transport driver encapsulates the Message Queue message into an HTTP request and makes sure that these requests are sent to the Web server / application server in the correct sequence.
- The client can use an HTTP proxy server to communicate with the broker if necessary. The proxy's address is specified using command line options when starting the client. See "Using an HTTP Proxy" on page 365 for more information.
- An HTTP or HTTPS tunnel servlet (both bundled with Message Queue) is loaded in a Web server / application server and used to pull payload messages out of client HTTP requests before forwarding them to the broker. The HTTP / HTTPS tunnel servlet also sends broker messages back to the client in response to HTTP requests made by the client. A single HTTP / HTTPS tunnel servlet can be used to access multiple brokers.

Figure C-1 HTTP / HTTPS Support Architecture



- On the broker side, the httpjms or httpsjms connection service unwraps and de-multiplexes incoming messages from the corresponding tunnel servlet.
- If the Web server / application server fails and is restarted, all connections are restored and there is no effect on clients. If the broker fails and is restarted, an exception is thrown and clients must re-establish their connections. In the unlikely case that both the Web server / application server and the broker fail, and the broker is not restarted, the Web server / application server will restore client connections and continue waiting for a broker connection— without notifying clients. To avoid this situation, always restart the broker.

As you can see from Figure C-1, the architecture for HTTP and HTTPS support are very similar. The main difference is that, in the case of HTTPS (httpsjms connection service), the tunnel servlet has a secure connection to both the client application and broker.

The secure connection to the broker is provided through an SSL-enabled tunnel servlet—Message Queue's HTTPS tunnel servlet—which passes a self-signed certificate to any broker requesting a connection. The certificate is used by the broker to set up an encrypted connection to the HTTPS tunnel servlet. Once this connection is established, a secure connection between a client application and the tunnel servlet can be negotiated by the client application and the Web server / application server.

Enabling HTTP Support

The following sections describe the steps you need to take to enable HTTP support.

➤ To Enable HTTP Support

1. Deploy the HTTP tunnel servlet. You can deploy the HTTP tunnerl servlet on the following:
   - Sun Java System Web Server
   - Sun Java System Application Server
2. Configure the broker's httpjms connection service and start the broker.
3. Configure an HTTP connection.

Step 1. Deploy the HTTP Tunnel Servlet

You can deploy the HTTP tunnel servlet as a Web archive (`.war`) file on a Sun Java System Web Server or Sun Java System Application Server.

Deploying the HTTP tunnel servlet as a `.war` file consists of using the deployment mechanism provided by the Web server/application server. The HTTP tunnel servlet `.war` file (`imqhttp.war`) is located in the directory containing `.jar`, `.war`, and `.rar` files, and depends on your operating system (see Appendix A, "Platform-Specific Locations of Message Queue Data").

The `.war` file includes a deployment descriptor that contains the basic configuration information needed by the Web server/application server to load and run the servlet. Depending on the Web server/application server, you might also need to specify the context root portion of the servlet's URL.

Deploying as a Web Archive File

For deployment on a Sun Java System Web Server, see "Deploying the HTTP Tunnel Servlet on Sun Java System Web Server" on page 358.

For deployment on a Sun Java System Application Server, see "Deploying the HTTP Tunnel Servlet on Sun Java System Application Server" on page 360.

Deploying the HTTP Tunnel Servlet on Sun Java System Web Server

The instructions below refer to deployment on Sun Java System Web Server. You can verify successful HTTP tunnel servlet deployment by accessing the servlet URL using a Web browser. It should display status information.

➤ To Deploy the http Tunnel Servlet as a .war File

1. In the browser-based administration GUI, select the Virtual Server Class tab and select Manage Classes.
2. Select the appropriate virtual server class name (for example, `defaultClass`) and click the Manage button.
3. Select Manage Virtual Servers.
4. Select an appropriate virtual server name and click the Manage button.

5. Select the Web Applications tab.

6. Click on Deploy Web Application.

7. Select the appropriate values for the WAR File On and WAR File Path fields so as to point to the `imqhttp.war` file, which can be found in a directory that depends on your operating system (see Appendix A, "Platform-Specific Locations of Message Queue Data").

8. Enter a path in the Application URI field.

   The Application URI field value is the /*contextRoot* portion of the tunnel servlet URL:

   `http://`*hostName*`:`*portNumber*`/`*contextRoot*`/tunnel`

   For example, if you set the *contextRoot* to `imq`, the Application URI field would be:

   ```
   /imq
   ```

9. Enter the installation directory path (typically somewhere under the Sun Java System Web Server installation root) where the servlet should be deployed.

10. Click OK.

11. Restart the Web server instance.

The servlet is now available at the following address:

`http://`*hostName*`:`*portNumber*`/`*contextRoot*`/tunnel`

Clients can now use this URL to connect to the message service using an HTTP connection.

Disabling a Server Access Log

You do not have to disable the server access log, but you will obtain better performance if you do.

➤ To Disable the Server Access Log

1. Select the Status tab.

2. Choose the Log Preferences Page.

Use the Log client accesses control to disable logging.

Deploying the HTTP Tunnel Servlet on Sun Java System Application Server

This section describes how you deploy the HTTP tunnel servlet as a `.war` file on the Sun Java System Application Server, and then configure the tunnel servlet to accept connections from a Message Queue broker.

Two steps are required:

- Deploy the HTTP tunnel servlet using the Application Server deployment tool.
- Modify the application server instance's `server.policy` file.

Using the Deployment Tool

➤ To Deploy the HTTP Tunnel Servlet in an Application Server Environment

1. In the Web-based administration GUI, choose

   App Server > Instances > server1 > Applications > Web Applications.

2. Click the Deploy button.

3. In the File Path: text field, enter the location of the HTTP tunnel servlet `.war` file (`imqhttp.war`), and click OK.

   The location of the `imqhttp.war` file depends on your operating system (see Appendix A, "Platform-Specific Locations of Message Queue Data").

4. Set the value for the Context Root text field, and click OK.

   The Context Root field value is the */contextRoot* portion of the tunnel servlet URL:

   `http://`*hostName*`:`*portNumber*`/`*contextRoot*`/tunnel`

   For example, you could set the Context Root field to `/imq`.

   The confirmation screen that appears confirms that the tunnel servlet has been successfully deployed, is enabled by default, and—in this case—is located at:

   ```
   /var/opt/SUNWappserver8/domains/domain1/server1/applications/
   j2ee-modules/imqhttp_1
   ```

The servlet is now available at the following URL:

`http://`*hostName*`:`*portNumber*`/`*contextRoot*`/tunnel`

Clients can now use this URL to connect to the message service using an HTTP connection.

Modifying the server.policy File

The Application Server enforces a set of default security policies that, unless modified, would prevent the HTTP tunnel servlet from accepting connections from the Message Queue broker.

Each application server instance has a file that contains its security policies, or rules. For example, the location of this file for the server1 instance on Solaris is:

```
/var/opt/SUNWappserver8/domains/domain1/server1/config/
server.policy
```

To configure the tunnel servlet to accept connections from the Message Queue broker, an additional entry is required in this file.

➤ To Modify the Application Server's server.policy File

1. Open the `server.policy` file.

2. Add the following entry:

```
grant codeBase
"file:/var/opt/SUNWappserver8/domains/domain1/server1/
             applications/j2ee-modules/imqhttp_1/-”
{
    permission java.net.SocketPermission "*",
             “connect,accept,resolve";
};
```

Step 2. Configure the httpjms Connection Service

HTTP support is not activated for a broker by default, so you need to reconfigure the broker to activate the httpjms connection service. Once reconfigured, the broker can be started as outlined in "Starting Brokers" on page 66.

➤ To Activate the httpjms Connection Service

1. Open the broker's instance configuration file.

   The instance configuration file is stored in a directory identified by the name of the broker instance (*instanceName*) with which the configuration file is associated (see Appendix A, "Platform-Specific Locations of Message Queue Data"):

   ```
   .../instances/instanceName/props/config.properties
   ```

2. Add the `httpjms` value to the `imq.service.activelist` property:

   ```
   imq.service.activelist=jms,admin,httpjms
   ```

At startup, the broker looks for a Web server/application server and HTTP tunnel servlet running on its host machine. To access a remote tunnel servlet, however, you can reconfigure the `servletHost` and `servletPort` connection service properties.

You can also reconfigure the `pullPeriod` property to improve performance. The httpjms connection service configuration properties are detailed in Table C-1.

Table C-1 `httpjms` Connection Service Properties

Property	Description
`imq.httpjms.http.servletHost`	Change this value, if necessary, to specify the name of the host (hostname or IP address) on which the HTTP tunnel servlet is running. (This can be a remote host or a specific hostname on a local host.) Default: `localhost`.
`imq.httpjms.http.servletPort`	Change this value to specify the port number that the broker uses to access the HTTP tunnel servlet. (If the default port is changed on the Web server, you must change this property accordingly.) Default: 7675.

Table C-1 `httpjms` Connection Service Properties *(Continued)*

Property	Description
`imq.httpjms.http.pullPeriod`	Specifies the interval, in seconds, between HTTP requests made by a client runtime to pull messages from the broker. (Note that this property is set on the broker and propagates to the client runtime.) If the value is zero or negative, the client keeps one HTTP request pending at all times, ready to pull messages as fast as possible. With a large number of clients, this can be a heavy drain on Web/application server resources and the server may become unresponsive. In such cases, you should set the `pullPeriod` property to a positive number of seconds. This sets the time the client's HTTP transport driver waits before making subsequent pull requests. Setting the value to a positive number conserves Web/application server resources at the expense of the response times observed by clients. Default: `-1`.
`imq.httpjms.http.connectionTimeout`	Specifies the time, in seconds, that the client runtime waits for a response from the HTTP tunnel servlet before throwing an exception. (Note that this property is set on the broker and propagates to the client runtime.) This property also specifies the time the broker waits after communicating with the HTTP tunnel servlet before freeing up a connection. A timeout is necessary in this case because the broker and the tunnel servlet have no way of knowing if a client that is accessing the HTTP servlet has terminated abnormally. Default: `60`.

Step 3. Configure an HTTP Connection

A client application must use an appropriately configured connection factory administered object to make an HTTP connection to a broker. This section discusses HTTP connection configuration issues.

Configuring the Connection Factory

To enable HTTP support, you need to set the connection factory's `imqAddressList` attribute to the HTTP tunnel servlet URL. The general syntax of the HTTP tunnel servlet URL is the following:

```
http://hostName:portNumber/contextRoot/tunnel
```

where *hostName*:*portNumber* is the name and port of the Web server/application server hosting the HTTP tunnel servlet and *contextRoot* is a path set when deploying the tunnel servlet on the Web server/application server.

For more information on connection factory attributes in general, and the `imqAddressList` attribute in particular, see the *Message Queue Developer's Guide for Java Clients*.

You can set connection factory attributes in one of the following ways:

- Using the `-o` option to the `imqobjmgr` command that creates the connection factory administered object (see "Adding a Connection Factory" on page 174), or set the attribute when creating the connection factory administered object using the Administration Console (`imqadmin`).
- Using the `-D` option to the command that launches the client (see the *Message Queue Developer's Guide for Java Clients*).
- Using an API call to set the attributes of a connection factory after you create it programmatically in client code (see the *Message Queue Developer's Guide for Java Clients*).

Using a Single Servlet to Access Multiple Brokers

You do not need to configure multiple Web servers/application servers and servlet instances if you are running multiple brokers. You can share a single Web server/application server and HTTP tunnel servlet instance among concurrently running brokers. If multiple broker instances are sharing a single tunnel servlet, you must configure the `imqAddressList` connection factory attribute as shown below:

```
http://hostName:portNumber/contextRoot/tunnel?ServerName=bkrHostName:instanceName
```

Where *bkrHostName* is the broker instance host name and *instanceName* is the name of the specific broker instance you want your client to access.

To check that you have entered the correct strings for *bkrHostName* and *instanceName*, generate a status report for the HTTP tunnel servlet by accessing the servlet URL from a browser. The report lists all brokers being accessed by the servlet:

```
HTTP tunnel servlet ready.
Servlet Start Time : Thu May 30 01:08:18 PDT 2005
Accepting TCP connections from brokers on port : 7675
Total available brokers = 2
Broker List :
   jpgserv:broker2
   cochin:broker1
```

Using an HTTP Proxy

If you are using an HTTP proxy to access the HTTP tunnel servlet:

- Set `http.proxyHost` system property to the proxy server host name.
- Set `http.proxyPort` system property to the proxy server port number.

You can set these properties using the `-D` option to the command that launches the client application.

Enabling HTTPS Support

The following sections describe the steps to enable HTTPS support. They are similar to those in "Enabling HTTP Support" on page 357 with the addition of steps needed to generate and access SSL certificates.

➤ **To Enable HTTPS Support**

1. Generate a self-signed certificate for the HTTPS tunnel servlet.
2. Modify the HTTP tunnel servlet `.war` file's deployment descriptor to:
   - point to the location where you have placed the certificate key store
   - specify the certificate key store password
3. Deploy the HTTP tunnel servlet. You can deploy the HTTP tunnel servlet on the following:
   - Sun Java System Web Server
   - Sun Java System Application Server
4. Configure the broker's httpsjms connection service and start the broker.
5. Configure an HTTPS connection.

Each of these steps is discussed in more detail in the sections that follow.

Step 1. Generating a Self-signed Certificate for the HTTPS Tunnel Servlet

Message Queue's SSL support is oriented toward securing on-the-wire data with the assumption that the client is communicating with a known and trusted server. Therefore, SSL is implemented using only self-signed server certificates. In the httpsjms connection service architecture, the HTTPS tunnel servlet plays the role of server to both broker and application client.

Run the `keytool` utility to generate a self-signed certificate for the tunnel servlet. Enter the following at the command prompt:

```
JRE_HOME/bin/keytool -servlet keyStoreLocation
```

The utility will prompt you for the information it needs. (On Unix systems you may need to run `keytool` as the superuser (root) in order to have permission to create the key store.)

First, `keytool` prompts you for a key store password, and then it prompts you for some organizational information, and then it prompts you for confirmation. After it receives the confirmation, it pauses while it generates a key pair. It then asks you for a password to lock the particular key pair (key password); you should enter Return in response to this prompt: this makes the key password the same as the key store password.

NOTE Remember the password you provide—you must provide this password later to the tunnel servlet so it can open the key store.

The JDK `keytool` utility generates a self-signed certificate and places it in Message Queue's key store file located as specified in the *keyStoreLocation* argument.

NOTE The HTTPS tunnel servlet must be able to see the key store. Make sure you move/copy the generated key store located in *keyStoreLocation* to a location accessible by the HTTPS tunnel servlet (see "Step 3. Deploying the HTTPS Tunnel Servlet" on page 368).

Step 2. Modifying the HTTP Tunnel Servlet .war File's Descriptor File

The HTTP Tunnel Servlet's `.war` file includes a deployment descriptor that contains the basic configuration information needed by the Web server/application server to load and run the servlet.

The deployment descriptor of the `imqhttps.war` file cannot know where you have placed the key store file needed by the tunnel servlet. This requires you to edit the tunnel servlet's deployment descriptor (an XML file) to specify the key store location and password before deploying the `imqhttps.war` file.

➤ To Modify the HTTPS Tunnel Servlet .war File

1. Copy the `.war` file to a temporary directory.

   `cp /usr/share/lib/imq/imqhttps.war /tmp` (Solaris)

   `cp /opt/sun/mq/share/lib/imqhttps.war /tmp` (Linux)

   `cp IMQ_HOME/lib/imqhttps.war /tmp` (Windows)

2. Make the temporary directory your current directory.

```
$ cd /tmp
```

3. Extract the contents of the `.war` file.

```
$ jar xvf imqhttps.war
```

4. List the `.war` file's deployment descriptor.

```
$ ls -l WEB-INF/web.xml
```

5. Edit the `web.xml` file to provide correct values for the `keystoreLocation` and `keystorePassword` arguments (as well as `servletPort` and `servletHost` arguments, if necessary).

6. Re-assemble the contents of the `.war` file.

   $ jar uvf imqhttps.war WEB-INF/web.xml

You are now ready to use the modified `imqhttps.war` file to deploy the HTTPS tunnel servlet. (If you are concerned about exposure of the key store password, you can use file system permissions to restrict access to the `imqhttps.war` file.)

Step 3. Deploying the HTTPS Tunnel Servlet

You can deploy the HTTP tunnel servlet as a Web archive (WAR) file on a Sun Java System Web Server or Sun Java System Application Server.

Deploying the HTTPS tunnel servlet as a `.war` file consists of using the deployment mechanism provided by the Web server/application server. The HTTPS tunnel servlet `.war` file (`imqhttps.war`) is located in a directory that depends on your operating system (see Appendix A, "Platform-Specific Locations of Message Queue Data").

You should make sure that encryption is activated for the Web server, enabling end-to-end secure communication between the client and broker.

Deploying as a Web Archive File

For deployment on a Sun Java System Web Server, see "Deploying the HTTPS Tunnel Servlet on Sun Java System Web Server" on page 369.

For deployment on a Sun Java System Application Server, see "Deploying the HTTPS Tunnel Servlet on Sun Java System Application Server" on page 370.

Deploying the HTTPS Tunnel Servlet on Sun Java System Web Server

This section describes how you deploy the HTTPS tunnel servlet as a `.war` file on the Sun Java System Web Server. You can verify successful HTTPS tunnel servlet deployment by accessing the servlet URL using a Web browser. It should display status information.

Before deploying the HTTPS tunnel servlet, make sure that JSSE `.jar` files are included in the Web server's classpath. The simplest way to do this is to copy the files `jsse.jar`, `jnet.jar`, and `jcert.jar` to `WebServer_TOPDIR/bin/https/jre/lib/ext`.

➤ To Deploy the https Tunnel Servlet as a .war File

1. In the browser-based administration GUI, select the Virtual Server Class tab. Click Manage Classes.
2. Select the appropriate virtual server class name (for example, `defaultClass`) and click the Manage button.
3. Select Manage Virtual Servers.
4. Select an appropriate virtual server name and click the Manage button.
5. Select the Web Applications tab.
6. Click on Deploy Web Application.
7. Select the appropriate values for the WAR File On and WAR File Path fields so as to point to the modified `imqhttps.war` file (see "To Modify the HTTPS Tunnel Servlet .war File" on page 367.)
8. Enter a path in the Application URI field.

   The Application URI field value is the /*contextRoot* portion of the tunnel servlet URL:

   `https://`*hostName*`:`*portNumber*`/`*contextRoot*`/tunnel`

   For example, if you set the *contextRoot* to `imq`, the Application URI field would be:

   ```
   /imq
   ```

9. Enter the installation directory path (typically somewhere under the Sun Java System Web Server installation root) where the servlet should be deployed.
10. Click OK.
11. Restart the Web server instance.

The servlet is now available at the following URL:

`https://`*hostName*`:`*portNumber*`/imq/tunnel`

Clients can now use this URL to connect to the message service using a secure HTTPS connection.

Disabling a Server Access Log

You do not have to disable the server access log, but you will obtain better performance if you do.

➤ To Disable the Server Access Log

1. Select the Status tab.
2. Choose the Log Preferences Page.

Use the Log client accesses control to disable logging.

Deploying the HTTPS Tunnel Servlet on Sun Java System Application Server

This section describes how you deploy the HTTPS tunnel servlet as a `.war` file on the Sun Java System Application Server.

Two steps are required:

- Deploy the HTTPS tunnel servlet using the Application Server deployment tool.
- Modify the application server instance's `server.policy` file.

Using the Deployment Tool

➤ To Deploy the HTTPS Tunnel Servlet in an Application Server Environment

1. In the Web-based administration GUI, choose

   App Server > Instances > server1 > Applications > Web Applications

2. Click the Deploy button.
3. In the File Path: text field, enter the location of the HTTPS tunnel servlet `.war` file (`imqhttps.war`), and click OK.

   The location of the `imqhttps.war` file depends on your operating system (see Appendix A, "Platform-Specific Locations of Message Queue Data").

4. Set the value for the Context Root text field, and click OK.

The Context Root field value is the */contextRoot* portion of the tunnel servlet URL:

`https://`*hostName*`:`*portNumber*`/`*contextRoot*`/tunnel`

For example, you could set the Context Root field to:

```
/imq
```

The next screen shows that the tunnel servlet has been successfully deployed, is enabled by default, and—in this case—is located at:

```
/var/opt/SUNWappserver8/domains/domain1/server1/applications/
j2ee-modules/imqhttps_1
```

The servlet is now available at the following URL:

`https://`*hostName*`:`*portNumber*`/`*contextRoot*`/tunnel`

Clients can now use this URL to connect to the message service using an HTTPS connection.

Modifying the server.policy file

Application Server enforces a set of default security policies that unless modified would prevent the HTTPS tunnel servlet from accepting connections from the Message Queue broker.

Each application server instance has a file that contains its security policies or rules. For example, the location of this file for the server1 instance on Solaris is:

```
/var/opt/SUNWappserver8/domains/domain1/server1/config/
server.policy
```

To make the tunnel servlet accept connections from the Message Queue broker, an additional entry is required in this file.

➤ To Modify the Application Server's server.policy File

1. Open the server.policy file.
2. Add the following entry:

```
grant codeBase
"file:/var/opt/SUNWappserver8/domains/domain1/server1/
             applications/j2ee-modules/imqhttps_1/-”
{
    permission java.net.SocketPermission "*",
             “connect,accept,resolve";
};
```

Step 4. Configuring the httpsjms Connection Service

HTTPS support is not activated for a broker by default, so you need to reconfigure the broker to activate the httpsjms connection service. Once reconfigured, the broker can be started as outlined in "Starting Brokers" on page 66.

➤ To Activate the httpsjms Connection Service

1. Open the broker's instance configuration file.

   The instance configuration file is stored in a directory identified by the name of the broker instance (*instanceName*) with which the configuration file is associated (see Appendix A, "Platform-Specific Locations of Message Queue Data"):

   …/instances/*instanceName*/props/config.properties

2. Add the `httpsjms` value to the `imq.service.activelist` property:

   ```
   imq.service.activelist=jms,admin,httpsjms
   ```

At startup, the broker looks for a Web server and HTTPS tunnel servlet running on its host machine. To access a remote tunnel servlet, however, you can reconfigure the `servletHost` and `servletPort` connection service properties.

You can also reconfigure the `pullPeriod` property to improve performance. The httpsjms connection service configuration properties are detailed in Table C-2.

Table C-2 `httpsjms` Connection Service Properties

Property	Description
`imq.httpsjms.https.servletHost`	Change this value, if necessary, to specify the name of the host (hostname or IP address) on which the HTTPS tunnel servlet is running. (This can be a remote host or a specific hostname on a local host.) Default: `localhost`.
`imq.httpsjms.https.servletPort`	Change this value to specify the port number that the broker uses to access the HTTPS tunnel servlet. (If the default port is changed on the Web server, you must change this property accordingly.) Default: `7674`.
`imq.httpsjms.https.pullPeriod`	Specifies the interval, in seconds, between HTTP requests made by each client to pull messages from the broker. (Note that this property is set on the broker and propagates to the client runtime.) If the value is zero or negative, the client keeps one HTTP request pending at all times, ready to pull messages as fast as possible. With a large number of clients, this can be a heavy drain on Web server resources and the server may become unresponsive. In such cases, you should set the `pullPeriod` property to a positive number of seconds. This sets the time the client's HTTP transport driver waits before making subsequent pull requests. Setting the value to a positive number conserves Web server resources at the expense of the response times observed by clients. Default: `-1`.
`imq.httpsjms.https.connectionTimeout`	Specifies the time, in seconds, that the client runtime waits for a response from the HTTPS tunnel servlet before throwing an exception. (Note that this property is set on the broker and propagates to the client runtime.) This property also specifies the time the broker waits after communicating with the HTTPS tunnel servlet before freeing up a connection. A timeout is necessary in this case because the broker and the tunnel servlet have no way of knowing if a client that is accessing the HTTPS servlet has terminated abnormally. Default: `60`.

Step 5. Configuring an HTTPS Connection

A client application must use an appropriately configured connection factory administered object to make an HTTPS connection to a broker.

However, the client must also have access to SSL libraries provided by the Java Secure Socket Extension (JSSE) and must also have a root certificate. The SSL libraries are bundled with JDK 1.4. If you have an earlier JDK version, see "Configuring JSSE," otherwise proceed to "Importing a Root Certificate."

Once these issues are resolved, you can proceed to configuring the HTTPS connection.

Configuring JSSE

➤ **To Configure JSSE**

1. Copy the JSSE `.jar` files to the `JRE_HOME/lib/ext` directory.

   ```
   jsse.jar, jnet.jar, jcert.jar
   ```

2. Statically add the JSSE security provider by adding

   ```
   security.provider.n=com.sun.net.ssl.internal.ssl.Provider
   ```

   to the `JRE_HOME/lib/security/java.security` file (where *n* is the next available priority number for security provider package).

3. If not using JDK1.4, you need to set the following JSSE property using the `-D` option to the command that launches the client application:

   ```
   java.protocol.handler.pkgs=com.sun.net.ssl.internal.www.protocol
   ```

Importing a Root Certificate

If the root certificate of the CA who signed your Web server's certificate is not in the trust database by default or if you are using a proprietary Web server/application server certificate, you must add that certificate to the trust database. If this is the case, follow the instruction below, otherwise go to "Configuring the Connection Factory."

Assuming that the certificate is saved in *certFile* and that *trustStoreFile* is your key store, run the following command:

```
JRE_HOME/bin/keytool -import -trustcacerts
-alias aliasForCertificate -file certFile
-keystore trustStoreFile
```

Answer `YES` to the question: `Trust this certificate?`

You also need to specify the following JSSE properties using the `-D` option to the command that launches the client application:

> `javax.net.ssl.trustStore=`*trustStoreFile*
>
> `javax.net.ssl.trustStorePassword=`*trustStorePasswd*

Configuring the Connection Factory

To enable HTTPS support, you need to set the connection factory's `imqAddressList` attribute to the HTTPS tunnel servlet URL. The general syntax of the HTTPS tunnel servlet URL is the following:

> `https://`*hostName*`:`*portNumber*`/`*contextRoot*`/tunnel`

where *hostName*`:`*portNumber* is the name and port of the Web server hosting the HTTPS tunnel servlet and *contextRoot* is a path set when deploying the tunnel servlet on the Web server.

For more information on connection factory attributes in general, and the `imqAddressList` attribute in particular, see the *Message Queue Developer's Guide for Java Clients*.

You can set connection factory attributes in one of the following ways:

- Using the `-o` option to the `imqobjmgr` command that creates the connection factory administered object (see "Adding a Connection Factory" on page 174), or set the attribute when creating the connection factory administered object using the Administration Console (`imqadmin`).
- Using the `-D` option to the command that launches the client application (see the *Message Queue Developer's Guide for Java Clients*).
- Using an API call to set the attributes of a connection factory after you create it programmatically in client application code (see the *Message Queue Developer's Guide for Java Clients*).

Using a Single Servlet to Access Multiple Brokers

You do not need to configure multiple Web servers and servlet instances if you are running multiple brokers. You can share a single Web server and HTTPS tunnel servlet instance among concurrently running brokers. If multiple broker instances are sharing a single tunnel servlet, you must configure the `imqAddressList` connection factory attribute as shown below:

> `https://`*hostName*`:`*portNumber*`/`*contextRoot*`/tunnel?ServerName=`*bkrHostName*`:`*instanceName*

Where *bkrHostName* is the broker instance host name and *instanceName* is the name of the specific broker instance you want your client to access.

To check that you have entered the correct strings for *bkrhostName* and *instanceName*, generate a status report for the HTTPS tunnel servlet by accessing the servlet URL from a browser. The report lists all brokers being accessed by the servlet:

```
HTTPS tunnel servlet ready.
Servlet Start Time : Thu May 30 01:08:18 PDT 2002
Accepting secured connections from brokers on port : 7674
Total available brokers = 2
Broker List :
   jpgserv:broker2
   cochin:broker1
```

Using an HTTP Proxy

If you are using an HTTP proxy to access the HTTPS tunnel servlet:

- Set `http.proxyHost` system property to the proxy server host name.
- Set `http.proxyPort` system property to the proxy server port number.

You can set these properties using the `-D` option to the command that launches the client application.

Troubleshooting

This section describes possible problems with an HTTP or HTTPS connection and provides guidance on how to handle them.

Server or Broker Failure

If the Web server fails and is restarted, all connections are restored and there is no effect on clients. However, if the broker fails and is restarted, an exception is thrown and clients must re-establish their connections.

If both the Web server and the broker fail, and the broker is not restarted, the Web server restores client connections and continues waiting for a broker connection without notifying clients. To avoid this situation, always make sure the broker is restarted.

Client Failure to Connect Through the Tunnel Servlet

If an HTTPS client cannot connect to the broker through the tunnel servlet, do the following:

1. Start the servlet and the broker.
2. Use a browser to manually access the servlet through the HTTPS tunnel servlet URL.
3. Use the following administrative commands to pause and resume the connection:

```
imqcmd pause svc -n httpsjms -u admin
imqcmd resume svc -n httpsjms -u admin
```

When the service resumes, an HTTPS client should be able to connect to the broker through the tunnel servlet.

Appendix D

Frequently Used Command Utility Commands

This appendix lists some frequently used Message Queue Command utility (`imqcmd`) commands. For a comprehensive list of command options and attributes available to you from the command line, refer to "Command Utility" on page 271 in Chapter 13, "Command Line Reference."

Syntax

```
imqcmd subcommand argument [options]
imqcmd -h|H
imqcmd -v
```

`-H` or `-h` provides comprehensive help. The `-v` subcommand provides version information.

When you use `imqcmd`, the Command utility prompts you for a password. To avoid the prompt (and to increase security), you can use the `-passfile` *pathToPassfile* option to point the utility to a password file that contains the administrator username and password.

Example: `imqcmd query bkr -u` *adminUserName* `-passfile` *pathToPassfile* `-b` *myServer*`:7676`

Broker and Cluster Management

```
imqcmd query bkr
imqcmd pause bkr
imqcmd restart bkr
imqcmd resume bkr
imqcmd shutdown bkr -b myBroker:7676
imqcmd update bkr -o "imq.system.max_count=1000"
imqcmd reload cls
```

Broker Configuration Properties (-o option)

Table D-1 lists frequently used broker configuration properties. For a full list of broker configuration properties and their descriptions, see Chapter 14, "Broker Properties Reference."

Table D-1 Broker Configuration Properties (-o option)

Property	Notes
`imq.autocreate.queue`	
`imq.autocreate.queue.maxNumActiveConsumers`	Specify -1 for unlimited
`imq.autocreate.queue.maxNumBackupConsumers`	Specify -1 for unlimited
`imq.autocreate.topic`	
`imq.cluster.url`	
`imq.destination.DMQ.truncateBody`	
`imq.destination.logDeadMessages`	
`imq.log.file.rolloverbytes`	Specify -1 for unlimited
`imq.log.file.rolloversecs`	Specify -1 for unlimited

`imq.log.level`	`NONE`
	`ERROR`
	`WARNING`
	`INFO`

`imq.message.max_size`	Specify -1 for unlimited
`imq.portmapper.port`	
`imq.system.max_count`	Specify -1 for unlimited

Table D-1 Broker Configuration Properties (`-o` option)

Property	Notes
`imq.system.max_size`	Specify `-1` for unlimited

Service and Connection Management

```
imqcmd list svc
```

```
imqcmd query svc
```

```
imqcmd update svc -n jms -o "minThreads=200" -o "maxThreads=400" -o
"port=8995"
```

```
imqcmd pause svc -n jms
```

```
imqcmd resume svc -n jms
```

```
imqcmd list cxn -svn jms
```

```
imqcmd query cxn -n 1234567890
```

Durable Subscriber Management

`imqcmd list dur -d` *MyTopic*

`imqcmd destroy dur -n` *myDurSub* `-c "clientID-111.222.333.444"`

`imqcmd purge dur -n` *myDurSub* `-c "clientID-111.222.333.444"`

Transaction Management

```
imqcmd list txn
```

```
imqcmd commit txn -n 1234567890
```

```
imqcmd query txn -n 1234567890
```

```
imqcmd rollback txn -n 1234567890
```

Destination Management

```
imqcmd create dst -n MyQueue -t q -o "maxNumMsgs=1000" -o
"maxNumProducers=5"
```

```
imqcmd update dst -n MyTopic -t t -o "limitBehavior=FLOW_CONTROL|
REMOVE_OLDEST|REJECT_NEWEST|REMOVE_LOW_PRIORITY"
```

```
imqcmd compact dst -n MyQueue -t q
```

```
imqcmd purge dst -n MyQueue -t q
```

```
imqcmd pause dst -n MyQueue -t q -pst PRODUCERS|CONSUMERS|ALL
```

```
imqcmd resume dst -n MyQueue -t q
```

```
imqcmd destroy dst -n MyQueue -t q
```

```
imqcmd query dst -n MyQueue -t q
```

```
imqcmd list dst -tmp
```

Destination Configuration Properties (-o option)

Table D-2 lists frequently used destination configuration properties. For a full list of destination configuration properties and their descriptions, see Chapter 15, "Physical Destination Property Reference."

Table D-2 Destination Configuration Properties (-o option)

Property	Notes
`consumerFlowLimit`	Specify -1 for unlimited
`isLocalOnly` (create only)	

`limitBehavior`	`FLOW_CONTROL`
	`REMOVE_OLDEST`
	`REJECT_NEWEST`
	`REMOVE_LOW_PRIORITY`

`localDeliveryPreferred (queue only)`	
`maxNumActiveConsumers (queue only)`	Specify -1 for unlimited
`maxNumBackupConsumers (queue only)`	Specify -1 for unlimited
`maxBytesPerMsg`	Specify -1 for unlimited
`maxNumMsgs`	Specify -1 for unlimited
`maxNumProducers`	Specify -1 for unlimited

Table D-2 Destination Configuration Properties (`-o` option)

Property	Notes
`maxTotalMsgBytes`	Specify `-1` for unlimited
`useDMQ`	

Metrics

```
imqcmd metrics bkr -m cxn|rts|ttl -int 5 -msp 20
```

```
imqcmd metrics svc -m cxn|rts|ttl
```

```
imqcmd metrics dst -m con|dsk|rts|ttl
```

Glossary

For information about Message Queue terms, see the glossary in the *Message Queue Technical Overview*. See the Java Enterprise System Glossary (`http://docs.sun.com/doc/816-3875`) for a complete list of terms that are used in the Sun Java System product suite.

Index