Sun Java™ System

Message Queue 3 Administration Guide

2005Q1

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

Part No: 819-0066-10

Contents

List of Figures

List of Tables

List of Procedures

Preface

The Sun Java™ System Message Queue *Administration Guide* provides the information you need in order to administer a Message Queue messaging system.

This book describes Sun Java System Message Queue 3 2005Q1 (Message Queue 3.6).

This preface contains the following sections:

Who Should Use This Book

This guide is meant for administrators and application developers who need to perform Message Queue administration tasks.

A Message Queue administrator is responsible for setting up and managing a Message Queue messaging system, especially the Message Queue message server at the heart of this system.

Before You Read This Book

You must 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

The following table briefly describes the contents of the manual.

Table 1 Book Contents

Part/Chapter	Description
Part I, "Introduction to Message Queue Administration"	
Chapter 1, "Administration Tasks and Tools"	Introduces Message Queue administration tasks and tools.
Chapter 2, "Administration Quick Start"	Provides a hands-on tutorial to acquaint you with the 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 Book Contents *(Continued)*

Part/Chapter	Description
Chapter 5, “Managing a Broker”	Describes broker management tasks.
Chapter 6, “Managing Physical Destinations”	Describes management tasks relating to topics and queues.
Chapter 7, “Managing Security”	Explains security-related tasks, such as managing password files, authentication, authorization, and encryption.
Chapter 8, “Managing Administered Objects”	Describes the object store and explains how to perform tasks related to destination administered objects and connection factory administered objects.
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 message server performance and explains how to tune the message server to optimize its performance.
Chapter 12, “Troubleshooting Problems”	Provides suggestions about how to determine the cause of common Message Queue problems, and about the actions you can take to resolve the problems.
Part III, “Reference”	
Chapter 13, “Command Reference”	Provides syntax and descriptions for the Message Queue command utilities.
Chapter 14, “Broker Properties Reference”	List and describes the properties you can use to configure a broker.
Chapter 15, “Physical Destination Property Reference”	List and describes the properties you can use to configure topics and queues.
Chapter 16, “Administered Object Attribute Reference”	List and describes the properties you can use to configure destination administered objects and connection factory administered objects.
Chapter 17, “JMS Resource Adapter Attribute Reference”	List and describes the properties you can use to configure the Message Queue resource adapter for use with an application server.
Chapter 18, “Metrics Reference”	List and describes the metrics produced by a Message Queue broker.
Part IV, “Appendixes”	

Table 1 Book Contents *(Continued)*

Part/Chapter	Description
Appendix A, "Operating System-Specific Locations of Message Queue Data"	Lists the location 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 use of HTTP for Message Queue communication.

Conventions Used In This Book

This section provides information about the conventions used in this document.

Text Conventions

Table 2 Document Conventions

Format	Description
italics	Italicized text represents a placeholder. Substitute an appropriate clause or value where you see italic text. Italicized text is also used to designate a document title, for emphasis, or for a word or phrase 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 to indicate optional values in a command line syntax statement.
ALL CAPS	Text in all capitals represents file system types (GIF, TXT, HTML and so forth), environment variables (IMQ_HOME), or acronyms (Message Queue, JSP).
Key+Key	Simultaneous keystrokes are joined with a plus sign: Ctrl+A means press both keys simultaneously.
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 summarizes how they are used on the Solaris™, Windows, and Linux platforms.

Table 3 Message Queue Directory Variables

Variable	Description
IMQ_HOME	This is generally used in Message Queue documentation to refer to the Message Queue base directory (root installation directory): • On Solaris and Linux, there is no root Message Queue installation directory. Therefore, IMQ_HOME is not used in Message Queue documentation to refer to file locations on Solaris. • On Solaris and Windows, for Sun Java System Application Server, the root Message Queue installation directory is /imq under the Application Server base directory. • On Windows, the root Message Queue installation directory is set by the Message Queue installer (by default, as C:\Program Files\Sun\MessageQueue3).
IMQ_VARHOME	This is the /var directory in which Message Queue temporary or dynamically-created configuration and data files are stored. It can be set as an environment variable to point to any directory. • On Solaris, IMQ_VARHOME defaults to the /var/imq directory. • On Solaris, for Sun Java System Application Server, Evaluation Edition, IMQ_VARHOME defaults to the IMQ_HOME/var directory. • On Windows IMQ_VARHOME defaults to the IMQ_HOME\var directory. • On Windows, for Sun Java System Application Server, IMQ_VARHOME defaults to the IMQ_HOME\var directory. • On Linux, IMQ_VARHOME defaults to the /var/opt/sun/mq directory.

Table 3 Message Queue Directory Variables *(Continued)*

Variable	Description
IMQ_JAVAHOME	This is an environment variable that points to the location of the Java™ runtime (JRE) required by Message Queue executables: • On Solaris, IMQ_JAVAHOME looks for the java runtime in the following order, but a user can optionally set the value to wherever the required JRE resides. Solaris 8 or 9: /usr/jdk/entsys-j2se /usr/jdk/jdk1.5.* /usr/jdk/j2sdk1.5.* /usr/j2se Solaris 10: /usr/jdk/entsys-j2se /usr/java /usr/j2se • On Linux, Message Queue first looks for the java runtime in the following order, but a user can optionally set the value of IMQ_JAVAHOME to wherever the required JRE resides. /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_JAVAHOME defaults to IMQ_HOME\jre, but a user can optionally set the value to wherever the required JRE resides.

In this guide, IMQ_HOME, IMQ_VARHOME, and IMQ_JAVAHOME are shown *without* platform-specific environment variable notation or syntax (for example, $IMQ_HOME on UNIX®). Path names generally use UNIX directory separator notation (/).

Related Documentation

In addition to this guide, Message Queue provides additional documentation resources.

Message Queue Documentation Set

The documents that comprise the Message Queue documentation set are listed in Table 4 in the order in which you would normally use them.

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	Describes Message Queue concepts, features, and components.
Message Queue Administration Guide	Administrators and developers	Provides background and information needed to perform administration tasks using Message Queue administration tools.
Message Queue Developer's Guide for Java Clients	Developers	Provides information about how to develop a Java client program that uses the Message Queue implementation of the JMS and SOAP/JAXM specifications.
Message Queue Developer's Guide for C Clients	Developers	Provides information about how to develop a C client program that uses the C interface (C-API) to the Message Queue message service.

Online Help

Message Queue includes command line utilities for performing Message Queue message service administration tasks. To access the online help for these utilities, see Chapter 13, "Command Reference.".

Message Queue also includes a graphical user interface (GUI) administration tool, the Administration Console (`imqadmin`). Context sensitive online help is included in the Administration Console.

JavaDoc

JMS and Message Queue API documentation in JavaDoc format is provided at the following location:

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

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 for Message Queue administered objects (see Chapter 3 of the *Message Queue Developer's Guide for Java Clients*), which are of value to developers of messaging applications.

Example Client Applications

A number of example applications that provide sample client application code are included in a platform-specific directory (see Appendix A, "Operating System-Specific Locations of Message Queue Data").

See the README file located in that directory and in each of its subdirectories.

The Java Message Service (JMS) Specification

The JMS specification can be found at the following location:

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

The specification includes sample client code.

Related Third-Party Web Site References

Third-party URLs are referenced in this document and provide additional, related information.

NOTE Sun is not responsible for the availability of third-party Web sites mentioned in this document. Sun does not endorse and is not responsible or liable for any content, advertising, products, or other materials that are 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 that are available on or through such sites or resources.

Sun Welcomes Your Comments

Sun is 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 online form, provide the document title and part number. 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

Administration Tasks and Tools

Sun Java™ System Message Queue administration consists of a number of tasks and a number of tools for performing those tasks.

This chapter first provides an overview of administrative tasks and then describes the administration tools, focusing on common features of the command line administration utilities. The chapter contains the following sections:

Administrative Tasks in a Development Environment

In a development environment, the work focuses on programming Message Queue client applications and programmers often administer their own systems. The Message Queue message server is needed principally for testing. In a development environment, the emphasis is on flexibility, and administration typically includes the following practices:

- Minimal administration, consisting mostly of starting up a broker for developers to use in testing.
- Use of built-in file-based persistence, a file-based user repository, and a file-system store. These simple configurations are usually adequate for development testing.
- In multi-broker testing, no use of a Master Broker.
- Use of auto-created destinations rather than administrator-created destinations.
- Instantiation of administered objects in client code rather than by an administrator.

Administrative Tasks in a Production Environment

In a production environment, in which applications must be reliably deployed and run, administration is more important. The administration tasks you perform depend on the complexity of your messaging system and the complexity of the applications it must support. In general, these tasks can be grouped into setup operations and maintenance operations.

Setup Operations

Typically you must perform at least some, if not all, of the following setup operations:

- **Administrator security** (protected use of administration tools):
  - Authorization: Allow a specific individual or group to access the administrative connection service and consume messages from the dead message queue (see "Access Control for Connection Services" on page 156 and "Access Control for Physical Destinations" on page 157).
  - If you are using the default administrative user (`admin`) and a file-based user repository, change the user password (see "Changing the Default Administrator Password" on page 148).
  - If you are authorizing a group, make sure each administrator belongs to the group.
    - File-based user repository

      The file-based user repository has a single group for administrators (`admin`). If you create a new administrative user, make sure that the new user is in the `admin` group.
    - LDAP user repository

      Create a group in the LDAP server, or use an existing group. Be sure that the user to whom you want to grant administrative privileges is a member of that group, and then authorize administrative connections for the members of that group.

      For more information, see "Using an LDAP Server for a User Repository" on page 149).
- **General security** (see Chapter 7, "Managing Security"):
  - Authentication: Make entries into the file-based user repository or configure the broker to use an existing LDAP user repository.

    (At a minimum, you want to password protect administration capability.)
  - Authorization: Modify access settings in the access control properties file.
  - Encryption: Set up SSL-based connection services (see "Working With an SSL-Based Service" on page 159).

- **Administered objects** (see Chapter 8, "Managing Administered Objects"):
  - Configure or set up an LDAP object store.
  - Create ConnectionFactory and destination administered objects.
- **Broker clusters** (see Chapter 9, "Working With Broker Clusters"):
  - Create a central configuration file.
  - Use a Master Broker.
- **Persistence**: Decide whether you want the broker to use plugged-in persistence or built-in persistence, and set up the desired store (see "Setting Up a Persistent Store" on page 99).
- **Memory management**: Set destination attributes so that the number of messages and the amount of memory allocated for messages fit within available broker memory resources (see Table 15-1 on page 329).

Maintenance Operations

In a production environment, Message Queue message server resources need to be tightly monitored and controlled. Application performance, reliability, and security are at a premium, and you must perform a number of ongoing tasks, described below, using Message Queue administration tools:

- **Application management**
  - Disable the broker's auto-create capability by setting the values for the `imq.autocreate.queue` and `imq.autocreate.topic` properties (see "Message Router Properties" on page 313).
  - Create physical destinations on behalf of applications (see Chapter 6, "Managing Physical Destinations" on page 127).
  - Set user access to destinations (see "Authorizing Users: the Access Control Properties File" on page 152).
  - Monitor and manage destinations (see "Managing Durable Subscriptions" on page 122).
  - Monitor and manage durable subscriptions (see "Managing Durable Subscriptions" on page 122).
  - Monitor and manage transactions (see "Managing Transactions" on page 123).

- **Broker administration and tuning**
  - Use broker metrics to tune and reconfigure the broker (see Chapter 11, "Analyzing and Tuning a Message Service" on page 219).
  - Manage broker memory resources (see Chapter 11, "Analyzing and Tuning a Message Service" on page 219).
  - Add brokers to clusters to balance loads (see Chapter 9, "Working With Broker Clusters").
  - Recover failed brokers (see "Starting Brokers Interactively" on page 67).
- **Managing applications**
  - Create additional ConnectionFactory and destination administered objects as needed (see "Adding and Deleting Administered Objects" on page 189).
  - Adjust ConnectionFactory attribute values to ensure the correct behavior of Java client applications (see Chapter 8, "Managing Administered Objects").

Administrative Tools

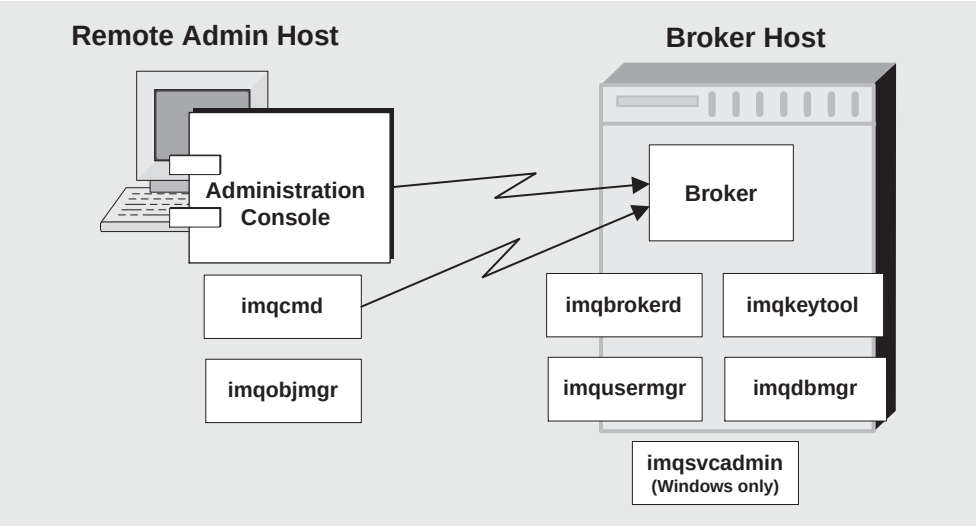
Message Queue administration tools fall into two categories:

- Command line utilities
- A graphical Administration Console (`imqadmin`)

Command Line Utilities

This section introduces the command line utilities you use to perform Message Queue administration tasks. You use the Message Queue utilities to start up and manage a broker and to perform other, more specialized administrative tasks.

Figure 1-1 Local and Remote Administration Utilities



All Message Queue utilities are accessible from a command line interface (CLI). Utility commands share common formats, syntax conventions, and options, as described later in this chapter. You can find reference information on the use of the command line utilities in Chapter 13, "Command Reference."

Broker (`imqbrokerd`) You use the Broker utility to start the broker. You use options to the `imqbrokerd` command to specify whether brokers should be connected in a cluster and to specify additional configuration information that the broker uses at startup.

Command (`imqcmd`) After starting a broker, you use the Command utility to create, update, and delete physical destinations; control the broker and its connection services; and manage the broker's resources.

Object Manager (`imqobjmgr`) You use the Object Manager utility to add, list, update, and delete administered objects in an object store accessible via JNDI. Administered objects allow JMS clients to be provider-independent by insulating them from JMS provider-specific naming and configuration formats.

User Manager (`imqusermgr`) You use the User Manager utility to populate a file-based user repository used to authenticate and authorize users.

Key Tool (`imqkeytool`) You use the Key Tool utility to generate self-signed certificates used for SSL authentication.

Database Manager (`imqdbmgr`) You use the Database Manager utility to create and manage a JDBC-compliant database used for persistent storage.

Service Administrator (`imqsvcadmin`) You use the Service Administrator utility to install, query, and remove the broker as a Windows service.

Administration Console

The Administration Console combines some of the capabilities of two command line utilities: the Command utility (`imqcmd`) and the Object Manager utility (`imqobjmgr`).

You can use the Administration Console and these two command line utilities to manage a broker remotely and to manage Message Queue administered objects. Other command line utilities (`imqusermgr`, `imqdbmgr`, and `imqkeytool`) must be run on the same host as their associated broker, as shown in Figure 1-1 on page 38.

Information on the Administration Console is available in its online help. The command line utilities, which are generally used to perform specialized tasks, are described in "Command Line Utilities."

You can use the administration console to do the following:

- Connect to a broker and manage it.
- Create and manage physical destinations on the broker.
- Connect to an object store.
- Add administered objects to the object store and manage them.

There are some tasks that you cannot use the Administration Console to perform, including starting up a broker, creating broker clusters, configuring more specialized properties of a broker and physical destinations, and managing a user database.

Chapter 2, "Administration Quick Start" provides a brief, hands-on exercise to familiarize you with the Administration Console and to illustrate how you use it to accomplish basic tasks.

Chapter 2

Administration Quick Start

This quick start focuses on basic administration tasks, using the Administration Console, a graphical interface for administering a Message Queue broker and object store. By following the instructions in this chapter, you will learn how to do the following:

- Start a 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 a destination object to the object store and view its properties.

The quick start 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, "Operating System-Specific Locations of Message Queue Data"). In the last part of the quick start, you run this application.

This quick start is provided mainly to guide you through performing basic administration tasks using the Administration Console. It is not a substitute for reading and referring to the documentation.

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

- Configuring certain physical destination properties
- Creating broker clusters
- Managing a user database

For more information on how to accomplish these tasks, see Chapter 6, "Managing Physical Destinations,"Chapter 9, "Working With Broker Clusters," and Chapter 7, "Managing Security."

Getting Ready

Before you can start, you must install the Message Queue product. For more information, see the *Message Queue Installation Guide*. Note that this chapter is Windows-centric, with added notes for UNIX® users.

In this chapter, choosing Item1 > Item2 > Item3 means that you should pull down the menu called Item1, choose Item2 from that menu and then choose Item3 from the selections offered by Item2.

Starting the Administration Console

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

- On Windows, choose Start > Programs > Sun Microsystems > Sun Java System Message Queue 3.6 > Administration.
- On Solaris, enter this command:

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

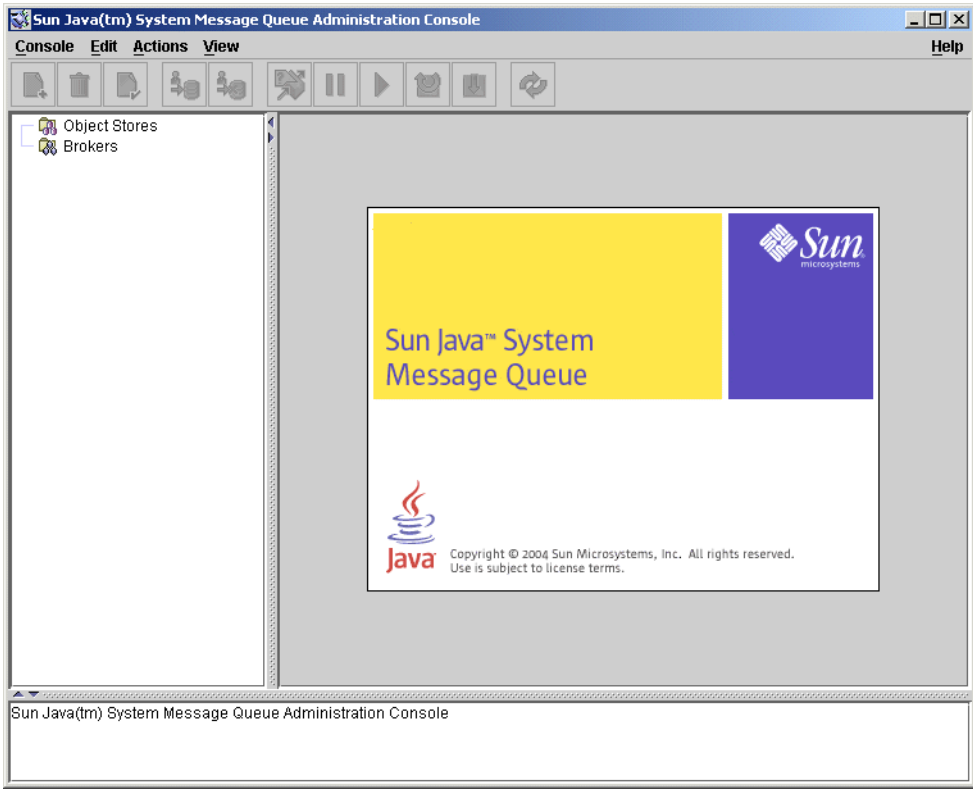
- On Linux, enter this command:

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

You may need to wait a few seconds before the Console window is displayed.

Take a few seconds to examine the Console window.

The Console features a menu bar at the top, a tool bar just underneath the menu bar, a navigational pane to the left, a results pane to the right (now displaying graphics identifying the Sun Java System Message Queue product), and a status pane at the bottom.



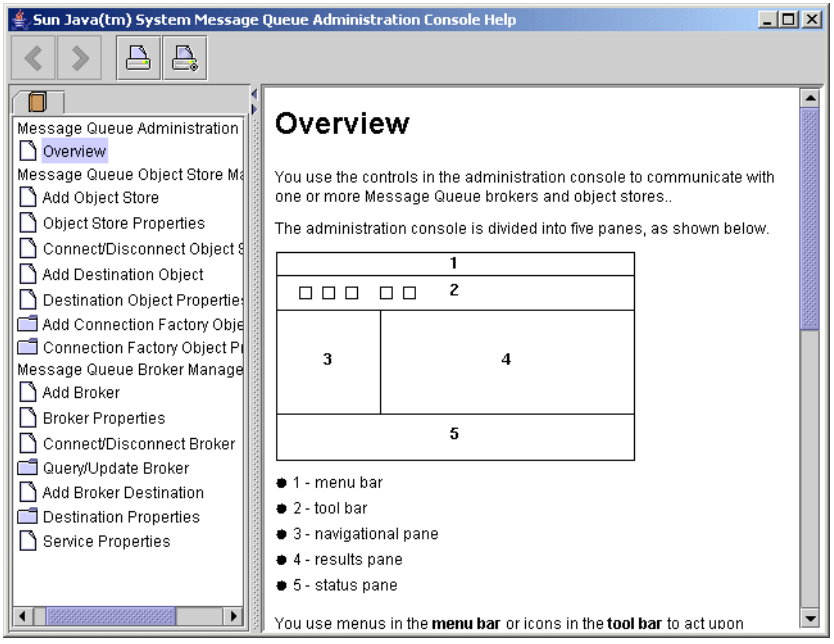
This chapter cannot provide complete information, so let's first find out how to get help information for the Administration Console.

Getting Help

Locate the Help menu at the extreme right of the menu bar.

➤ **To Display Administration Console Help Information**

1. Pull down the Help menu and choose Overview. A help window is displayed.



Notice how the help information is organized. The navigation pane, on the left, shows a table of contents; the results pane, on the right, shows the contents of any item you select in the navigation pane.

Look at the results pane of the Help window. It shows a skeletal view of the Administration Console, identifying the use of each of the Console's panes.

2. Look at the Help window's navigational pane. It organizes topics in three areas: overview, object store management, and broker management. Each of these areas contains files and folders. Each folder provides help for dialog boxes containing multiple tabs; each file provides help for a simple dialog box or tab.

   Your first Console administration task, "Adding a Broker" on page 46, will be to create a reference to a broker you manage through the Console. Before you start, however, check the online help for information.

3. Click the Add Broker item in the Help window's navigational pane.

   Note that the results pane has changed. It now contains text that explains what it means to add a broker and that describes the use of each field in the Add Broker dialog box. Field names are shown in bold text.

4. Read through the help text.

5. Close the Help window.

Starting a Broker

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

- On Windows, choose Start > Programs > Sun Microsystems > Sun Java System Message Queue 3.6 > Message Broker.
- On Solaris, enter this command:

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

- On Linux, enter this command:

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

If you used the Windows Start menu, the command window appears. The command response appears, and indicates that the broker is ready by displaying lines like the following:

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

Bring the Administration Console window back into focus. You are now ready to add the broker to the Console and to connect to it.

You do not have to start the broker before you add a reference to it in the Administration Console, but you must start the broker before you can connect to it.

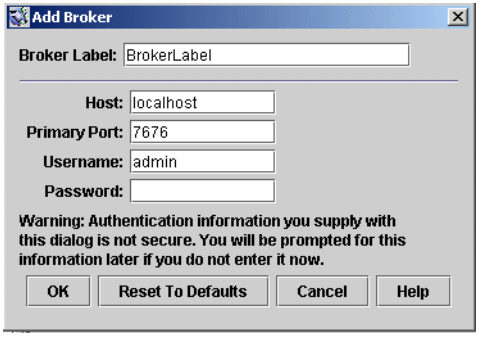
Adding a Broker

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. Right-click on Brokers in the navigation pane and choose Add Broker.
2. Enter `MyBroker` in the Broker Label field.

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

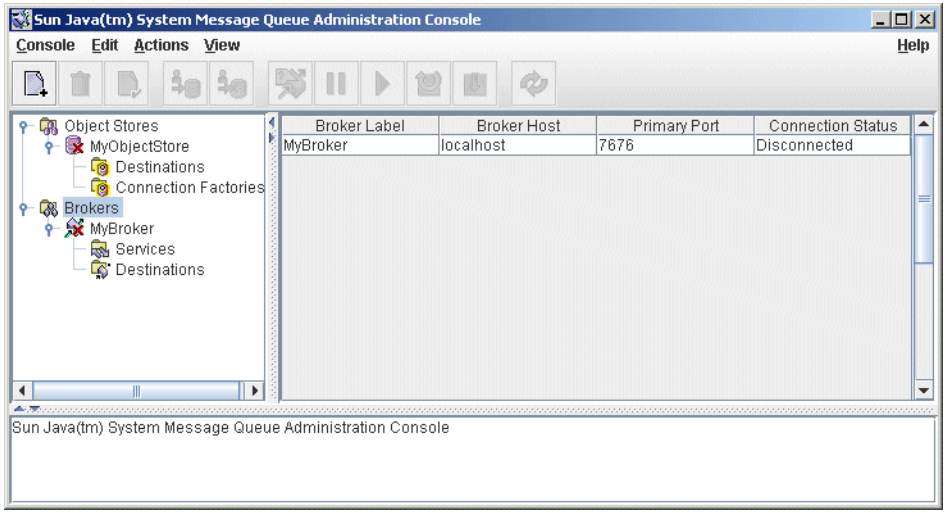


   Note the default host name (`localhost`) and primary port (`7676`) specified in the dialog box. These are the values you must specify later, when you configure the connection factory that the client will use to set up connections to this broker.

   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.

   Look at the navigation pane. The broker you just added should be listed there under Brokers. The red X over the broker icon tells you that the broker is not currently connected to the console.



4. Right-click on MyBroker and choose Properties from the popup menu.

   The broker properties dialog box is displayed. You can use this dialog box to update any of the properties you specified when you added the broker.

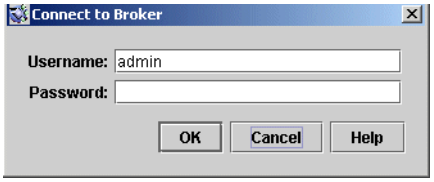
5. Click Cancel to dismiss the dialog box.

Connecting to the Broker

➤ **To Connect to the Broker**

1. Right-click `MyBroker` and choose Connect to Broker.

   A dialog box appears and requests a user name and password.



   By default, the Administration Console can connect to a broker as user `admin` with password `admin`. For this exercise, you use the default value. In a real-world environment, you should establish secure user names and passwords as soon as you can. See "Authenticating Users" on page 142 for more information.

2. Enter `admin` in the Password field.

   Specifying the user name `admin` and supplying the correct password connects you to the broker, with administrative privileges.

3. Click OK to connect to the broker.

After you connect to the broker, you can choose from the Actions menu to get information about the broker, to pause and resume the broker, to shutdown and restart the broker, and to disconnect from the broker.

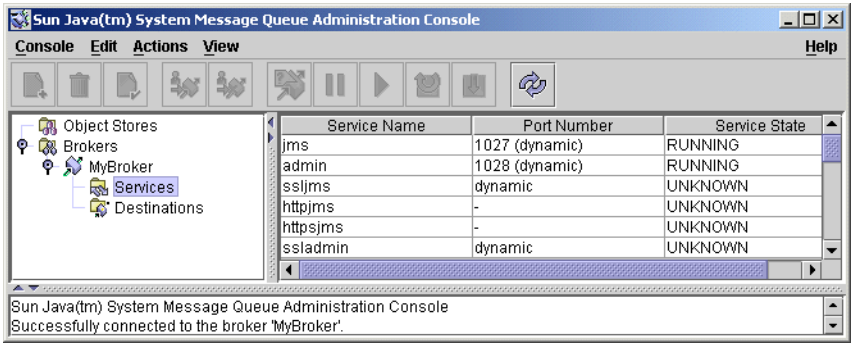
Viewing Connection Services

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

➤ **To View Available Connection Services**

1. Select Services in the navigation pane.

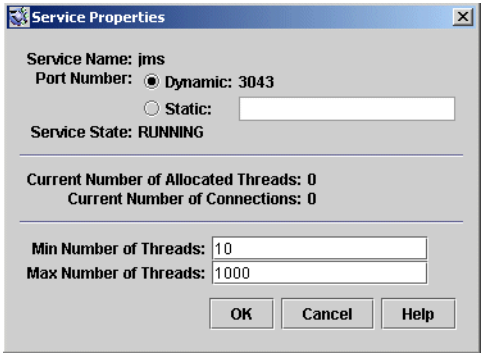
   Available services are listed in the results pane. For each service, its name, port number, and state is provided.



2. Select the jms service by clicking on it in the results pane.
3. Pull down the Actions menu and note the highlighted items.

   You have the option of pausing the jms service or of viewing and updating its properties.
4. Choose Properties from the Actions menu.

   Note that by using the Service Properties dialog box, you can assign the service a static port number and you can change the minimum and maximum number of threads allocated for this service.



5. Click OK or Cancel to close the Properties dialog box.
6. Select the admin service in the results pane.

7. Pull down the Actions menu.

   Notice that you cannot pause this service (the pause item is disabled). The admin service is the administrator's link to the broker. If you paused it, you would no longer be able to access the broker.

8. Choose Actions > Properties to view the properties of the admin service.

9. Click OK or Cancel when you're done.

Adding Physical Destinations to a Broker

By default, physical destination auto-creation is enabled for a broker. Auto-creation enables a broker to dynamically create physical destinations.

In a development environment, you do not have to explicitly create physical destinations in order to test client code.

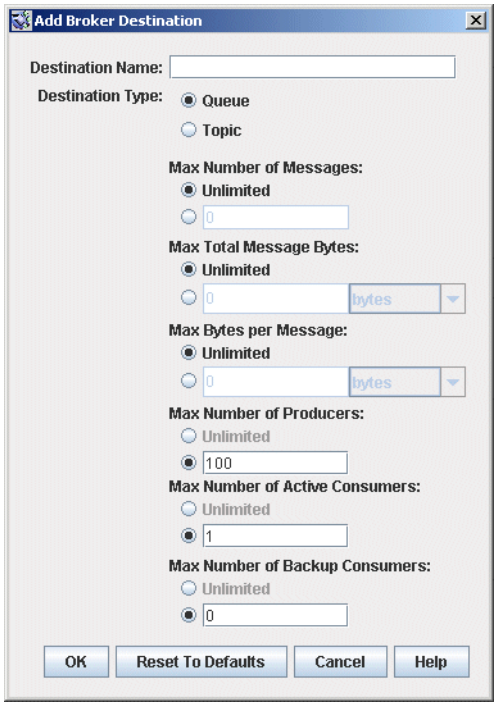
In a production setting, it is advisable to explicitly create physical destinations. This allows you, the administrator, to be fully aware of the physical destinations that are in use on the broker.

You will now add a physical destination to the broker. Note the name that you assign to the destination; you will need it later when you create an administered object that corresponds to this physical destination.

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

1. Right-click the Destinations node of MyBroker and choose Add Broker Destination.

   The following dialog box is displayed:



2. Enter `MyQueueDest` in the Destination Name field.
3. Select the Queue radio button if it is not already selected.
4. Click OK to add the physical destination.

   The physical destination now appears in the results pane.

Administering Physical Destinations

Once you have added a physical destination on the broker, you can do any of the following tasks, as described in the following procedures:

- View and update the properties of a physical destination
- Purge messages at a physical destination
- Delete a physical destination

➤ To View the Properties of a Physical Destination

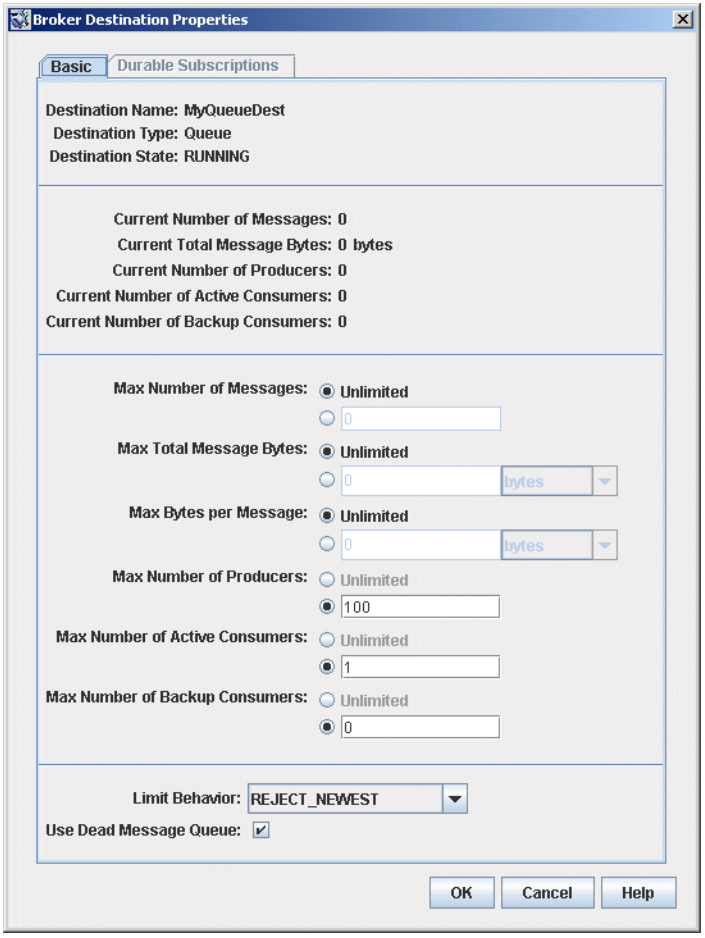
1. Select the Destinations node of MyBroker.

   Two physical destinations appear in the results panel, MyQueueDest and `mq.sys.dmq`. The `mq.sys.dmq` destination is a system-created queue that stores expired and rejected messages for the broker. For now, ignore this dead message queue.

2. Select MyQueueDest in the results pane.

3. Choose Actions > Properties.

   The following dialog box is displayed:



Note that the dialog box displays current status information about the queue as well as some properties that you can change.

4. Click Cancel to close the dialog box.

➤ **To Purge Messages From a Physical Destination**

1. Select the physical destination in the results pane.
2. Choose Actions > Purge Messages.

   A confirmation dialog box is displayed.

Purging messages removes the messages and leaves an empty destination.

➤ **To Delete a Destination**

1. Select the physical destination in the results pane.
2. Choose Edit > Delete.

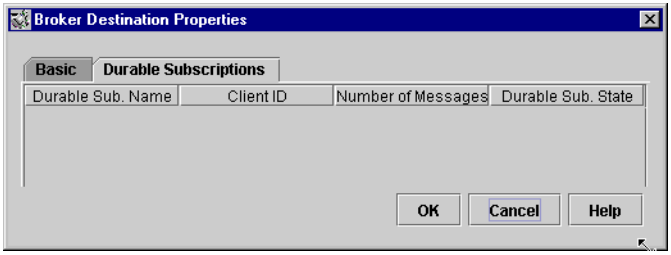
   A confirmation dialog box is displayed.

NOTE Do not delete the MyQueueDest queue destination.

Deleting a physical destination purges the messages at that destination and removes the destination.

Getting Information About Topics

The broker topic destination properties dialog box includes an additional tab that lists information about durable subscriptions. This tab is disabled for queues.



You can use this dialog box to:

- Purge durable subscriptions, removing all messages associated with a durable subscription
- Delete durable subscriptions, purging all messages associated with a durable subscription and also removing the durable subscription

Working with Object Stores

An object store is used to store Message Queue administered objects. These administered objects encapsulate Message Queue-specific implementation and configuration information about objects that are used by client applications. An object store can be an LDAP directory server or a file system store (directory in the file system).

Administered objects can be instantiated and configured within client code. However, it is preferable that an administrator create, configure, and store these objects in an object store that client applications can access using JNDI. This allows client code to be provider-independent.

You cannot use the Administration Console to *create* an object store. You must do this ahead of time as described in the following section.

Adding an Object Store

Adding an object store creates a reference to an existing object store in the Administration Console. This reference is retained even if you quit and restart the Console.

➤ **To Add a File-System Object Store**

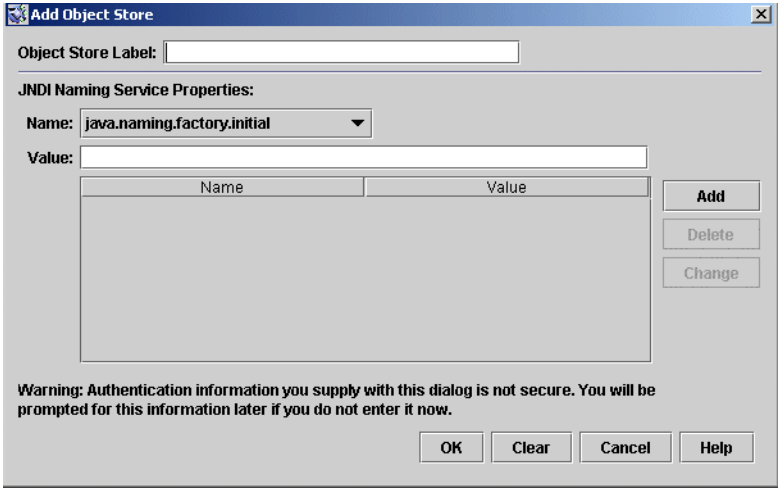
1. If you do not already have a folder named `Temp` on your C drive, create it now.

   The sample application used in this chapter assumes that the object store is a folder named `Temp` on the C drive. In general, a file-system object store can be any directory on any drive.

   Non-Windows: you can use the `/tmp` directory, which should already exist.

2. Right-click on Object Stores and choose Add Object Store.

   The following dialog box is displayed:



3. Enter `MyObjectStore` in the field named ObjectStoreLabel.

   This simply provides a label for the display of the object store in the Administration Console.

   In the following steps, you must enter JNDI name/value pairs. These pairs are used by JMS-compliant applications for looking up administered objects.

4. From the Name drop-down list, select `java.naming.factory.initial`.

   This property allows you to specify what JNDI service provider you wish to use. For example, a file system service provider or an LDAP service provider.

5. In the Value field, enter the following

   `com.sun.jndi.fscontext.RefFSContextFactory`

   This means that you will be using a file system store. (For an LDAP store, you would specify `com.sun.jndi.ldap.LdapCtxFactory`.)

   In a production environment, you will probably want to use an LDAP directory server as an object store. For information about setting up the server and doing JNDI lookups, see "LDAP Server Object Store" on page 174.

6. Click the Add button.

   Notice that the property and its value are now listed in the property summary pane.

7. From the Name drop-down list, choose `java.naming.provider.url`.

   This property allows you to specify the exact location of the object store. For a file system type object store, this will be the name of an existing directory.

8. In the Value field, enter the following

   `file:///C:/Temp`

   (`file:///tmp` on Solaris and Linux)

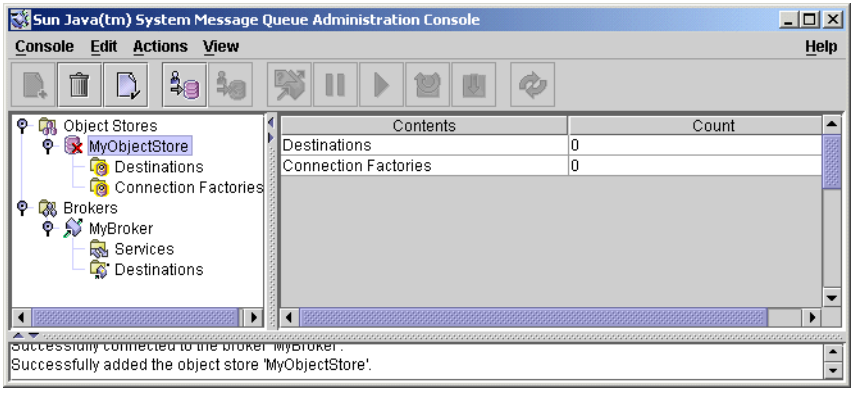
9. Click the Add button.

   Notice that both properties and their values are now listed in the property summary pane. If you were using an LDAP server, you might also have to specify authentication information; this is not necessary for a file-system store.

10. Click OK to add the object store.

11. If the node MyObjectStore is not selected in the navigation pane, select it now.

    The Administration Console now looks like this:



The object store is listed in the navigation pane and its contents, Destinations and Connection Factories, are listed in the results pane. We have not yet added any administered objects to the object store, and this is shown in the Count column of the results pane.

A red X is drawn through the object store's icon in the navigation pane. This means that it is disconnected. Before you can use the object store, you must connect to it.

Checking Object Store Properties

While the Administration Console is disconnected from an object store, you can examine and change some of the properties of the object store.

➤ **To Display the Properties of an Object Store**

1. Right click on MyObjectStore in the navigational pane.
2. Choose Properties from the popup menu.

   A dialog box is displayed that shows all the properties you specified when you added the object store. You can change any of these properties and click OK to update the old information.
3. Click OK or Cancel to dismiss the dialog box.

Connecting to an Object Store

Before you can add objects to an object store, you must connect to it.

➤ **To Connect to an Object Store**

1. Right click on MyObjectStore in the navigational pane.
2. Choose Connect to Object Store from the popup menu.

   Notice that the object store's icon is no longer crossed out. You can now add objects, connection factories and destinations, to the object store.

Adding a Connection Factory Administered Object

You can use the administration console to create and configure a connection factory. A connection factory is used by client code to connect to the broker. By configuring a connection factory, you can control the behavior of the connections it is used to create.

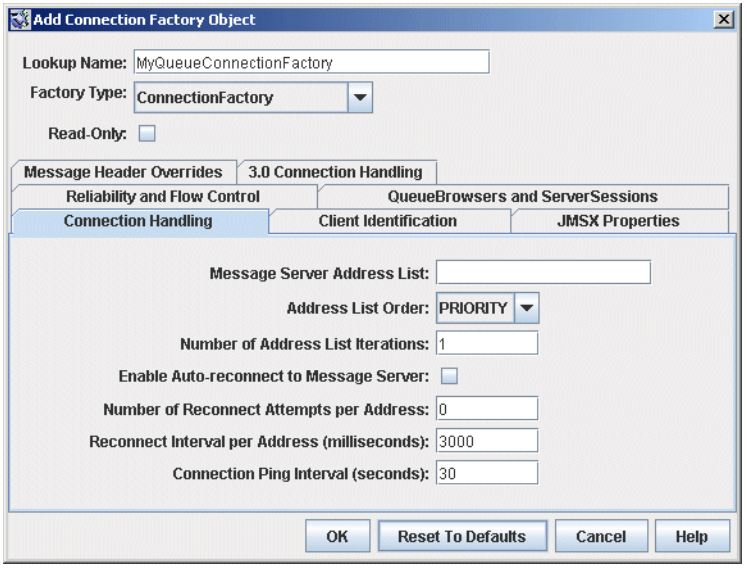
For information on configuring connection factories, see the online help and the *Message Queue Developer's Guide for Java Clients*.

NOTE The Administration Console lists and 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 receive an error when you attempt the add operation.

➤ To Add a Connection Factory to an Object Store

1. If not already connected, connect to MyObjectStore (see "Connecting to an Object Store" on page 57)
2. Right click on the Connection Factories node and choose Add Connection Factory Object.

   The Add Connection Factory Object dialog box is displayed.



3. Enter the name "`MyQueueConnectionFactory`" in the Lookup Name field.

   This is the name that the client code uses when it looks up the connection factory as shown in the following line from `HelloWorldMessageJNDI.java`:

```
qcf=(javax.jms.QueueConnectionFactory)
       ctx.lookup("MyQueueConnectionFactory")
```

4. Select the QueueConnectionFactory from the pull-down menu to specify the type of the connection factory.
5. Click the Connection Handling tab.
6. The Message Server Address List field is where you would normally enter the address of the broker to which the client will connect. An example for this field looks like this:

   ```
   mq://localhost:7676/jms
   ```

   You do not need to enter a value since, by default, the connection factory is configured to connect to a broker running on the localhost on port `7676`, which is the configuration that the quick start example expects.
7. Click through the tabs for this dialog box to see the kind of information that you can configure for the connection factory. Use the Help button in the lower right hand corner of the Add Connection Factory Object dialog box to get information about individual tabs. Do not change any of the default values for now.
8. Click OK to create the queue connection factory.
9. Look at the results pane: the lookup name and type of the newly created connection factory are listed.

Adding a Destination Object

Destination administered objects are associated with physical destinations on the broker and they point to those destinations. Destination administered objects enable clients to look up and find physical destinations, independently of provider-specific destination names and configurations.

When a client sends a message, it either looks up or instantiates a destination administered object and references it in the `send()` method of the JMS API. The broker is then responsible for delivering the message to the physical destination that is associated with that administered object, as follows:

- If you have created a physical destination that is associated with that administered object, the broker delivers the message to that physical destination.
- If you have not created a physical destination and auto-creation of physical destinations is enabled, the broker itself creates the physical destination and delivers the message to that destination.

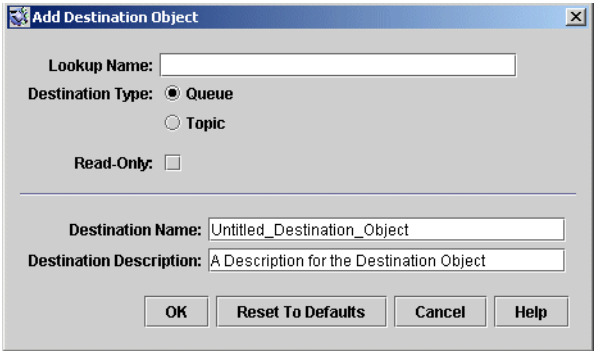
- If you have not created a physical destination and auto-creation of physical destinations is disabled, the broker cannot create a physical destination and cannot deliver the message.

In the next part of the quick start, you will be adding an administered object that corresponds to the physical destination you added earlier.

➤ To Add a Destination to an Object Store

1. Right-click on the Destinations node (under the MyObjectStore node) in the navigation pane.
2. Choose Add Destination Object.

   The Administration Console displays an Add Destination Object dialog box that you use to specify information about the object.



3. Enter "`MyQueue`" in the Lookup Name field.

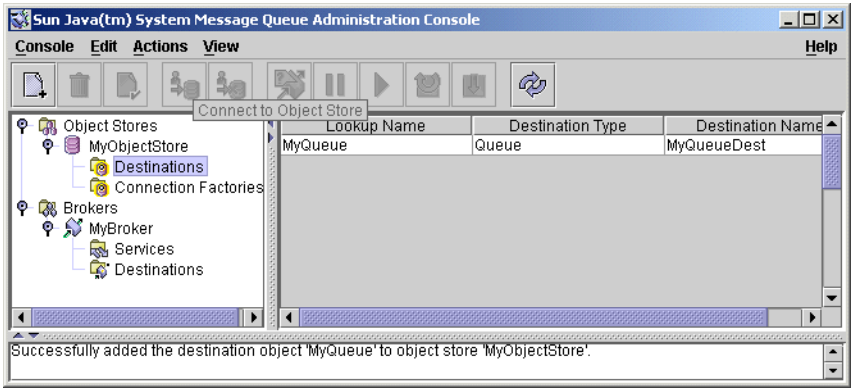
   The lookup name is used to find the object using JNDI lookup calls. In the sample application, the call is the following:

   ```
   queue=(javax.jms.Queue)ctx.lookup("MyQueue");
   ```

4. Select the Queue radio button for the Destination Type.
5. Enter `MyQueueDest` in the Destination Name field.

   This is the name you specified when you added a physical destination on the broker (see "Adding Physical Destinations to a Broker" on page 50).

6. Click OK.
7. Select Destinations in the navigation pane and notice how information about the queue destination administered object you have just added is displayed in the results pane.



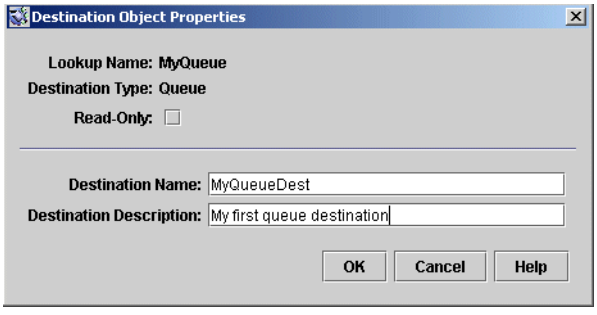
Viewing Administered Object Properties

To view or update the properties of an administered object, you select Destinations or Connection Factories in the navigation pane, select a specific object in the results pane, and choose Actions > Properties.

➤ **To View or Update the Properties of a Destination Object**

1. Select the Destinations node of MyObjectStore in the navigational pane.
2. Select MyQueue in the results pane.
3. Choose Actions > Properties to view the Destination Object Properties dialog box.

   Note that the only values you can change are the destination name and the description. To change the lookup name, you would have to delete the object and then add a new queue administered object with the desired lookup name.
4. Click Cancel to dismiss the dialog box.



Updating Console Information

Whether you work with object stores or brokers, you can update the visual display of any element or groups of elements by choosing View > Refresh.

Running the Sample Application

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

- A queue physical destination named MyQueueDest
- A queue connection factory administered object and queue administered object with JNDI lookup names MyQueueConnectionFactory and MyQueue respectively

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

➤ **To Run the HelloWorldMessageJNDI Application**

1. Make the directory that includes the `HelloWorldmessageJNDI` application your current directory; for example:

   `cd IMQ_HOME\demo\helloworld\helloworldmessagejndi` (Windows)

   `cd /usr/demo/imq/helloworld/helloworldmessagejndi` (Solaris)

   `cd /opt/sun/mq/examples/helloworld/helloworldmessagejndi` (Linux)

   You should find the HelloWorldMessageJNDI.class file present. (If you make changes to the application, you must re-compile it using the instructions for compiling a client application in the Quick Start Tutorial of the *Message Queue Developer's Guide for C Clients*.) 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`, and `fscontext.jar`. See the *Message Queue Developer's Guide for Java Clients* for instructions on setting the `CLASSPATH`.

   The JNDI jar file (`jndi.jar`) file is bundled with JDK 1.4. If you are using this JDK, you do not have to add `jndi.jar` to your `CLASSPATH` setting. If you are using an earlier version of the JDK, you must include `jndi.jar` in your `CLASSPATH`. See the *Message Queue Developer's Guide for Java Clients* for additional information)

2. Before you run the application, open the source file `HelloWorldMessageJNDI.java` and read through the source. It is short, but it is amply documented and it should be fairly clear how it uses the administered objects and destinations you have created.

3. Run the `HelloWorldMessageJNDI` application by executing one of the commands below:

   `java HelloWorldMessageJNDI` (Windows)

   `% java HelloWorldMessageJNDI file:///tmp` (Solaris and Linux)

   If the application runs successfully, you should see the following output:

```
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 start brokers and clients.

The chapter contains the following sections:

The configuration of the broker instance is governed by a set of configuration files and by options passed with the `imqbrokerd` command, which override corresponding properties in the configuration files. For information about broker configuration, see Chapter 4, "Configuring a Broker" on page 73.

Preparing System Resources

Before you start a broker, there are two system-level tasks to perform: synchronizing the system clocks, and, on Solaris or Linux, setting the file descriptor limits. The next 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 (TimeToLive). Timestamps from clocks that are not synchronized could prevent the TimeToLive feature 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 Time 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 Limits (Solaris or Linux)

On the Solaris and Linux platforms, the shell in which the client or broker is running places a soft limit on the number of file descriptors that a process can use. In the Message Queue system, each connection a client makes, or each connection 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 number of connections is limited by these factors. You cannot have a broker or client running with more than 256 connections on Solaris or 1024 on Linux without changing the file descriptor limit. (The connection limit is actually lower than that due to the use of file descriptors for persistence.)

To change the file descriptor limit, see the `ulimit` man page. The limit needs to be changed in each shell in which a client or broker will be executing.

Starting Brokers Interactively

You can start brokers interactively from the command line, using the `imqbrokerd` command. (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 user who initially started it. When the broker instance first starts, Message Queue uses that user's umask to set permissions on broker instance directories containing configuration information and persistent data. Each broker instance has its own set of configuration properties and file-based message store.

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`.

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 Chapter 13, "Command Reference," 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 you're running the broker on (Solaris, Linux, or Windows).

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. 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.

For reference information about the syntax, subcommands, and options of the `imqsvcadmin` command, see Chapter 13, "Command Reference."

Installing a broker as a Windows service means that it will start at system startup time and run in the background until you shut down. Consequently, you do not use the `imqbrokerd` command to start the broker unless you want to start an additional instance.

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 Interactively" on page 67. Use the `imqcmd` command to control broker operations as usual.

When a broker runs as a Windows service, Task Manager lists the broker as two executable processes:

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

A system can have only one broker that is running as a Windows service.

Reconfiguring the Broker Service

The sequence for reconfiguring the Windows service is as follows:

1. Stop the service.
2. Remove the service.
3. Add the service, specifying different broker startup options with the `-args` option, or different Java version arguments with the `-vmargs` option.

Using an Alternative Java Runtime

You can use either the `-javahome` or `-jrehome` options to specify the location of an alternative Java runtime. You can also specify these options in the Windows Services Control Panel Startup Parameters field.

The Startup Parameters field treats the back slash (\) as an escape character, so you must type it twice when using it as a path delimiter; for example, `-javahome d:\\jdk1.3`.

Displaying the Broker Service Startup Options

To determine the startup options for the broker service, use the `query` option to the `imqsvcadmin` command.

```
imqsvcadmin query

Service iMQ_Broker is installed.
Display Name: iMQ_Broker
Start Type: Manual
Binary location: c:\Program Files\Sun Microsystems\
                    Message Queue 3.5\bin\imqbrokersvc
JavaHome: c:\j2sdk1.4.0
Broker Args: -passfile d:\imqpassfile
```

Troubleshooting Service Startup Problems

If you get an error when you try to start the service, you can view error events that were logged.

➤ **To See Logged Service Error Events**

1. Start the Event Viewer.
2. Look under Log > Application.
3. Select View > Refresh to see any error events.

Removing a Broker That Is Running as a Windows Service

To remove a broker that is running as a service, do one of the following:

- Use commands. First use the `imqcmd shutdown bkr` command to shut down the broker and then use the `imqsvcadmin remove` command to remove the service.

- Use the Control Panel's management tool for Windows services. This feature is available in different locations in different versions of Windows.

Restart your computer when you are done.

Starting Message Queue 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 and ensure 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 developers might 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 developer or application documentation should provide information on attribute values that the application sets. You might want to override some attributes that the application sets. You do so by specifying those attributes on the command line.

You might also want to specify attributes on the command line for any Java client that uses a JNDI lookup to find its connection factory. If the lookup returns a connection factory that is older than the application, the connection factory might lack support for more recent attributes. In such a case, Message Queue sets those attributes to default values. By specifying the attributes on the command line, you can set them to nondefault values.

To provide attribute values on the command line, use the following command line syntax for a Java application:

`java [[-D`*attribute*`=`*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 the client application `MyMQClient`. The application connects to a broker on the host `OtherHost` at port `7677`, overriding any host name and port set by the application.

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

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 to do so. Communication with the application developer is necessary to understand the best way to start the client program.

Removing a Broker Instance

This section contains information on removing a broker instance on Solaris or Linux. For information about removing a Windows service, see "Removing a Broker That Is Running as a Windows Service" on page 70.

To remove a broker instance, use the `imqbrokerd` command with the `-remove` option. The command format for removing a broker instance is as follows:

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

For example, if the name of the broker is `myBroker`, this is the command:

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

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

For a list of options that you can use to remove a broker, see the `imqbrokerd` reference information in "Command Reference" on page 279.

On Solaris or Linux, 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`.

Chapter 4

Configuring a Broker

When a broker instance starts, its configuration is governed by a set of configuration files and by the options passed to the `imqbrokerd` command. This chapter explains how configuration files and command line options interact to configure a broker instance, describes the functions of each broker component and lists its configuration properties, and then explains how to set up the configuration.

The chapter contains the following sections:

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

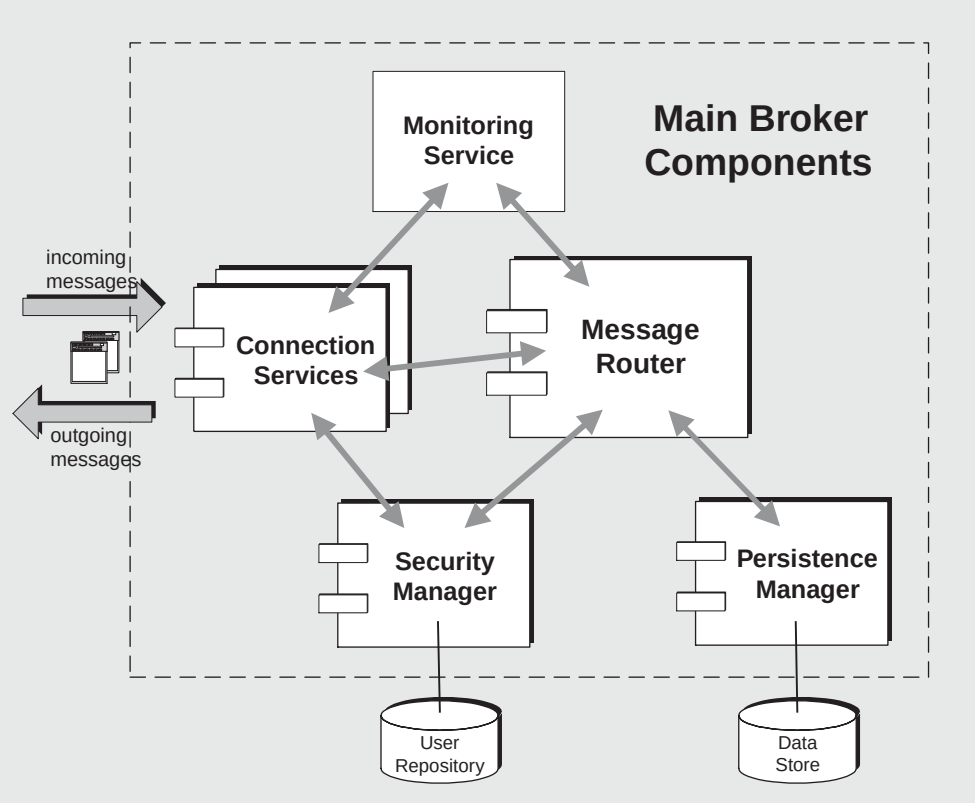
About Configurable Broker Components

Message delivery in a Message Queue messaging system—from producing clients to destinations, and then from destinations to one or more consuming clients—is performed by a broker, or by a cluster of broker instances working in tandem.

To perform message delivery, a broker must set up communication channels with clients, perform authentication and authorization, route messages appropriately, guarantee reliable delivery, and provide data for monitoring system performance.

To perform its functions, a broker uses a number of internal components, each with a specific role in the delivery process. These broker components are illustrated in Figure 4-1.

Figure 4-1 Broker Service Components



The Message Router component performs the key message routing and delivery service, and the others provide important support services. Table 4-1 briefly describes each component.

Table 4-1 Main Broker Service Components and Functions

Component	Description/Function	For Property Descriptions...
Connection Services	Manages the physical connections between a broker and clients, providing transport for incoming and outgoing messages.	"Connection Service Properties" on page 311
Message Router	Manages the routing and delivery of messages: These include JMS messages as well as control messages used by the Message Queue messaging system to support JMS message delivery.	"Message Router Properties" on page 313
Persistence Manager	Manages the writing of data to persistent storage and the retrieval of data from persistent storage.	"Persistence Manager Properties" on page 316
Security Manager	Provides authentication services for users requesting connections to a broker and authorization services (access control) for authenticated users.	"Security Manager Properties" on page 320
Monitoring Service	Generates metrics and diagnostic information that can be written to a number of output channels that an administrator can use to monitor and manage a broker.	"Monitoring and Logging Properties" on page 324

You can configure these components to optimize broker performance, depending on load conditions, application complexity, and so on. The following sections explore the functions that each component performs and the properties that you can set to affect its behavior.

Connection Services

A Message Queue broker supports communication with both Message Queue application clients and Message Queue administration clients. Each connection service is specified by its service type and protocol type, as follows:

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

Table 4-2 lists the connection services available from a Message Queue broker:

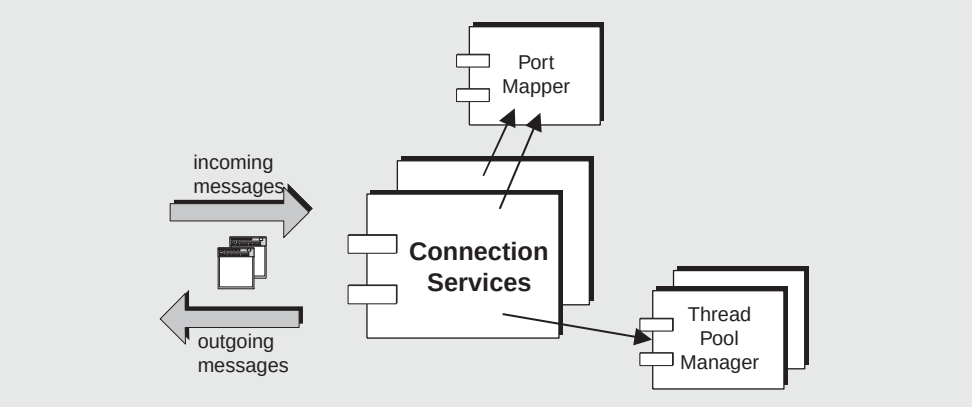
Table 4-2 Connection Services Supported by a Broker

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	ADMIN	tls (SSL-based security)

You can configure a broker to run any or all of these connection services. Each connection service is available at a particular port, specified by the broker's host name and a port number. The `jms` and `admin` services are enabled by default.

Message Queue can dynamically map a connection service to a port number, or you can explicitly assign a port. Each service registers itself with a common Port Mapper but has its own Thread Pool Manager, as shown in Figure 4-2.

Figure 4-2 Connection Services Support



The next sections describe the relationship between a connection service and the Port Mapper and Thread Pool Manager.

Port Mapper

Message Queue provides a *Port Mapper* that assigns ports to connection services. The Port Mapper resides at a standard port number, 7676. When a client sets up a connection with the broker, it first contacts the Port Mapper, requesting the port number of a specified connection service.

The port numbers for the jms, ssljms, admin and ssladmin connection services can be dynamic or static. By default, a connection service dynamically configures its port when it starts up. Alternatively, you can specify a static port for the service, but static port numbers are not generally recommended. Static port numbers are typically used only for special situations, such as connections that traverse a firewall.

The httpjms and httpsjms services are configured using properties described in Table C-1 on page 373 and Table C-3 on page 385, respectively, in Appendix C, "HTTP/HTTPS Support."

Thread Pool Manager

Each connection service is multi-threaded, supporting multiple connections. The threads needed for these connections are maintained in a thread pool managed by a *Thread Pool Manager* component.

You can configure the Thread Pool Manager to set a minimum number and maximum number of threads maintained in the thread pool. As threads are needed by connections, they are added to the thread pool. When the minimum number of threads is exceeded, the system shuts down threads as they become free, until the minimum number threshold is reached, to save memory resources. This number should be large enough so that new threads do not have to be continually created. Under heavy connection loads, the number of threads might increase until the thread pool's maximum number is reached, after which connections must wait until a thread becomes available.

The threads in a thread pool can be dedicated to a single connection (*dedicated* model) or assigned to multiple connections, as needed (*shared* model).

Dedicated model Each connection to the broker requires two dedicated threads: one handles incoming messages for the connection and one handles outgoing messages for the connection. This limits the number of connections to half the maximum number of threads in the thread pool, but it provides for high performance.

Shared model (Enterprise Edition) Connections are processed by a shared thread whenever sending or receiving messages. Because each connection does not require dedicated threads, this model increases the number of connections that a connection service (and therefore, a broker) can support. However there is some performance overhead involved in the sharing of threads. The Thread Pool Manager uses a set of distributor threads that monitor connection activity and assign connections to threads as needed. The performance overhead involved in this activity can be minimized by limiting the number of connections monitored by each such distributor thread.

Security

Each connection service supports specific authentication and authorization (access control) features (see "Security Manager" on page 88).

Connection Service Properties

These are the configurable properties related to connection services:

- `imq.service.activelist`. List of connection services to be started at broker startup.
- `imq.hostname`. Specifies the host to which all connection services bind if there is more than one host available (for example, if there is more than one network interface card in a computer).
- `imq.portmapper.port`. Specifies the broker's primary port—the port at which the Port Mapper resides.
- `imq.portmapper.`*hostname*. Specifies the host to which the Port Mapper binds if there is more than one host available.
- `imq.portmapper.backlog`. Specifies the maximum number of concurrent requests that the Port Mapper can handle before rejecting requests. The property sets the number of requests that can be stored in the operating system backlog waiting to be handled by the Port Mapper.
- `imq.`*service_name*`.`*protocol_type*`.port`. For `jms`, `ssljms`, `admin`, and `ssladmin` services only, specifies the port number for the named connection service.
- `imq.`*service_name*`.`*protocol_type*`.hostname`. For `jms`, `ssljms`, `admin`, and `ssladmin` services only, specifies the host to which the named connection service binds if there is more than one host available.
- `imq.`*service_name*`.min_threads`. Specifies the number of threads, which once reached, are maintained in the thread pool for use by the named connection service.

- `imq.`*service_name*`.max_threads`. Specifies the number of threads beyond which no new threads are added to the thread pool for use by the named connection service.
- `imq.`*service_name*`.threadpool_model`. Specifies whether threads are dedicated to connections or shared by connections as needed for the named connection service.
- `imq.shared.connectionMonitor_limit`. For shared thread pool model only, specifies the maximum number of connections that can be monitored by a distributor thread.

For full descriptions of these properties, see Table 14-2 on page 311.

Message Router

Once connections have been established between clients and a broker using the supported connection services, message routing and delivery can proceed.

Basic Delivery Mechanisms

Broadly speaking, messages handled by a broker fall into two categories:

- JMS payload messages that are sent by producer clients and destined for consumer clients
- Control messages that are sent to and from clients to support the delivery of the JMS messages

If an incoming message is a JMS message, the broker routes it to consumer clients, based on whether the destination is a queue or topic:

- If the destination is a topic, the JMS message is immediately routed to all active subscribers to the topic. If a durable subscriber is inactive, the Message Router holds the message until the subscriber becomes active, and then delivers the message.
- If the destination is a queue, the JMS message is placed in the corresponding queue, and delivered to the appropriate consumer when the message reaches the front of the queue. The order in which messages reach the front of the queue depends on the order of their arrival and on their priority.

Once the Message Router has delivered a message to all its intended consumers, it clears the message from memory. If the message is persistent, the Message Router removes it from the broker's persistent data store.

Reliable Delivery: Acknowledgments and Transactions

The delivery mechanism just described becomes more complicated when adding requirements for *reliable* delivery. There are two aspects involved in reliable delivery:

- Assuring that delivery of messages to and from a broker is successful
- Assuring that the broker does not lose messages or delivery information before messages are actually delivered

To ensure that messages are successfully delivered to and from a broker, Message Queue uses a number of response control messages.

For example, when a producer sends a JMS message (a payload message) to a destination, the broker responds that it received the JMS message. (By default, Message Queue does this only if the producer specifies the JMS message as persistent.) The producing client uses the broker response to guarantee delivery to the destination.

Similarly, when a broker delivers a JMS message to a consumer, the consuming client sends back an acknowledgment that it has received and processed the message. A client specifies how automatically or how frequently to send these acknowledgments when creating session objects, but the Message Router does not delete a JMS message from memory until it receives an acknowledgment from each consumer to which it has delivered the message—for example, from each of the multiple subscribers to a topic.

If there are durable subscriptions to a topic, the Message Router retains each JMS message in that destination, delivering it as each durable subscriber becomes an active consumer.

The Message Router records client acknowledgments as they are received, and deletes the JMS message only after all the acknowledgments have been received, unless the JMS message expires before then.

Furthermore, the Message Router confirms receipt of the client acknowledgment by sending a broker response back to the client. The consuming client uses the broker response to make sure that the broker will not deliver a JMS message more than once. This could happen if the broker fails to receive the client acknowledgment.

If the broker does not receive a client acknowledgment and delivers a JMS message a second time, the message is marked with a Redeliver flag. The broker generally redelivers a JMS message under the following circumstances:

- The client connection closes before the broker receives a client acknowledgment, and a new connection is subsequently opened.

- The client application recovers a session.
- The client application recovers a rolled back transaction.

For example, if a message consumer of a queue goes off line before acknowledging a message, and another consumer subsequently registers with the queue, the broker redelivers the unacknowledged message to the new consumer.

The client acknowledgments and broker responses described above apply, as well, to JMS message deliveries grouped into transactions. In such cases, these processes operate on the level of a transaction as well as on the level of individual JMS message sends or receives. When a transaction commits, a broker response is sent automatically.

The broker tracks transactions, allowing them to be committed or, if they fail, rolled back. This transaction management also supports local transactions that are part of larger, distributed transactions. The broker tracks the state of these transactions until they are committed. When a broker starts up, it inspects all uncommitted transactions, and by default, the broker rolls back all transactions except those in a `PREPARED` state. If you set the `imq.transaction.autorollback` property, the broker also rolls back transactions that are in a `PREPARED` state.

Reliable Delivery: Persistence

The other aspect of reliable delivery is assuring that the broker does not lose messages or delivery information before messages are actually delivered. In general, messages remain in memory until they have been delivered or they expire. However, if the broker fails, these messages are lost.

If a producer client specifies that a message is persistent, the Message Router passes the message to a *Persistence Manager*. The Persistence Manager stores the message in a database or file system (see "Persistence Manager" on page 83) so that the message can be recovered if the broker fails.

Managing Memory Resources and Message Flow

The performance and stability of a broker depends on the system resources available and how efficiently resources such as memory are utilized. In particular, the Message Router could become overwhelmed, using up all its memory resources, when production of messages is much faster than consumption. To prevent this from happening, the Message Router uses three levels of memory protection to keep the system operating as resources become scarce:

Message limits on individual destinations You can set physical destination properties that specify limits on the number of messages and the total memory consumed by messages (see Chapter 15, "Physical Destination Property Reference"). You can also specify the behavior of the Message Router when limits are reached. The four limit behaviors are:

- Slowing message producers (`FLOW_CONTROL`)
- Throwing out the oldest messages in memory (`REMOVE_OLDEST`)
- Throwing out the lowest priority messages in memory, according to age of the messages (`REMOVE_LOW_PRIORITY`)
- Rejecting the newest messages (`REJECT_NEWEST`)

System-wide message limits System-wide message limits constitute a second line of protection. You can specify system-wide limits that apply collectively to all destinations on the system: the total number of messages and the memory consumed by all messages (see Table 14-3 on page 313). If any of the system-wide message limits are reached, the Message Router rejects new messages.

System memory thresholds System memory thresholds are a third line of protection. You can specify thresholds of available system memory at which the broker takes increasingly serious action to prevent memory overload. The action taken depends on the state of memory resources, as follows:

- `green` (plenty of memory is available)
- `yellow` (broker memory is running low)
- `orange` (broker is low on memory)
- `red` (broker is out of memory).

As the broker's memory state progresses from `green` through `yellow` and `orange` to `red`, the broker takes increasingly serious actions of the following types:

- Swapping messages out of active memory into persistent storage (see "Persistence Manager" on page 83).
- Throttling back producers of non-persistent messages, eventually stopping the flow of messages into the broker. Persistent message flow is automatically limited by the requirement that the broker acknowledge each message.

Both of these measures degrade performance.

If system memory thresholds are reached, destination message limits and system-wide message limits are too small. In some situations, the thresholds cannot catch all potential memory overloads in time. Therefore, do not rely on this feature to control memory resources, but instead configure destinations individually and collectively to optimize memory resources.

Message Router Properties

These are the system-wide limits and system memory thresholds for managing memory resources:

- `imq.destination.DMQ.truncateBody`. Specifies that the dead message queue contains only a message's header and property data. Message body contents are discarded.
- `imq.message.expiration.interval`. Specifies how often reclamation of expired messages occurs, in seconds.
- `imq.system.max_count`. Specifies the maximum number of messages held by the broker.
- `imq.system.max_size`. Specifies the maximum total size of messages held by the broker.
- `imq.message.max_size`. Specifies the maximum size of a message body.
- `imq.`*resource_state*`.threshold`. Specifies the percent memory utilization at which each memory resource state is triggered.
- `imq.`*resource_state*`.count`. Specifies the maximum number of incoming messages allowed in a batch as each memory resource state is triggered.
- `imq.transaction.autorollback`. Specifies whether distributed transactions left in a `PREPARED` state are automatically rolled back when a broker starts up.

For full descriptions of these properties, see Table 14-3 on page 313.

Persistence Manager

For a broker to recover, in case of failure, it needs to recreate the state of its message delivery operations. This requires it to save all persistent messages, as well as essential routing and delivery information, to a data store. A *Persistence Manager* component manages the writing and retrieval of this information.

To recover a failed broker requires more than simply restoring undelivered messages. The broker must also be able to do the following:

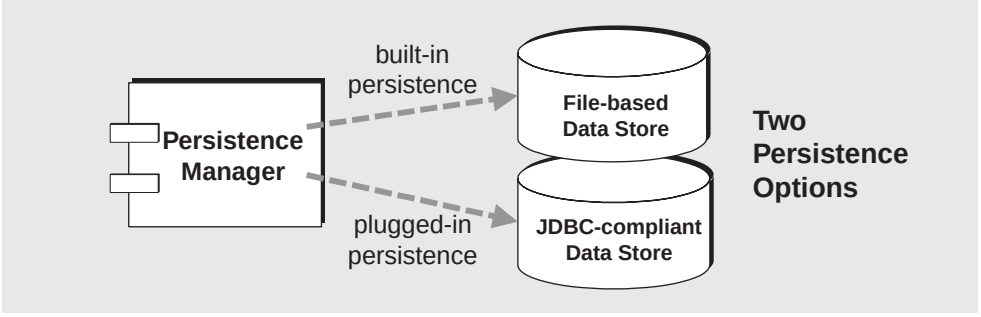
- Re-create destinations
- Restore the list of durable subscriptions for each topic
- Restore the acknowledge list for each message
- Reproduce the state of all committed transactions

The Persistence Manager manages the storage and retrieval of all this state information.

When a broker restarts, it recreates destinations and durable subscriptions, recovers persistent messages, restores the state of all transactions, and recreates its routing table for undelivered messages. It can then resume message delivery.

Message Queue supports both built-in and plugged-in persistence modules (see Figure 4-3). Built-in persistence is a file-based data store. Plugged-in persistence uses a Java Database Connectivity (JDBC™) interface and requires a JDBC data store. The built-in persistence is generally faster than plugged-in persistence; however, some users prefer the redundancy and administrative features of using a JDBC-compliant database system.

Figure 4-3 Persistence Manager Support



Built-in Persistence

The default Message Queue persistent storage solution is a file-based data store. This approach uses individual files to store persistent data, such as messages, destinations, durable subscriptions, and transactions.

The file-based data store is located in a directory identified by the name of the broker instance (*instanceName*) with which the data store is associated (see Appendix A, "Operating System-Specific Locations of Message Queue Data"):

…/`instances`/*instanceName*/`fs350`/

The file-based data store is structured so that persistent messages are stored in a directory according to the destination in which they reside. Most messages are stored in a single file consisting of variable-sized records.

To alleviate fragmentation as messages are added and removed, you can compact the variable-sized record file (see "Compacting Physical Destinations" on page 136). In addition, built-in persistence manager stores messages whose size exceeds a configurable threshold (`imq.persist.file.message.max_record_size`) in their own respective files, rather than in the variable-sized record file. For these individual files, a file pool is maintained so that files can be reused. When a message file is no longer needed, it is not deleted. Instead, the message file is added to the pool of free files in its destination directory, to be used to store new messages.

You can configure the maximum number of files in the destination file pool (`imq.persist.file.destination.message.filepool.limit`). You can also specify the percentage of free files in the file pool that are cleaned up by being truncated to zero and not simply tagged for reuse (`imq.persist.file.message.filepool.cleanratio`). As the percentage of cleaned files increases, the amount of disk space decreases and the overhead required to maintain the file pool increases.

You can specify whether or not tagged files will be cleaned up at shutdown (`imq.persist.file.message.cleanup`). If the files are cleaned up, they will take up less disk space, but the broker will take longer to shut down.

All other persistent data (destinations, durable subscriptions, and transactions) are stored in separate files. All destinations are in one file, all durable subscriptions are in another file, and so on.

To maximize reliability, you can use the `imq.persist.file.sync.enabled` attribute to specify that persistence operations should synchronize the in-memory state with the physical storage device. This helps eliminate data loss due to system crashes, but at the expense of performance. If you are running Message Queue in a Sun Cluster environment, you must set this attribute to `true` for all nodes in the cluster.

Because the data store can contain messages of a sensitive or proprietary nature, you should secure the `instances/`*instanceName*`/fs350/` directory against unauthorized access. For instructions, see "Securing Persistent Data" on page 104.

Plugged-In Persistence

You can set up a broker to access any data store accessible through a JDBC driver. This involves setting a number of JDBC-related broker configuration properties and using the database manager utility (`imqdbmgr`) to create a data store with the proper schema. The procedures and related configuration properties are detailed in "Setting Up a Persistent Store" on page 99.

Persistence Manager Properties

This property specifies what type of persistence you are using:

- `imq.persist.store`. Specifies whether the broker is using built-in, file-based (file) persistence or plugged-in JDBC-compliant (jdbc) persistence.

These properties pertain to built-in persistence:

- `imq.persist.file.sync.enabled`. Specifies whether persistence operations synchronize in-memory state with the physical storage device.
- `imq.persist.file.message.max_record_size`. Specifies the maximum size of messages that will be added to the message storage file.
- `imq.persist.file.destination.message.filepool.limit`. Specifies the maximum number of free files available for reuse in the destination file pool.
- `imq.persist.file.message. filepool.cleanratio`. Specifies the percentage of free files in destination file pools that are maintained in a *clean* state (truncated to zero).
- `imq.persist.file.message.cleanup`. Specifies whether or not the broker cleans up free files in destination file pools when it shuts down.

For full descriptions of these properties, see Table 14-6 on page 317.

These properties pertain to JDBC-based persistence:

- `imq.persist.jdbc.brokerid`. Specifies a broker instance identifier to append to the names of tables in a database used by multiple broker instances.
- `imq.persist.jdbc.driver`. Specifies the java class name of the JDBC driver to connect to the database.
- `imq.persist.jdbc.opendburl`. Specifies the database URL for opening a connection to an existing database.

- `imq.persist.jdbc.createdburl`. Specifies the database URL for opening a connection to create a database.
- `imq.persist.jdbc.closedburl`. Specifies the database URL for shutting down the current database connection when the broker is shut down.
- `imq.persist.jdbc.user`. Specifies the user name used to open a database connection, if required.
- `imq.persist.jdbc.needpassword`. Specifies whether the database requires a password for broker access.
- `imq.persist.jdbc.password`. Specifies the password for use in opening a database connection, if required.
- `imq.persist.jdbc.table.IMQSV35`. SQL command used to create the version table.
- `imq.persist.jdbc.table.IMQCCREC35`. SQL command used to create the configuration change record table.
- `imq.persist.jdbc.table.IMQDEST35`. SQL command used to create the destination table.
- `imq.persist.jdbc.table.IMQINT35`. SQL command used to create the interest table.
- `imq.persist.jdbc.table.IMQMSG35`. SQL command used to create the message table.
- `imq.persist.jdbc.table.IMQPROPS35`. SQL command used to create the property table.
- `imq.persist.jdbc.table.IMQILIST35`. SQL command used to create the interest state table.
- `imq.persist.jdbc.table.IMQTXN35`. SQL command used to create the transaction table.
- `imq.persist.jdbc.table.IMQTACK35`. SQL command used to create the transaction acknowledgment table.

For full descriptions of these properties, see Table 14-7 on page 318.

Security Manager

Message Queue provides authentication and authorization (access control) features, and also supports encryption capabilities.

The authentication and authorization features depend upon a user repository (see Figure 4-4 on page 89): a file, directory, or database that contains information about the users of the messaging system—their names, passwords, and group memberships. The names and passwords are used to authenticate a user when a connection to a broker is requested. The user names and group memberships are used, in conjunction with an access control file, to authorize operations such as producing or consuming messages for destinations.

Message Queue administrators populate a Message Queue-provided user repository (see "Using a Flat-File User Repository" on page 142), or plug a pre-existing LDAP user repository into the Security Manager component (see "Using an LDAP Server for a User Repository" on page 149).

Authentication

Message Queue security supports password-based authentication. When a client requests a connection to a broker, the client must submit a user name and password.

The Security Manager compares the name and password submitted by the client to those stored in the user repository. On transmitting the password from client to broker, the passwords are encoded using either base 64 encoding or message digest (MD5). For more secure transmission, see "Encryption" on page 90. You can separately configure the type of encoding used by each connection service or set the encoding on a broker-wide basis.

All Security Manager properties are listed under "Security Manager Properties" on page 90 and described in detail under "Security Manager Properties" on page 90.

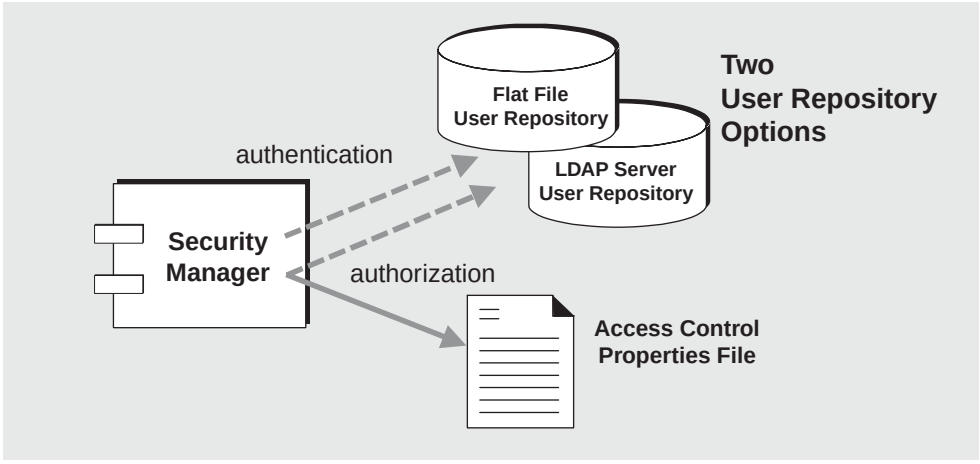
Authorization

Once the user of a client application has been authenticated, the user can be authorized to perform various Message Queue-related activities. The Security Manager supports both user-based and group-based access control. Depending on a user's name or the groups to which the user is assigned in the user repository, that user has permission to perform certain Message Queue operations. You specify these access controls in an access control properties file (see Figure 4-4).

When a user attempts to perform an operation, the Security Manager 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 for the following operations:

- Establishing a connection with a broker
- Accessing destinations: creating a consumer, a producer, or a queue browser for any given destination or all destinations
- Auto-creating destinations

Figure 4-4 Security Manager Support



The default access control properties file explicitly references only one group: *admin* (see "Groups" on page 145). A user in the *admin* group has admin service connection permission. The admin service lets the user 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.

As a Message Queue administrator you can define groups and associate users with those groups in a user repository (though groups are not fully supported in the flat-file user repository).

By editing the access control properties file, you can specify access to destinations by users and groups for the purpose of producing and consuming messages, or browsing messages in queue destinations. You can make individual destinations or all destinations accessible only to specific users or groups. If the broker is configured to allow auto-creation of destinations, you can edit the access control properties file to control the users and groups for whom the broker can auto-create destinations.

All Security Manager properties are listed under "Security Manager Properties" on page 90 and described in detail under "Security Manager Properties" on page 90.

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 a connection level by establishing an encrypted connection between an SSL-enabled broker and an SSL-enabled client.

To use a Message Queue 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 and places it in a Message Queue keystore. The Message Queue keystore is, itself, password protected; to unlock it, you must provide a keystore password at startup time. See "Working With an SSL-Based Service" on page 159.

Once the keystore 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.

All Security Manager properties are listed in the next section and described in detail under "Security Manager Properties" on page 90.

Security Manager Properties

These are the configurable properties for authentication, authorization, encryption, and other secure communications:

- `imq.authentication.type`. Specifies whether the password should be passed in base 64 coding (`basic`) or as an MD5 digest (`digest`).
- `imq.`*service_name*`. authentication.type`. Specifies whether the password should be passed in base 64 coding (`basic`) or as an MD5 digest (`digest`).
- `imq.authentication.basic.user_repository`. For base 64 coding, specifies the type of user repository used for authentication, either file-based or LDAP.

- `imq.authentication.client.response.timeout`. Specifies the time (in seconds) the system will wait for a client to respond to an authentication request from the broker.
- `imq.accesscontrol.enabled`. Indicates whether the system will check whether an authenticated user has permission to use a connection service or to perform specific Message Queue operations with respect to specific destinations, as specified in the access control properties file.
- `imq.`*service_name*`.accesscontrol.enabled`. Sets access control (`true/false`) for the named connection service, overriding the broker-wide setting.
- `imq.accesscontrol.file.filename`. Specifies the name of an access control properties file for all connection services supported by a broker instance.
- `imq.`*service_name*`.accesscontrol.file.filename`. Specifies the name of an access control properties file for a named connection service of a broker instance.
- `imq.passfile.enabled`. Specifies whether user passwords (for SSL, LDAP, JDBC™) for secure communications are specified in a file.
- `imq.passfile.dirpath`. Specifies the path to the directory containing the passfile.
- `imq.passfile.name`. Specifies the name of the passfile.
- `imq`.keystore.*property_name*. For SSL-based services: specifies security properties relating to the SSL keystore. See Table 14-9 on page 324.

For full descriptions of these properties, see Table 14-8 on page 320.

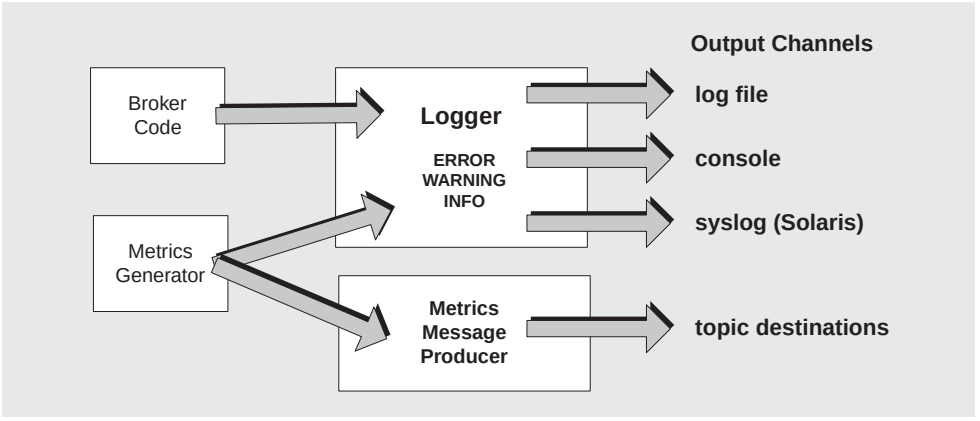
Monitoring Service

The broker includes a number of components for monitoring and diagnosing its operation. Among these are the following:

- Components that generate data (broker code that logs events and a metrics generator)
- A logger component (see "Logger") that writes out information through a number of output channels
- A message producer that sends JMS messages containing metrics information to topic destinations for consumption by JMS monitoring clients.

The general scheme is illustrated in Figure 4-5.

Figure 4-5 Monitoring Service 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 connections open, and the number of threads being used.

You can turn the generation of metrics data on and off, and specify how frequently metrics reports are generated.

Logger

The Message Queue logger takes information generated by broker code 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 the Solaris™ operating system, 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.

For example, you can specify the logger level to determine the type of information that the logger gathers: errors (`ERROR`); errors and warnings (`WARNING`); or errors, warnings, and information (`INFO`).

For each output channel, you can specify which of the categories set for the logger will be written to that channel. For example, if the logger level is set to `INFO`, you can specify that you want only errors and warnings written to the console, and only info (metrics data) written to the log file.

If you are using a log file, you can specify the point at which the log file is closed and output is rolled over to a new file. An archive of the nine most recent log files is retained as new rollover log files are created.

For information on configuring the logger, see "Configuring and Using Broker Logging" on page 205. For information on configuring and using the Solaris `syslog`, see the `syslog(1M)`, `syslog.conf(4)` and `syslog(3C)` man pages.

Metrics Message Producer (Enterprise Edition)

The Message Producer component receives information from the Metrics Generator component at regular intervals. It writes the information into messages, which it then sends to metric topic destinations. The destination to which a metrics message is sent depends on the type of information it contains.

There are five metrics topic destinations, whose names are shown in Table 4-3, along with the type of metrics messages delivered to each destination.

Table 4-3 Metrics Topic Destinations

Topic Destination 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

Message Queue clients that subscribe to these metric topic destinations consume the messages in the destinations and process the metrics information. For example, a client can subscribe to the `mq.metrics.broker` destination to receive and process information such as the total number of messages in the broker.

The Metrics Message Producer is an internal Message Queue client that creates messages (of type `MapMessage`) that contain name-value pairs corresponding to metrics data. These messages are produced only if there are one or more subscribers to the corresponding metrics topic destination.

The messages produced by the Metrics Message Producer are of type `MapMessage`. They consist of a number of name/value pairs, depending on the type of metrics they contain. Each name/value pair corresponds to a metric quantity and its value.

As an example, broker metrics messages contain values for the number of messages that have flowed into and out of the broker, the size of these messages, the number and size of messages currently in memory, and so forth. For details of the metrics quantities reported in each type of metrics message, see the *Message Queue Developer's Guide for Java Clients*. That manual explains how to write a Message Queue client for consuming metrics messages.

In addition to the metrics information contained in the body of a metrics message, the header of each message has properties that provide the following information:

- Message type
- Host, port, and address of the broker that sent the message
- Time that the metric sample was taken

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

Monitoring Service Properties

These are the configurable properties for setting the generation, logging, and metrics message production of information by the broker:

- `imq.metrics.enabled`. Specifies whether metrics information is being written to the logger.
- `imq.metrics.interval`. If metrics logging is enabled, specifies the time interval, in seconds, at which metrics information is written to the logger.
- `imq.log.level`. Specifies the logger level: the categories of output that can be written to an output channel.
- `imq.log.file.output`. Specifies which categories of logging information are written to the log file.
- `imq.log.file.dirpath`. Specifies the path to the directory containing the log file.
- `imq.log.file.filename`. Specifies the name of the log file.
- `imq.log.file.rolloverbytes`. Specifies the size, in bytes, of the log file at which output rolls over to a new log file.
- `imq.log.file.rolloversecs`. Specifies the age, in seconds, of log file at which output rolls over to a new log file.
- `imq.log.console.output`. Specifies which categories of logging information are written to the console.

- `imq.log.console.stream`. Specifies whether console output is written to stdout (`OUT`) or stderr (`ERR`).
- `imq.log.syslog.facility`. (Solaris only) Specifies what `syslog` facility the Message Queue broker should log as.
- `imq.log.syslog.logpid`. (Solaris only) Specifies whether to log the broker process ID with the message.
- `imq.log.syslog.logconsole`. (Solaris only) Specifies whether to write messages to the system console if they cannot be sent to `syslog`.
- `imq.log.syslog.identity`. (Solaris only) Specifies the identity string that should be prepended to every message logged to `syslog`.
- `imq.log.syslog.output`. (Solaris only) Specifies which categories of logging information are written to `syslog`d(1M).
- `imq.log.timezone`. Specifies the time zone for log time stamps.
- `imq.metrics.topic.enabled`. Specifies whether metrics message production is enabled.
- `imq.metrics.topic.interval`. Specifies the time interval, in seconds, at which metrics messages are produced.
- `imq.metrics.topic.persist`. Specifies whether or not metrics messages are persistent.
- `imq.metrics.topic.timetolive`. Specifies the lifetime, in seconds, of metrics messages sent to metric topic destinations.
- `imq.destination.logDeadMsgs`. Specifies whether the broker writes a message to the log each time it discards a dead message or puts a dead message on the dead message queue.

For full reference information about these properties, see Table 14-10 on page 324.

About Configuration Files

Broker configuration files are used to configure the broker. Appendix A, "Operating System-Specific Locations of Message Queue Data" lists the directory where these files are located for your operating system.

The directory stores the following files:

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

Instance Configuration File

The first time you run a broker, an instance configuration file is created. Use the instance configuration file to specify configuration properties for that instance of the broker.

The instance configuration file is stored in a directory that is identified by the name of the broker instance (*instanceName*) with which the configuration file is associated:

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

See Appendix A, "Operating System-Specific Locations of Message Queue Data" for the location of the `instances` directory.

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. It is modified when you make configuration changes using administration tools. You can also edit an instance configuration file by hand to make configuration changes (see "Editing the Instance Configuration File" on page 98). To do so, you must be the owner of the …`/instances/`*instanceName* directory or log in as root to change privileges on the directory.

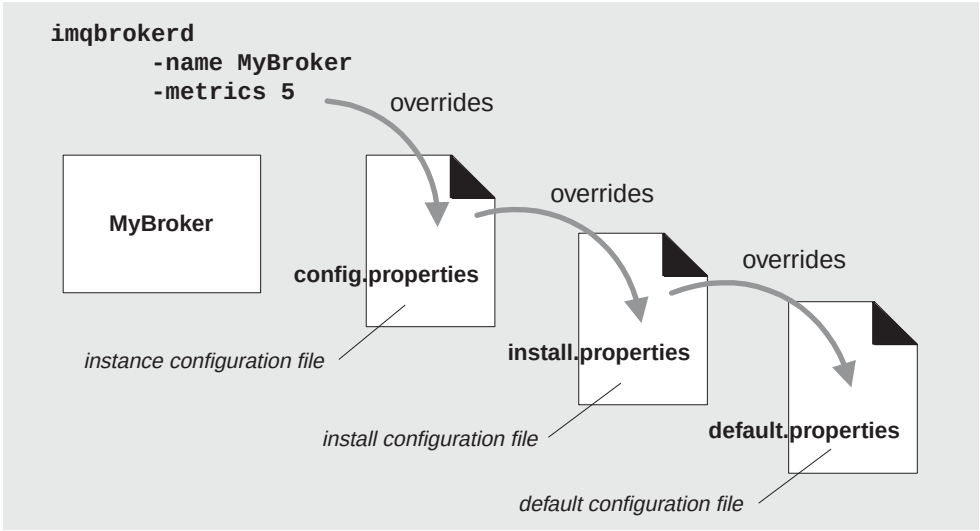
If you connect broker instances in a cluster, you may also need to use a *cluster configuration file* to specify cluster configuration information. For more information, see "Cluster Configuration Properties" on page 327.

Merging Property Values

At startup, the broker merges property values in the different configuration files. It uses values in the installation and instance configuration files to override values specified in the default configuration file.

You can override the resulting values by using `imqbrokerd` command options. This scheme is illustrated in Figure 4-6.

Figure 4-6 Broker Configuration Files



Property Naming Syntax

Any Message Queue property definition in a configuration file uses the following naming 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 (86400 seconds):

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

Chapter 14, "Broker Properties Reference" on page 307 lists the broker configuration properties and their default values.

Editing the Instance Configuration File

The first time a broker instance is run, a `config.properties` file is automatically created. You can edit this instance configuration file to customize the behavior and resource use of the corresponding broker instance.

The broker instance reads the `config.properties` file only at startup. To make permanent changes to the `config.properties` file, you can do one of the following:

- Use administration tools. For information about properties you can set using `imqcmd`, see Table 14-1 on page 308.
- Edit the `config.properties` file while the broker instance is shut down; then restart the instance. (On Solaris and Linux operating systems, only the user that first started the broker instance has permission to edit the `config.properties` file.)

Table 14-1 lists the broker instance configuration properties in alphabetical order, with their default values. For more information about the meaning and use of each property, please consult the specified cross-referenced section.

Entering Configuration Options on the Command Line

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

At startup time, you use the `imqbrokerd` command to start a broker instance. Using the command's `-D` option, you can specify any broker configuration property and its value. If you start the broker as a Windows service, using the `imqsvcadmin` command, you use the `-args` option to specify startup configuration properties.

You can also set certain broker properties when a broker instance is running. To modify the configuration of a running broker, you use the `imqcmd update bkr` command.

For more information about startup configuration, see Chapter 3, "Starting Brokers and Clients," particularly the examples under "Starting Brokers Interactively" on page 67.

For information about modifying the configuration of a running broker, see Chapter 5, "Managing a Broker" and Chapter 14, "Broker Properties Reference."

Setting Up a Persistent Store

Message Queue brokers include a Persistence Manager component that manages the writing and retrieval of persistent information. The Persistence Manager is configured by default to access a built-in, file-based data store, but you can reconfigure it to plug in any data store accessible through a JDBC-compliant driver.

The Message Queue data store contains information about transactions, messages, durable subscriptions, and physical destinations. It also contains information about the state of messages with respect to acknowledgments.

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

Configuring a File System Store

A file system data store is automatically created when you create a broker instance. The store is located under the instance directory for that broker. The location is operating system-specific; for the exact location of the persistent store, see Appendix A, "Operating System-Specific Locations of Message Queue Data."

By default, Message Queue performs non-synchronous write operations to disk. The operating system can buffer these operations to provide for good performance. However, if an unexpected system failure occurs between write operations, messages could be lost. To improve reliability, you can cause Message Queue to perform synchronous writes to disk, but be aware that this option causes reduced performance. To specify synchronous writes to disk, set the broker property `imq.persist.file.sync`. For details about this property, see Table 14-6 on page 317.

When you start a broker instance, you can use the `imqbrokerd -reset` option to clear the file system store. For more information about this option and its suboptions, see Table 13-2 on page 282

Configuring a JDBC Store

To configure a broker to use JDBC-based persistence, you set JDBC-related properties in the broker 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.

The procedure described in this chapter is illustrated using, as an example, the PointBase DBMS bundled with the Java 2 Platform, Enterprise Edition (J2EE) SDK. Version 1.4 is available for download from `java.sun.com`. The example uses PointBase's embedded version (instead of the client/server version). In the procedures, instructions are illustrated using path names and property names from the PointBase example. They are identified by the word "Example:"

Example configurations for Oracle and PointBase are available. To find the example files, see Appendix A, "Operating System-Specific Locations of Message Queue Data." In the table that lists information for your operating system, look for the location of "Example applications and configurations."

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`.

Plugging In a JDBC-Accessible Data Store

It takes just a few steps to plug in a JDBC-accessible data store.

➤ **To Plug in a JDBC-Accessible Data Store**

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

   See the properties documented in "JDBC-Based Persistence" on page 317

2. Place a copy or a symbolic link to your JDBC driver jar file located in the following path:

   `/usr/share/lib/imq/ext/` (Solaris)

   `/opt/sun/mq/share/lib/` (Linux)

   `IMQ_VARHOME\lib\ext` (Windows)

   Copy Example (Solaris):

   ```
   % cp j2eeSDK_install_directory/pointbase/lib/pointbase.jar
   /usr/share/lib/imq/ext
   ```

   Symbolic Link Example (Solaris):

   ```
   % ln -s j2eeSDK_install_directory/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 (imqdbmgr)" on page 104.

   Example:

   a. Change to 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`/*dabatabseName*.

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 (see "Database Manager Utility (imqdbmgr)" on page 104).

JDBC-Related Broker Properties

The broker's instance configuration file is located in a directory identified by the name of the broker instance with which the configuration file is associated (see Appendix A, "Operating System-Specific Locations of Message Queue Data"):

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

If the file does not yet exist, you must start the broker by using the `-name` *instanceName* option, so that Message Queue can create the file.

"JDBC-Based Persistence" on page 317 presents the configuration properties that you need to set when plugging in a JDBC- accessible data store. There is a summary of these properties at the end of this section. You set these properties in the instance configuration file (`config.properties`) of each broker instance that uses plugged-in persistence.

The instance configuration properties enable you to customize the SQL code that creates the Message Queue database schema: there is a configurable property that specifies the SQL code that creates each database table. These properties are needed to properly specify the data types used by the plugged-in database.

Since there are incompatibilities between database vendors with respect to the exact SQL syntax, be sure to check the corresponding documentation from your database vendor and adjust the properties in Table 14-7 on page 318 accordingly. For example, for the PointBase database, you may need to adjust the maximum length allowed for the MSG column (see the `imq.persist.jdbc.table.IMQMSG35` property) in the IMQMSG35 table.

As with all broker configuration properties, values can be set using the `-D` command line option. If a database requires certain database specific properties to be set, these also can be set using the `-D` command line option when starting the broker (`imqbrokerd`) or the Database Manager utility (`imqdbmgr`).

Example:

For the PointBase embedded database example, instead of specifying the absolute path of a database in database connection URLs, you can use the `-D` command line option to define the PointBase system directory:

```
-Ddatabase.home=IMQ_VARHOME/instances/instanceName/dbstore
```

In that case, you can specify the URL to create a database as follows:

```
imq.persist.jdbc.createdburl=jdbc:pointbase:embedded:dbName;new
```

You can specify the URL to open a database as follows

```
imq.persist.jdbc.opendburl=jdbc:pointbase:embedded:dbName
```

This is a summary of the JDBC-related properties:

- `imq.persist.store`. Specifies a file-based or JDBC-based data store.
- `imq.persist.jdbc.brokerid`. Specifies a broker instance identifier that is appended to database table names to make them unique.
- `imq.persist.jdbc.driver`. Specifies the java class name of the JDBC driver to connect to the database.
- `imq.persist.jdbc.opendburl`. Specifies the database URL for opening a connection to an existing database.
- `imq.persist.jdbc.createdburl`. Specifies the database URL for opening a connection to create a database.
- `imq.persist.jdbc.closedburl`. Specifies the database URL for shutting down the current database connection when the broker is shut down.
- `imq.persist.jdbc.user`. Specifies the user name used to open a database connection, if required.
- `imq.persist.jdbc.needpassword`. Specifies whether the database requires a password for broker access.
- `imq.persist.jdbc.password`. Specifies the password for use in opening a database connection, if required.
- `imq.persist.jdbc.table.IMQSV35`. SQL command used to create the version table.
- `imq.persist.jdbc.table.IMQCCREC35`. SQL command used to create the configuration change record table.

- `imq.persist.jdbc.table.IMQDEST35`. SQL command used to create the destination table.
- `imq.persist.jdbc.table.IMQINT35. SQL` command used to create the interest table.
- `imq.persist.jdbc.table.IMQMSG35`. SQL command used to create the message table.
- `imq.persist.jdbc.table.IMQPROPS35`. SQL command used to create the property table.
- `imq.persist.jdbc.table.IMQILIST35`. SQL command used to create the interest state table.
- `imq.persist.jdbc.table.IMQTXN35`. SQL command used to create the transaction table.
- `imq.persist.jdbc.table.IMQTACK35`. SQL command used to create the transaction acknowledgment table.

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

Database Manager Utility (imqdbmgr)

Message Queue provides a Database Manager utility (`imqdbmgr`) for setting up the schema needed for persistence. You can also use the utility to delete Message Queue database tables if the tables become corrupted or if you want to use a different database as a data store.

For reference information about the syntax, subcommands, and options of the `imqdbmgr` command, see Chapter 13, "Command Reference."

Securing Persistent Data

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

Built-In (File-Based) Persistent Store

A broker using built-in persistence writes persistent data to a flat file data store whose location is operating system-specific (see Appendix A, "Operating System-Specific Locations of Message Queue Data"):

...`/instances/`*instanceName*`/fs350/`

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

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

Solaris and Linux The permissions on the `IMQ_VARHOME/instances/`*instanceName*`/filestore/` directory depend on the umask of the user that started the broker instance. Hence, permission to start a broker instance and to read its persistent files can be restricted by appropriately setting the umask. Alternatively, an administrator (superuser) can secure persistent data by setting the permissions on the `IMQ_VARHOME/instances` directory to 700.

Windows The permissions on the `IMQ_VARHOME/instances/`*instanceName*`/filestore/` directory can be set using the mechanisms provided by the Windows operating system that you are using. This generally involves opening a properties dialog for the directory.

Plugged-In (JDBC) Persistent Store

A broker using plugged-in persistence writes persistent data to a JDBC Compliant database.

For a database managed by a database server (for example, an Oracle database), 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 database vendor for documentation 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 (see *Message Queue Administration Guide*, Appendix A, "Setting Up Plugged-in Persistence").

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 in "Built-In (File-Based) Persistent Store." To ensure that the database is readable and writable by both the broker and the `imqdbmgr` utility, however, both should be run by the same user.

Chapter 5

Managing a Broker

This chapter explains how to perform basic tasks related to managing the broker and its services. This chapter has the following sections:

This chapter does not cover all topics related to managing a broker. Additional large 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` commands to manage the broker. Before managing the broker, you must do the following:

- Start the broker using the `imqbrokerd` command. You cannot use the other commands 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 159.

Using the imqcmd Command Utility

The `imqcmd` command 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 Reference" on page 279. Reference information for use in managing physical destinations is in a separate chapter, Chapter 15, "Physical Destination Property Reference" on page 329.

Specifying the User Name and Password

Because each `imqcmd` command is authenticated against the user repository, it requires a user name and password. The only exceptions are as follows:

- Commands that use the `-h` or `-H` option to display help commands.
- 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.

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 by 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 passfile.
- Let the command prompt you for the password. This is the most secure method of specifying a password, unless other people can see what you are typing.

In previous versions of Message Queue, you could use the `-p` option to specify a password on the 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 to 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 `myserver` at port `1564`. The user name is `aladdin`. This command requires that the user repository is updated so that the user name `aladdin` is assigned 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 Help

To display help on the `imqcmd` command 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 command 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 command utility processes only the `-v` option. All other items on the command line are ignored.

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:port
```

This subcommand lists the current settings of properties of 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:port*`]-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.

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

For example, the following command turns off the auto-creation of queue destinations:

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

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, 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. For example, if a particular physical destination is bombarded with messages, you can pause the broker and take actions that might help you fix the problem, such as:

- Trace the source of the messages
- Limit the size of the physical destination
- Destroy the physical destination.

Pausing a broker also does not suspend the `cluster` connection service. However message delivery within a cluster depend on the delivery functions performed by the different brokers in the cluster.

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

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

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 120 and "Pausing and Resuming Physical Destinations" on page 133.

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:port]
```

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:port]
```

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
```

You can shut down and restart the broker. This is the syntax of the `restart bkr` subcommand:

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

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.

The following command restarts the broker running on `localhost` at port `7676`:

```
imqcmd restart bkr -u admin
```

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:port]
       [-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 description of the use of `imqcmd` to report broker metrics, see "Broker-wide Metrics" on page 350.

Managing Connection Services

The Command 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. The values in the Service Name column are the values you use to specify a service name for the `-n` option. As shown in the table, each service is associated with a service type it uses (`NORMAL` for application clients or `ADMIN` for administration clients) and 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)

Table 5-1 Connection Services Supported by a Broker *(Continued)*

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

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:port]
```

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

Use the subcommand in a command line like the following:

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

For example, the following command lists the services available for the broker running on the host `myServer` on port `6565`.

```
imqcmd list svc -b MyServer:6565 -u admin
```

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:port]
```

The subcommand 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:port]
        -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 Service Properties" on page 311.

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:port] [-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 352.

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:port]
```

The subcommand pauses the specified service running on the default broker or on a broker at the specified host and port. You cannot pause the admin service.

Use a command line like the following:

```
imqcmd pause svc -n serviceName -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:port]
```

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

Use a command line like the following:

```
imqcmd resume svc -n serviceName -u admin
```

Getting Information About Connections

The Command 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:port]
```

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:

```
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:port]
```

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

You might need to use `imqcmd` subcommands to manage a broker's durable subscriptions. 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 `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 client_id
```

Use the name of the subscription and the client ID to identify the subscription. For example:

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

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 client_id
```

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 14-3 on page 313), 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 transaction_id
```

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

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

This is the output produced by the command:

```
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
6469706F6C7369646577696E6465723130313234313431313030373230
```

The `commit` and `rollback` subcommands can be used 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 transaction_id
```

For example:

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

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

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

```
imqcmd rollback txn -n transaction_id
```

See the `imq.transaction.autorollback` property in Table 14-3 on page 313 for more information.

Chapter 6

Managing 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 explains how to perform the following tasks:

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

NOTE A client application uses a `Destination` object whenever it interact 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 imqcmd Command Utility

The `imqcmd` command utility enables you to manage physical destinations. The syntax of `imqcmd` command is the same as it is 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 Reference" on page 279.

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 Subcommands" on page 290.

Table 6-1 Physical Destination Subcommands for the imqcmd Command Utility

Subcommand and Argument	Description
`compact dst`	Compacts the built-in 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.
`query dst`	Queries and displays information on a physical destination.

Table 6-1 Physical Destination Subcommands for the imqcmd Command Utility

Subcommand and Argument	Description
`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=value*`1]`...

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."
- Any nondefault values for the physical destination's properties.

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

Many physical destination properties manage 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 broker-wide limits that broker configuration properties control.

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 329 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 14-4 on page 314.

➤ **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. To do so, use 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 have the value `q` (queue) or `t` (topic).

If the destination type is omitted, physical destinations of all types are listed.

The `list dst` subcommand can optionally specify the type of destination to list or optionally 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
```

The dead message queue, `mq.sys.dmq`, always appears, 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 property values, 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:

```
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 133).

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 described 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 138.

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.

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 queued 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 the built-in file-based data store (as opposed to a plugged-in JDBC-compliant 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 according 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 built-in 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 329).

➤ **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 autocreated 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

The `imqcmd` command utility manages the dead message queue. You 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.

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 a complete message on the dead message queue, or 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 queue size and if you do not plan to restore dead messages, consider discarding the body contents.

To discard the body contents and retain only the headers and property data, set the `imq.destination.DMQ.truncateBody` broker property to `true`, as the following example shows:

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

Enabling Dead Message Logging

In addition to standard queue monitoring and logging options, you can log the messages that a broker has classified as dead.

If dead message logging is enabled, the broker logs the following types of 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.

Dead message logging is disabled by default. The following example 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 for use in authenticating users; define access control; configure a Secure Socket Layer (SSL) connection service that encrypts client-broker communication; and set up a passfile for use in broker startup.

The chapter includes the following sections:

Authenticating Users

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.

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.

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 149.

Using a Flat-File User Repository

Message Queue provides a flat-file user repository and a command line tool, Message Queue User Manager (`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 Message Queue User Manager utility (`imqusermgr`) 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 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, "Operating System-Specific Locations of Message Queue Data"):

.../`instances`/*instanceName*/`etc`/`passwd`

The repository is created with two entries (rows), as illustrated in Table 7-1, below.

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. Initial user/password setup is not required for the Message Queue broker to be used.

The initial `guest` user entry allows clients to connect to a broker instance using the default `guest` user name and password (for testing purposes, for example).

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 148).

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

User Manager Utility (imqusermgr)

The 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 Reference."

Before using `imqusermgr`, 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`.

add Subcommand 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]`

delete Subcommand 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]`

list Subcommand 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*`]`

update Subcommand 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` *active_state*	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 user repository 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.

- **`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 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 152.

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 (:), new line or carriage return.
- 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 only known to you. You need to use the `imqusermgr` tool to do this.

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. The broker queries the LDAP server whenever a user attempts to connect to the broker instance or perform certain 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, "Operating System-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 base64 encoding (`basic`) or in MD5 digest (`digest`). 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 described in and summarized later in this section.

   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 152.

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`.

These are the LDAP-related properties:

- `imq.user_repository.ldap.server`. The *host:port* for the LDAP server
- `imq.user_repository.ldap.principal`. The distinguished name that the broker will use to bind to the directory server for a search.
- `imq.user_repository.ldap.password`. The password associated with the distinguished name used by the broker.
- `imq.user_repository.ldap.base`. The directory base for user entries.

- `imq.user_repository.ldap.uidattr`. The provider-specific attribute identifier whose value uniquely identifies a user. For example: `uid`, `cn`.
- `imq.user_repository.ldap.usrfilter`. A JNDI search filter to use with users.
- `imq.user_repository.ldap.grpsearch`. A boolean specifying whether you want to enable group searches.
- `imq.user_repository.ldap.grpbase`. The directory base for group entries.
- `imq.user_repository.ldap.gidattr`. The provider-specific attribute identifier whose value is a group name.
- `imq.user_repository.ldap.memattr`. The attribute identifier in a group entry whose values are the distinguished names of the group's members.
- `imq.user_repository.ldap.grpfiltler`. A JNDI search filter to use with groups.
- `imq.user_repository.ldap.timeout`. An integer specifying (in seconds) the time limit for a search.
- `imq.user_repository.ldap.ssl.enabled`. A boolean specifying whether the broker should use the SSL protocol when talking to an LDAP server.

For full reference information about these properties, see "Security Manager Properties" on page 320.

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, "Operating System-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.

A broker checks its ACL file when a client application performs one of the following operations:

- Create a connection
- Create a producer

- Create a consumer
- Browse 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 that the broker checks the file to verify authorization. You need not restart the broker after editing the file.

The ACL file is used whether user information is placed in a flat-file user repository (see "Using a Flat-File User Repository" on page 142) or in an LDAP user repository (see "Using an LDAP Server for a User Repository" on page 149).

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, "Operating System-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 145.
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 her 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 `imqusermgr`. 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 is 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 a 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.

Table 7-4 Elements of Physical Destination Access Control Rules *(Continued)*

Component	Description
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` 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 369.

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 199).

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 166 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 `imqkeytool` command to generate a self-signed certificate for the broker. On UNIX® systems you may need to run `imqkeytool` as the superuser (`root`) in order to have permission to create the keystore.

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 keystore 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`
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, imqkeytool 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, `imqkeytool` pauses while it generates a key pair.

Next, `imqkeytool` 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 keystore password.

NOTE Remember the password you provide. You must provide this password when you start the broker, to allow the broker to open the keystore. You can store the keystore password in a passfile (see "Using a Passfile" on page 169).

Running `imqkeytool` runs the JDK `keytool` utility to generate a self-signed certificate and places it in Message Queue's keystore, located in a directory that depends upon the operating system, as shown in Appendix A, "Operating System-Specific Locations of Message Queue Data."

The keystore is in the same format as that supported by the JDK1.2 `keytool` utility.

These are the configurable properties for the Message Queue keystore:

- `imq.keystore.file.dirpath`. For SSL-based services: specifies the path to the directory containing the keystore file. For the default value, see Appendix A, "Operating System-Specific Locations of Message Queue Data."
- `imq.keystore.file.name`. For SSL-based services: specifies the name of the keystore file.
- `imq.keystore.password`. For SSL-based services: specifies the keystore password.

You might need to regenerate a key pair in order to solve certain problems; for example:

- You forgot the keystore 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 keystore password when you generated the self-signed certificate in "Step 1. Generating a Self-Signed Certificate" on page 160.

➤ **To Regenerate a Key Pair**

1. Remove the broker's keystore, located as shown in Appendix A, "Operating System-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 160.

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 199.

➤ **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, "Operating System-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 keystore 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 passfile, as described in "Using a Passfile" on page 169. Once you have put the password in the passfile and set the property `imq.passfile.enabled=true`, do one of the following:
  - Pass the location of the passfile to the `imqbrokerd` command:

    `imqbrokerd -passfile` */tmp/mypassfile*
  - Start the broker without the `-passfile` option, but specify the location of the passfile 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:port* -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 148).

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 keystore, 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 160. 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 keystore.

   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` keystore.

   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 keystore 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` keystore 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 164.

   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 truststore, as described in the following sections.

There are three options for configuring the client with a truststore:

- 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 keystore file and configure the client to use that as its truststore.

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 keystore 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 keystore 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 keystore by default, so you must provide the location of the truststore 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 Passfile

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`	Starts the broker	Access a plugged-in persistent data store, an SSL certificate keystore, or an LDAP user repository
`imqcmd`	Manages the broker	Authenticate an administrative user who is authorized to use the command
`imqdbmgr`	Manages a plugged-in data store	Access the data store

You can specify these passwords in a password file (*passfile*) 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 this 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 passfile on the command line. For non-interactive use of commands, however, you must use a passfile.

A passfile 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.

Passfile Contents

A passfile is a simple text file that contains a set of properties and values. Each value is a password used by a command.

A passfile can contain the passwords shown in Table 7-7:

Table 7-7 Passwords in a Passfile

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 keystore 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 passfile is part of the Message Queue product. For the location of the sample file, see Appendix A, "Operating System-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`.

For reference information about the `imq.audit.enabled` property, see "Security Manager Properties" on page 320.

Chapter 8

Managing Administered Objects

The use of administered objects enables the development of client applications that are portable to other JMS providers. Administered objects encapsulate provider-specific configuration and naming information.

A Message Queue administrator typically creates administered objects for client applications to use in obtaining broker connections. A client application uses a connection to send messages to physical destinations and receive messages from physical destinations.

This chapter explains how you use the Object Manager utility (`imqobjmgr`) to perform these tasks. Because these tasks involve an understanding of the attributes of both the object store you are using and of the administered objects you are creating, this chapter provides background on these two topics before describing how to use `imqobjmgr` to manage administered objects.

This chapter contains the following sections:

About Object Stores

Administered objects are placed in a readily available object store where they can be accessed by client applications through a JNDI lookup. There are two types of object stores you can use: a standard LDAP directory server or a file-system object store.

LDAP Server Object Store

An LDAP server is the recommended object store for production messaging systems. LDAP implementations are available from a number of vendors and are designed for use in distributed systems. LDAP servers also provide security features that are useful in production environments.

Message Queue administration tools can manage object stores on LDAP servers. However, you might first need to configure the LDAP server to store java objects and perform JNDI lookups, as prescribed in the documentation for the LDAP server.

In using an LDAP server as your object store, you need to specify the attributes shown in Table 8-1. These attributes fall into the following categories:

- **Initial Context**: This attribute is fixed for an LDAP server object store.
- **Location**: Specifies the URL and directory path for storing your administered objects, as set up in the LDAP server. In particular you must check that the specified path exists.
- **Security Information**: Depends on the LDAP provider. You should consult the documentation provided with your LDAP implementation to determine whether security information is required on all operations or only on operations that change the stored data.

Table 8-1 LDAP Object Store Attributes

Attribute	Description
`java.naming.factory.initial`	The initial context for a JNDI lookup on an LDAP server `com.sun.jndi.ldap.LdapCtxFactory`
`java.naming.provider.url`	LDAP server URL and directory path information. For example: `ldap://mydomain.com:389/ou=mqobjs,o=myapp` where administered objects are stored in the `/myapp/mqobjs` directory.

Table 8-1 LDAP Object Store Attributes *(Continued)*

Attribute	Description
`java.naming.security.principal`	The identity of the principal for authenticating the caller to the LDAP server.The format of this entry depends on the authentication scheme.For example: `uid=fooUser, ou=People, o=mq` If this property is unspecified, the behavior is determined by the LDAP service provider.
`java.naming.security.credentials`	The credentials of the principal for authenticating the caller to the LDAP server. The value of the property depends on the authentication scheme: it could be a hashed password, clear-text password, key, certificate, and so on. For example: `fooPasswd` If this property is unspecified, the behavior is determined by the LDAP service provider.
`java.naming.security.authentication`	Security level to use. Its value is one of the following key words: `none`, `simple`, `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 Store

Message Queue also supports a file-system object store implementation. While the file-system object store is not fully tested and is therefore not recommended for production systems, it has the advantage of being very easy to use in development environments. Rather than setting up an LDAP server, all you must do is create a directory on your local file system.

However a file-system store cannot be used as a centralized object store for clients deployed across multiple computer nodes unless these clients have access to the directory where the object store resides. In addition, any user with access to that directory can use Message Queue administration tools to create and manage administered objects.

In using a file-system object store, you need to specify the attributes shown in Table 8-2. These attributes fall into the following categories:

- **Initial Context**: The value of this attribute is fixed for a file system object store.
- **Location**: The value of this attribute specifies the directory path for storing your administered objects. The 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`	The initial context for a JNDI lookup on a file system object store: `com.sun.jndi.fscontext.RefFSContextFactory`
`java.naming.provider.url`	Directory path information. For example: `file:///C:/myapp/mqobjs`

About Administered Object Attributes

Message Queue administered objects are of two basic kinds:

- *Connection factory* administered objects are used by client applications to create connections to brokers.
- *Destination* administered objects are used by client applications to identify destinations to which producers send messages or from which consumers retrieve messages.

Message Queue provides two administration tools for creating and managing administered objects: the command line Object Manager utility (`imqobjmgr`) and the GUI Administration Console. This chapter describes only how to use the command line.

The attributes of an administered object are specified using attribute-value pairs.

Connection Factory Attributes.

The configuration of a connection factory passes to all the connections that the connection factory creates on behalf of client applications. Connections are configured to define the parties involved in sending or receiving messages, to specify how the client runtime handles message flow, and to automatically set certain information for all messages sent across a connection.

There are two types of connection factory objects:

- `ConnectionFactory` supports normal messaging and nondistributed transactions.
- `XAConnectionFactory` supports distributed transactions.

The `ConnectionFactory` and `XAConnectionFactory` objects share the same set of attributes.

A connection factory object can be created and configured by an administrator or by an application (for prototyping or testing). You set connection factory attributes using the `imqobjmgr` tool or the administration console.

This section describes the connection factory attributes in the following sections, which are organized by the behaviors that the attributes affect:

- "Connection Handling" on page 178
- "Client Identification" on page 180
- "Reliability And Flow Control" on page 182
- "Queue Browser Behavior and Server Session" on page 183
- "Message Header Overrides" on page 184
- "JMS-Defined Properties Support" on page 183

The attribute you are primarily concerned with is `imqAddressList`, which you use to specify the broker to which the client will establish a connection. "Adding a Connection Factory" on page 189 explains how to specify attributes when you add a connection factory administered object to your object store.

For reference information about connection factory attributes, see Chapter 16, "Administered Object Attribute Reference," and the JavaDoc API documentation for the Message Queue class `com.sun.messaging.ConnectionConfiguration`.

Connection Handling

You use connection handling attributes to specify the message server address to which you want to connect and, if reconnection is required, to specify how many times reconnection should be attempted and the interval between attempts.

A client connects to a message server at the message server address that you specify as the value for the `imqAddressList` attribute. The message server address contains a broker host name, a port number, and a connection service type.

The port number can be the port where the broker's Port Mapper resides, or the port where a specific connection service resides. If you specify the Port Mapper port, the Port Mapper dynamically assigns the port number for the connection. For complete information about specifying a message server address, see "Syntax for the imqAddressList Attribute Value" on page 335.

Automatic Reconnection

In a single broker environment or multi-broker cluster environment, you can set connection handling attributes that enable a client to automatically reconnect to a broker if a connection fails. You can also configure the reconnection process.

The reconnection feature 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.

If auto-reconnect is enabled, Message Queue persists temporary destinations when a connection fails, because clients might reconnect and access them again. After giving clients time to reconnect and use these destinations, the broker deletes the destinations.

The way that reconnection is handled depends on whether the client is connected to a single broker or to a broker that is part of a cluster. The following sections describe each of these possibilities.

Reconnecting to a Single Broker To enable a client to be automatically reconnected to a broker when a connection fails, you set the following connection factory attributes:

- `imqReconnectEnabled`. Enables the automatic reconnect behavior.
- `imqReconnectAttempts`. Specifies how many times the client runtime attempts to reconnect the client.
- `imqReconnectInterval`. Specifies how long the client runtime waits between attempts to reconnect the client.

For full reference information about these attributes, see "Connection Handling" on page 334

Reconnecting to a Broker in a Cluster In a multi-broker cluster environment, automatic reconnection iterates through a list of brokers if you specify multiple addresses for the `imqAddressList` attribute. All brokers in the list must be installations of Message Queue Enterprise Edition.

If the client connection to the first address in the list fails, the client runtime attempts to reconnect the client to another broker in the list. If that attempt fails, the client runtime continues through the list until it is able to reconnect the client.

If no attempt is successful, the client runtime cycles through the list for a specified number of times until it finds an available broker or fails to find one. The setting of the `imqAddressListBehavior` attribute determines whether the broker chosen for reconnection is next in the sequence of addresses provided in the address list, or whether it is randomly chosen from that list.

To enable a client to be reconnected to a broker in a cluster, use the following attributes:

- `imqReconnectEnabled`. Enables the automatic reconnect behavior.
- `imqReconnectAttempts`. Specifies how many times to try each broker address before passing to the next.
- `imqReconnectInterval`. Specifies how long to wait between attempts.
- `imqAddressListIterations` . Specifies the number of times to iterate through the list.
- `imqAddressListBehavior`. Specifies whether connection attempts are in the order of addresses in the address list or in a random order.

For full reference information about these attributes, see "Connection Handling" on page 334.

Connection Ping

The `imqPingInterval` attribute specifies the frequency of a ping operation from the client runtime to the broker. By periodically testing the connection, the client runtime can preemptively detect a failed connection. If the ping operation fails, the client runtime throws an exception to the client application's exception listener object. If the application does not have an exception listener, the application's next attempt to use the connection fails.

Use of the ping is especially important for consumer client applications that wait to receive messages and do not send messages. Such an application would not otherwise know when a connection fails. A client that produces infrequent messages can also benefit from this feature, because it could handle a failed connection before needing to send a message.

By default, the ping interval is set at 30 seconds. A value of -1 disables the ping operation.

The response to a broken connection is operating system-specific. For example, on some operating systems, a ping reports a failure immediately. Other operating systems might continue trying to establish the connection to the broker, buffering successive pings until the ping is successful or the buffer overflows.

For full reference information about the `imqPingInterval` attribute, see "Connection Handling" on page 334.

Client Identification

Message Queue defines a set of connection factory attributes to support client authentication and the setting of a unique client ID, which is required for durable subscribers.

Clients attempting to connect to the broker must be authenticated. If the client does not specify a user name or password when creating the connection, one of the following happens:

- If the connection factory attributes `imqDefaultUsername` and `imqDefaultPassword` are not set, the client runtime passes the values `guest/guest` to the broker, and the broker authenticates the client using those values.

  The user repository is shipped with the entry `guest/guest`, so the client will obtain the connection.

- If the connection factory attributes `imqDefaultUsername` and `imqDefaultPassword` attributes are set, the client runtime passes those values to the broker, and the broker authenticates the client using those values.

  If that user/password pair is in the user repository, the client gets the connection.

This scheme allows any user to get a connection, which is convenient for development and testing. In a production system, access to connections should be limited to users that have been added to the user repository.

In addition to broker authentication of clients that request a connection, the JMS specification requires that a connection provides a unique client identifier when where state has to be maintained for the client. Message Queue uses the client ID to keep track of its durable subscribers. If a durable subscriber becomes inactive, the broker retains messages for that subscriber and delivers them when the subscriber becomes active again. The broker identifies the subscriber by means of its client ID.

You can set ClientID administratively, or clients can set it programmatically. If multiple clients obtain connections from the same connection factory object, set ClientID for a connection factory. Message Queue can then provide a unique ClientID for each connection obtained from that factory.

To ensure a unique ClientID value, set the `imqConfiguredClientID` attribute using the following format:

`imqConfiguredClientID=${u}`*string*

The `${u}` must be the first four characters of the attribute value. If anything other than "u" is encountered, a JMS exception occurs upon connection creation.

The value for *string* is any value that you want to associate with a connection produced by this connection factory, such as `Xconn`. During the user authentication stage, Message Queue substitutes `u:`*userName* for *u*. For example, if the user associated with the connection is `Athena` and the string specified for the connection is `${u}Xconn`, the ClientID will be `u:AthenaXconn`.

This scheme ensures that each connection produced by a connection factory, although identical in every other way, will contain a unique ClientID.

There is one case in which this scheme will not work: If two clients obtain a connection using a default user name such as `guest`, each will have a ClientID with the same `${u}` component. At runtime, the first client to request the connection will get it; the second will not because MQ cannot create a connection with a non-unique ClientID.

You can set the `imqDisableSetClientID` attribute to disallow clients that use the connection factory from programmatically changing the configured client ID.

You must set the `imqConfiguredClientID` attribute for durable subscriptions, unless the application code uses the `setClientId()` method.

In summary, these are the attributes that affect client identification:

- `imqDefaultUsername`. Specifies the default user name that will be used to authenticate with the broker when the client does not specify a user name in creating the connection.
- `imqDefaultPassword`. Specifies the default password that will be used to authenticate with the broker when the client does not specify a password in creating the connection..
- `imqConfiguredClientID`. Specifies the value of an administratively configured client ID.
- `imqDisableSetClientID`. Specifies whether a client who uses the connection factory can change the client ID programmatically.

For full reference information about these attributes, see "Client Identification" on page 338.

Reliability And Flow Control

Messages sent and received by clients and control messages used by Message Queue pass over the same client-broker connection. As a result, delays can occur in the delivery of control messages, such as broker acknowledgments, if they are held up by the delivery of JMS messages.

You can set connection factory attributes that allow you to manage the flow of control messages relative to the flow of client messages. Controlling the flow of the two types of messages involves a compromise between reliability and throughput. For a discussion of how you use these attributes to manage flow control and reliability, see "Client Runtime Message Flow Adjustments" on page 244

The following attributes affect the flow of client and control messages:

- `imqAckTimeout`. Specifies, in milliseconds, the maximum time that the client runtime will wait for any broker response.
- `imqConnectionFlowCount`. Specifies the number of JMS messages in a metered batch.
- `imqConnectionFlowLimitEnabled`. Limits message flow at the connection level.
- `imqConnectionFlowLimit`. Specifies a limit on the number of messages that can be delivered over a connection and buffered in the client runtime, waiting to be consumed.
- `imqConsumerFlowLimit`. Specifies a per-consumer limit on the number of messages that can be delivered over a connection and buffered in the client runtime, waiting to be consumed.
- `imqConsumerFlowThreshold`. Specifies, as a percentage of `imqConsumerFlowLimit`, the number of messages for each consumer to buffer in the client runtime, below which delivery of messages for a consumer will resume.

For full reference information about these attributes, see "Reliability and Flow Control" on page 339.

Queue Browser Behavior and Server Session

These attributes affect client queue browsing:

- `imqQueueBrowserMaxMessagesPerRetrieve`. Specifies the maximum number of messages that a client retrieves at one time, when browsing the contents of a queue destination.
- `imqQueueBrowserRetrieveTimeout`. Specifies how long the client waits to retrieve messages, when browsing the contents of a queue destination.
- `imqLoadMaxToServerSession`.For JMS application server facilities, specifies whether a Message Queue ConnectionConsumer loads up to the `maxMessages` number of messages into a ServerSession's session, or loads a single message at a time.

For full reference information about these attributes, see "Queue Browser Behavior and Server Session" on page 340.

JMS-Defined Properties Support

You can use connection factory attributes to automatically set JMS-defined properties on messages that a connection produces. The JMS properties are defined in the JMS specification, at `http://java.sun.com/products/jms/docs.html`.

Use the following attributes to set JMS-defined properties:

- `imqSetJMSXUserID`. For produced messages, specifies whether Message Queue sets the JMS-defined property `JMSXUserID` (identity of user sending the message).
- `imqSetJMSXAppID`. For produced messages, specifies whether Message Queue sets the JMS-defined property `JMSXAppID` (identity of application sending the message).
- `imqSetJMSXProducerTXID`. For produced messages, specifies whether Message Queue sets the JMS-defined property `JMSXProducerTXID` (transaction identifier of the transaction that produced the message).
- `imqSetJMSXConsumerTXID`. For consumed messages, specifies whether Message Queue should set the JMS-defined property `JMSXConsumerTXID` (transaction identifier of the transaction that consumed the message).
- `imqSetJMSXRcvTimestamp`. For consumed messages, specifies whether Message Queue should set the JMS-defined property, `JMSXRcvTimestamp` (the time the message is delivered to the consumer).

For full reference information about these attributes, see "JMS-Defined Properties Support" on page 341.

Message Header Overrides

You can override JMS message header fields that specify the persistence, lifetime, and priority of messages by setting attributes of a connection factory. The settings are used for all messages produced by connections obtained from the connection factory.

The values in the following JMS fields can be overridden:

- JMSDeliveryMode (message persistence/non-persistence)
- JMSExpiration (message lifetime)
- JMSPriority (message priority—an integer from 0 to 9)

For more information about these fields, see the JMS specification at `http://java.sun.com/products/jms/docs.html`.

Because overriding message headers could interfere with application requirements, use this feature only in consultation with application users or designers.

The following list contains the connection factory attributes that deal with message overrides. Most of these attributes are paired. For each pair, the first attribute specifies whether a specified header field can be overridden, and the second attribute specifies the override value.

- `imqOverrideJMSDeliveryMode` and `imqJMSDeliveryMode`.. The first attribute specifies whether a client-set `JMSDeliveryMode` field can be overridden; the second attribute specifies its override value.
- `imqOverrideJMSExpiration` and `imqJMSExpiration`. The first attribute specifies whether a client-set `JMSExpiration` field can be overridden; the second attribute specifies its override value.
- imqOverrideJMSPriority and imqJMSPriority. The first attribute specifies whether a client-set `JMSPriority` field can be overridden; the second specifies attribute its override value.
- imqOverrideJMSHeadersToTemporaryDestinations. Specifies whether overrides apply to temporary destinations.

For full reference information about these attributes, see "Message Header Overrides" on page 338.

Destination Administered Object Attributes

The destination administered object that identifies a physical topic or queue destination has the attributes listed in Table 16-1 on page 333. The section, "Adding a Topic or Queue" on page 190, explains how you specify these attributes when you add a destination administered object to your object store.

The attribute you are primarily concerned with is `imqDestinationName`. This is the name you assign to the physical destination that corresponds to the topic or queue administered object. You can also provide a description of the destination that will help you distinguish it from others that you might create to support many applications.

For more information, see the JavaDoc API documentation for the Message Queue class `com.sun.messaging.DestinationConfiguration`.

Using the Object Manager Utility (imqobjmgr)

The Object Manager utility allows you to create and manage Message Queue administered objects. Using this utility, you can perform the following tasks:

- Add or delete administered objects to an object store.
- List existing administered objects.
- Query and display information about an administered object.
- Modify an existing administered object in the object store.

For reference information about the syntax, subcommands, and options of the `imqobjmgr` command, see Chapter 13, "Command Reference." The following section describes information that you need to provide when working with any `imqobjmgr` subcommand.

Required Information

When performing most tasks related to administered objects, you must specify the following information as options to `imqobjmgr` subcommands:

- **The administered object type**

  The allowed types are shown in Table 13-11 on page 297.

- **The JNDI lookup name** of the administered object:

  This is the logical name that will be used in the client code to refer to the administered object (using JNDI) in the object store.

- **Administered object attributes** (needed especially for the `add` and `update` subcommands):

  - For destinations: The name of the physical destination on the broker. This is the name that was specified with the `-n` option to the `imqcmd create dst` subcommand. If you do not specify the name, the default name of `Untitled_Destination_Object` will be used.

  - For connection factories: The most commonly used attribute is the address list (`imqAddressList`) specifying the message server addresses (one or more) to which the client will attempt to connect. If you do not specify this information, the local host and default port number (`7676`) are used, meaning the client will attempt a connection to a broker on port `7676` of the local host. The section "Adding a Connection Factory" on page 189 explains how you specify object attributes.

    For additional attributes, see "Connection Factory Attributes." on page 177.

- **Object store attributes**

  This information depends on whether you are using a file-system store or LDAP server, but must include the following attributes:

  - The type of JNDI implementation (initial context attribute). For example, file-system or LDAP.

  - The location of the administered object in the object store (provider URL attribute), that is, its "folder" as it were.

  - The user name, password, and authorization type, if any, required to access the object store.

  For more information about object store attributes see "LDAP Server Object Store" on page 174 and "File-System Object Store" on page 175.

Using Command Files

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 `imqobjmgr` subcommand clause.

Using a command file with the Object Manager utility (`imqobjmgr`) is especially useful to specify object store attributes, which are likely to be the same across multiple invocations of `imqobjmgr` and which normally require a lot of typing. Using an command file can also allow you to avoid a situation in which you might otherwise exceed the maximum number of characters allowed for the command line.

The general syntax for an `imqobjmgr` command file is as follows (the version property reflects the version of the command file and not of the Message Queue product—it is not a command line option—and its value must be set to `2.0`):

```
version=2.0
cmdtype=[ add | delete | list | query | update ]
obj.type=[ q | t | qf | tf | cf | xqf | xtf | xcf | e ]
obj.lookupName=lookup name
obj.attrs.objAttrName1=value1
obj.attrs.objAttrName2=value2
obj.attrs.objAttrNameN=valueN
…
objstore.attrs.objStoreAttrName1=value1
objstore.attrs.objStoreAttrName2=value2
objstore.attrs.objStoreAttrNameN=valueN
…
```

As an example of how you can use an command file, consider the following `imqobjmgr` command:

```
imqobjmgr add
       -t qf
       -l "cn=myQCF"
       -o "imqAddressList=mq://foo:777/jms"
       -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=fooUser, ou=People, o=imq"
       -j "java.naming.security.credentials=fooPasswd"
       -j "java.naming.security.authentication=simple"
```

This command can be encapsulated in a file, say `MyCmdFile`, that has the following contents:

```
version=2.0
cmdtype=add
obj.type=qf
obj.lookupName=cn=myQCF
obj.attrs.imqAddressList=mq://foo:777/jms
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=fooUser, ou=People, o=imq
  objstore.attrs.java.naming.security.credentials=fooPasswd
  objstore.attrs.java.naming.security.authentication=simple
```

You can then use the `-i` option to pass this file to the Object Manager utility (`imqobjmgr`):

```
imqobjmgr -i MyCmdFile
```

You can also use the command file to specify some options, while using the command line to specify others. This allows you to use the command file to specify parts of the subcommand clause that is the same across many invocations of the utility. For example, the following command specifies all the options needed to add a connection factory administered object, except for those that specify where the administered object is to be stored.

```
imqobjmgr add
       -t qf
       -l "cn=myQCF"
       -o "imqAddressList=mq://foo:777/jms"
       -i MyCmdFile
```

In this case, the file `MyCmdFile` would contain the following definitions:

```
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=fooUser, ou=People, o=imq
      objstore.attrs.java.naming.security.credentials=fooPasswd
      objstore.attrs.java.naming.security.authentication=simple
```

Additional examples of command files can be found at the following location:

`/usr/demo/imq/imqobjmgr` (Solaris)
`/opt/sun/mq/examples/imqobjmgr` (Linux)
`IMQ_HOME/demo/imqobjmgr` (Windows)

Adding and Deleting Administered Objects

This section explains how you add administered objects for connection factories and topic or queue destinations to the object store.

NOTE The Object Manager utility (`imqobjmgr`) 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 obtain a connection to the broker, you add an administered object that represents the type of connections the client applications want: a topic connection factory or a queue connection factory

To add a queue connection factory, use a command like the following:

```
imqobjmgr add
    -t qf
    -l "cn=myQCF"
    -o "imqAddressList=mq://myHost:7272/jms"
    -j "java.naming.factoryinitial=
                com.sun.jndi.ldap.LdapCtxFactory"
    -j "java.naming.provider.url=ldap://mydomain.com:389/o=imq"
    -j "java.naming.security.principal=
```

```
            uid=fooUser, ou=People, o=imq"
    -j "java.naming.security.credentials=fooPasswd"
    -j "java.naming.security.authentication=simple"
```

The preceding command creates an administered object whose lookup name is `cn=myQCF` and which connects to a broker running on `myHost` and listens on port `7272`. The administered object is stored in an LDAP server. You can accomplish the same thing by specifying an command file as an argument to the `imqobjmgr` command. For more information, see "Using Command Files" on page 186.

NOTE **Naming Conventions:** If you are using an LDAP server to store the administered object, it is important that you assign a lookup name that has the prefix "cn=" as in the example above (`cn=myQCF`). You specify the lookup name with the `-l` option. You do not have to use the `cn` prefix if you are using a file-system object store, but do not use lookup names that have a "/" in them. See Table 8-3.

Table 8-3 Naming Convention Examples

Object Store Type	Good Name	Ban Name
LDAP server	cn=myQCF	myQCF
file system	myTopic	myObjects/myTopic

Adding a Topic or Queue

To enable client applications to access physical destinations on the broker, you add administered objects that identify these destinations, to the object store.

It is a good practice to first create the physical destinations before adding the corresponding administered objects to the object store. Use the Command utility (`imqcmd`) to create the physical destinations on the broker that are identified by destination administered objects in the object store. For information about creating physical destinations, see "Getting Information About Connections" on page 121.

The following command adds an administered object that identifies a topic destination whose lookup name is `myTopic` and whose physical destination name is `TestTopic`. The administered object is stored in an LDAP server.

```
imqobjmgr add
-t t
-l "cn=myTopic"
-o "imqDestinationName=TestTopic"
-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=fooUser, ou=People, o=imq"
-j "java.naming.security.credentials=fooPasswd"
-j "java.naming.security.authentication=simple"
```

This is the same command, only the administered object is stored in a Solaris file system:

```
imqobjmgr add
      -t t
      -l "cn=myTopic"
      -o "imqDestinationName=TestTopic"
      -j "java.naming.factory.initial=
                com.sun.jndi.fscontext.RefFSContextFactory"
      -j "java.naming.provider.url=
                file:///home/foo/imq_admin_objects"
```

In the LDAP server case, as an example, you could use an command file, `MyCmdFile`, to specify the subcommand clause. The file would contain the following text:

```
version=2.0
cmdtype=add
obj.type=t
obj.lookupName=cn=myTopic
obj.attrs.imqDestinationName=TestTopic
objstore.attrs.java.naming.factory.initial=
            com.sun.jndi.fscontext.RefFSContextFactory
objstore.attrs.java.naming.provider.url=
            file:///home/foo/imq_admin_objects
  objstore.attrs.java.naming.security.principal=
            uid=fooUser, ou=People, o=imq
  objstore.attrs.java.naming.security.credentials=fooPasswd
  objstore.attrs.java.naming.security.authentication=simple
```

Use the -i option to pass the file to the `imqobjmgr` command:

```
imqobjmgr -i MyCmdFile
```

NOTE If you are using an LDAP server to store the administered object, it is important that you assign a lookup name that has the prefix "cn=" as in the example above. You specify the lookup name with the `-l` option. You do not have to use this prefix if you are using a file-system object store.

Adding a queue object is exactly the same, except that you specify `q` for the `-t` option.

Deleting Administered Objects

Use the `delete` subcommand to delete an administered object. You must specify the lookup name of the object, its type, and its location.

The following command deletes an administered object for a topic whose lookup name is `cn=myTopic` and which is stored on an LDAP server.

```
imqobjmgr delete
     -t t
     -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=fooUser, ou=People, o=imq"
      -j "java.naming.security.credentials=fooPasswd"
      -j "java.naming.security.authentication=simple"
```

Listing Administered Objects

Use the `list` subcommand to get a list of all administered objects or to get a list of all administered objects of a specific type. The following sample code assumes that the administered objects are stored in an LDAP server.

The following command lists all 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=fooUser, ou=People, o=imq"
    -j "java.naming.security.credentials=fooPasswd"
    -j "java.naming.security.authentication=simple"
```

The following command lists all objects of type `queue`.

```
imqobjmgr list
    -t q
    -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=fooUser, ou=People, o=imq"
    -j "java.naming.security.credentials=fooPasswd"
    -j "java.naming.security.authentication=simple"
```

Getting Information About a Single Object

Use the `query` subcommand to get information about an administered object. You must specify the object's lookup name and the attributes of the object store containing the administered object (such as initial context and location).

In the following example, the `query` subcommand is used to display information about an object whose lookup name is cn=`myTopic`.

```
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=fooUser, ou=People, o=imq"
    -j "java.naming.security.credentials=fooPasswd"
  -j "java.naming.security.authentication=simple"
```

Updating Administered Objects

You use the `update` command to modify the attributes of administered objects. You must specify the lookup name and location of the object. You use the -o option to modify attribute values.

This command changes the attributes of an administered object that represents a topic connection factory:

```
imqobjmgr update
 -t tf
 -l "cn=MyTCF"
 -o imqReconnectAttempts=3
 -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=fooUser, ou=People, o=imq"
 -j "java.naming.security.credentials=fooPasswd"
 -j "java.naming.security.authentication=simple"
```

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

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 14-11 on page 327. 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's 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.

Instead of using a cluster configuration file, you can 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 159. 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 202 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:

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	No single command gets all data
	Convenient for spot checking	Difficult to analyze data programmatically
	Reporting interval set in command option; can be changed on the fly	Doesn't create historical record
	Easy to select specific data of interest	Difficult to see historical trends
	Data presented in easy tabular format	
Log files	Regular sampling	Need to configure broker properties; must shut down and restart broker to take effect
	Creates a historical record	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	Need to configure broker properties; must shut down and restart broker to take effect
	Easy to select specific data of interest	You need to write your own metrics monitoring client
	Data can be analyzed programmatically and presented in any format	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 349.

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, "Operating System-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 timestamp, 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 timestamp time zone, see information about the imq.log.timezone property, which is described in Table 14-10 on page 324.

Changing the Logger Configuration

Log-related properties are described in Table 14-10 on page 324.

➤ **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 205.

➤ **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.
- **Broker-wide 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 214 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:port*] [-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:port*] [-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:port*] [-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:port*	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` *destTyp*	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 Interactively" on page 67.

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 broker-wide, connection service, and physical destination metrics.

Broker-wide 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 350):

```
-------------------------------------------------------
 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 352):

```
-------------------------------------------------
  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 354):

```
--------------------------------------------------------------------------------
  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 354):

```
-----------------------------------------------------------------
  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:port*]	Information on the current number of messages and message bytes stored in broker memory and persistent store (see "Displaying Broker Information" on page 111)
or	
`query svc -n` *serviceName* [`-b` *hostName:port*]	Information on the current number of allocated threads and number of connections for a specified connection service (see "Displaying Connection Service Information" on page 118)
or	
`query dst -t` *destType* `-n` *destName* [`-b` *hostName:port*]	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 131)

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 349.

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 152.

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 159.

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 349.

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 non-persistent 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 224), 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, built-in versus plugged-in 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:

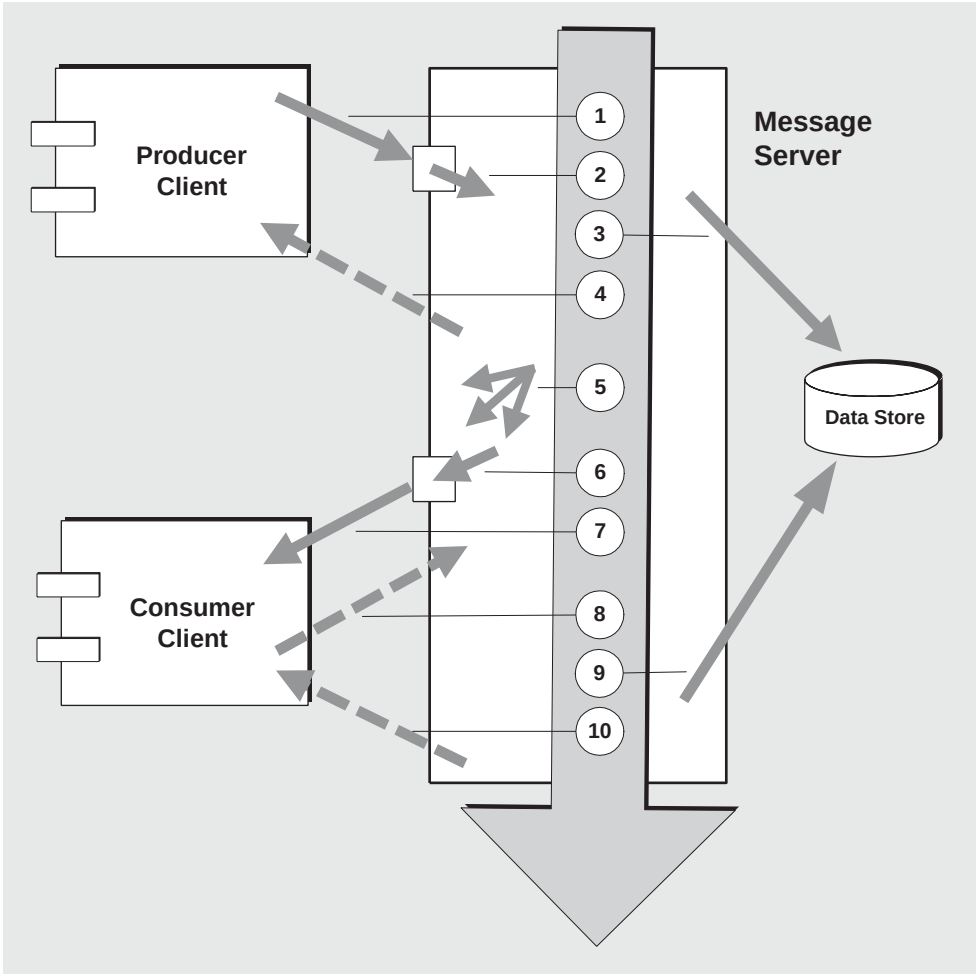
- If one physical destination is used significantly more than others, you might want to set higher message memory limits on that physical destination than on others, or to adjust limit behaviors accordingly.
- If the number of connections needed is significantly greater than allowed by the maximum thread 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/Non-persistent 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	Non-persistent 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/Non-persistent Messages)

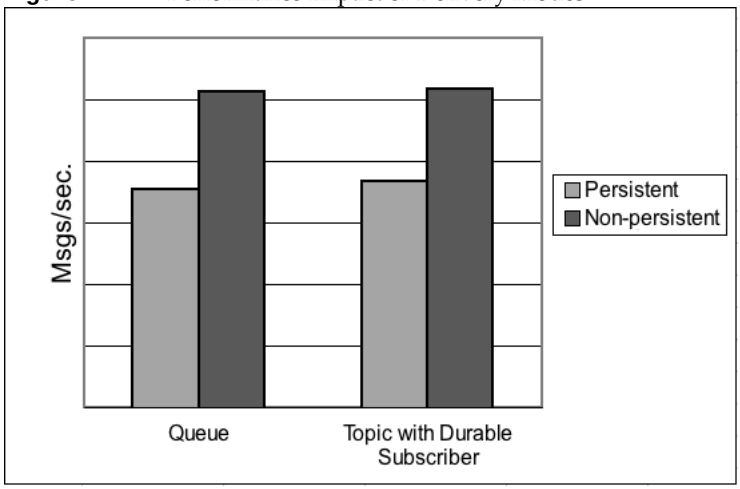
Persistent messages guarantee message delivery in case of message server failure. The broker stores the message in a persistent store until all intended consumers acknowledge they have consumed the message.

Broker processing of persistent messages is slower than for non-persistent 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 non-persistent modes can be significant. Figure 11-2 compares throughput for persistent and non-persistent 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.

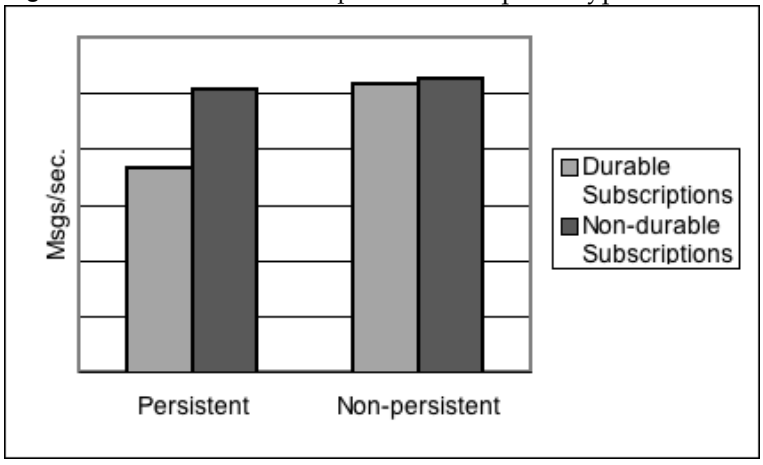
Durable subscriptions provide increased reliability but slower throughput, for the following reasons:

- 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 non-persistent 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

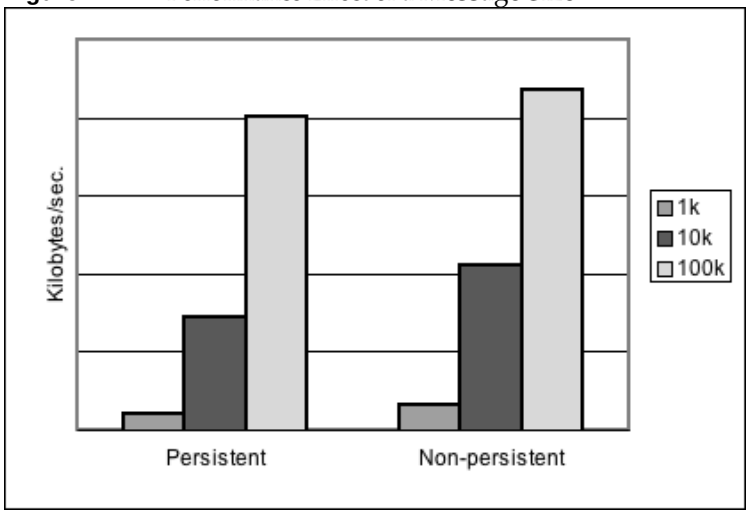
Message size affects performance because more data must be passed from producing client to broker and from broker to consuming client, and because for persistent messages a larger message must be stored.

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 non-persistent 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 non-persistent 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 uses a Thread Pool Manager, which you can configure to support either a dedicated thread model or a shared thread model (see "Thread Pool Manager" 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 75.

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

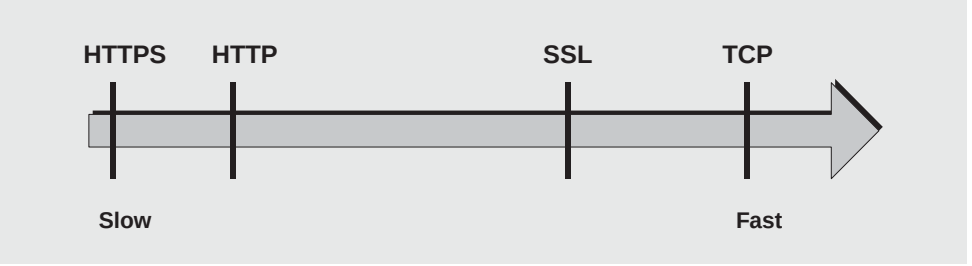


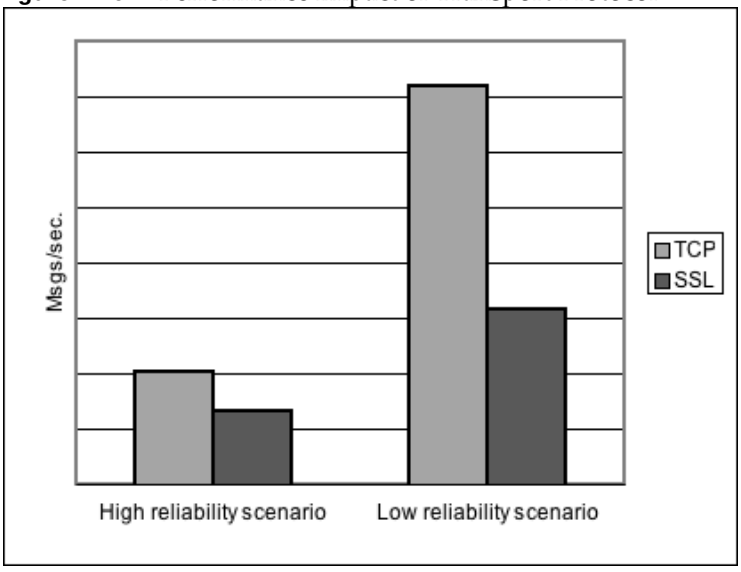
Figure 11-5 reflects the performance characteristics of the various protocol technologies:

- 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 non-persistent 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 non-persistent 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 built-in and plugged-in persistence. Built-in persistence is a file-based data store. Plugged-in persistence uses a Java Database Connectivity (JDBC™) interface and requires a JDBC-compliant data store.

The built-in persistence is significantly faster than plugged-in persistence; however, a JDBC-compliant database system might provide the redundancy, security, and administrative features needed for an application.

In the case of built-in 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 set properties and behaviors that can generally improve performance and message throughput.

For example, the Message Queue client runtime supports the following configurable behaviors:

- Connection flow metering (`imqConnectionFlowCount`), which helps you prevent congestion due to the flow of both JMS messages and Message Queue control messages across the same connection.
- Connection flow limits (`imqConnectionFlowLimit`), which helps you avoid client resource limitations by limiting the number of messages that can be delivered over a connection to the client runtime, waiting to be consumed.
- Consumer flow limits (`imqConsumerFlowLimit`), which helps improve load balancing among consumers in multi-consumer queue delivery situations (so no one consumer can be sent a disproportionate number of messages) and which helps prevent any one consumer on a connection from overwhelming other consumers on the connection. This property limits the number of messages per consumer that can be delivered over a connection to the client runtime, waiting to be consumed. This property can also be configured as a queue destination property (`consumerFlowLimit`).

For more information on these behaviors and the attributes used to configure them, see "Client Runtime Message Flow Adjustments" on page 244.

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 "imqsvcadmin" in Chapter 13, "Command 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.`*protocol_type*`.nodelay`
- `imq.protocol.`*protocol_type*`.inbufsz`
- `imq.protocol.`*protocol_type*`.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 241).

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 non-persistent 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

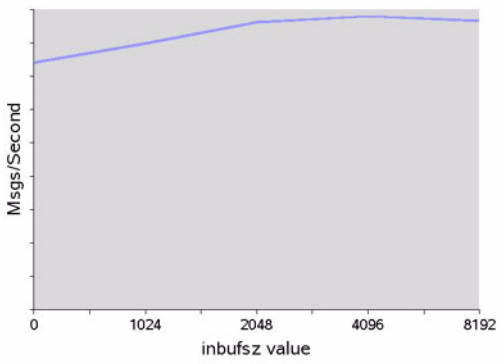
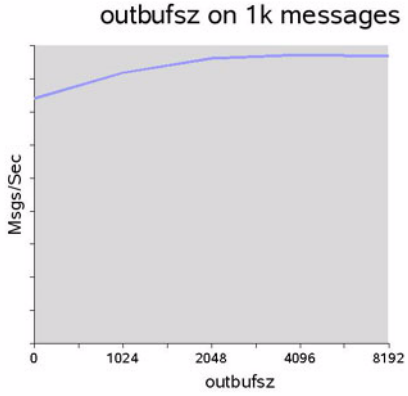


Figure 11-8 shows the consequence of changing `outbufsz` on a 1k packet.

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 Manager" on page 83.

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 system-wide level (for all destinations, collectively).

Using Physical Destination Limits

For information on physical destination limits, see Chapter 6, "Managing Physical Destinations."

Using System-wide 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 non-persistent, the broker silently drops the message.

To have non-persistent messages return an exception like persistent messages, set the following property on the connection factory object used by the client:

```
imqAckOnProduce = true
```

The setting above may decrease the performance of sending non-persistent messages to the broker (the client waits for a reply before sending the next message), but often this is acceptable since message inflow to the broker is typically not a system bottleneck.

When an exception is returned in sending a message, the client should pause for a moment and retry the send again.

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 237). 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 (JMS 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 JMS messages. To prevent this type of congestion, Message Queue meters the flow of JMS messages across a connection.

JMS messages are batched (as specified with the `imqConnectionFlowCount` property) so that only a set number are delivered. When the batch has been delivered, delivery of JMS messages is suspended, and only pending control messages are delivered. This cycle repeats, as other batches of JMS 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, the client is using the `CLIENT_ACKNOWLEDGE or AUTO_ACKNOWLEDGE` modes, persistent messages, transactions, queue browsers, or if the client 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 JMS 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-based Limits

When the number of JMS 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 might be able to finish processing the remaining buffered messages before the broker delivers the next set, causing message throughput degradation. Generally speaking, unless you have specific performance or reliability concerns, you will not have 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-based Limits

In the case of some client applications, however, the number of consumers might 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 `imqConnectionFlowLimit`, delivery of messages through the connection stops until that total drops below the connection limit. (The `imqConnectionFlowLimit` is only enabled if you set the `imqConnectionFlowLimitEnabled` property 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 159.

- If the status of a connection service is shown as `paused`, resume the service (see "Pausing and Resuming a Connection Service" on page 120).

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 Manager" on page 77).

To resolve the problem

- If you are using a dedicated thread pool model (`imq.`*service_name*`.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.`*service_name*`.max_threads`) or switch to the shared thread pool model.
- If you are using a shared thread pool model (`imq.`*service_name*`.threadpool_model=shared`), the maximum number of connections is half the product of the following two properties: the connection Monitor limit (`imq.`*service_name*`.connectionMonitor_limit`) and the maximum number of threads (`imq.`*service_name*`.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 Limits (Solaris or Linux)" 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 147).
- 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 156).

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 248, 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 234).

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 238).

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 238).

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 223 for other possible bottlenecks and check for symptoms associated with the following problems:

- "Message Production Is Delayed or Slowed" on page 256
- "Messages Are Backlogged" on page 259
- "Message Server Throughput Is Sporadic" on page 264

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 131):

```
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 133).

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 157).

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 broker-wide 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 broker-wide 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 broker-wide) being careful not to exceed memory resources.

  In general, you should manage memory on a destination-by-destination level so that broker-wide message limits are never reached. For more information, see "Broker Adjustments" on page 242.

- 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 329).

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 broker-wide 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 built-in persistence, try increasing the disk space of the file-based data store.
- In the case of a JDBC-compliant data store, check that plugged-in 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 "Connection Factory Attributes." on page 177).

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 238).

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 broker-wide 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 122).
- Specify message limit and limit behavior attributes for the topic (see Table 15-1 on page 329). 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 134).
- 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 338).

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 243).
- Specify message limit and limit behavior attributes for the queue (see Table 15-1 on page 329). 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 134).
- 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 338).

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 243).

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 244).

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 238).
- 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 269. Specifically, use the instructions under "The number of messages, or their sizes, exceed destination limits" on page 270. 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.

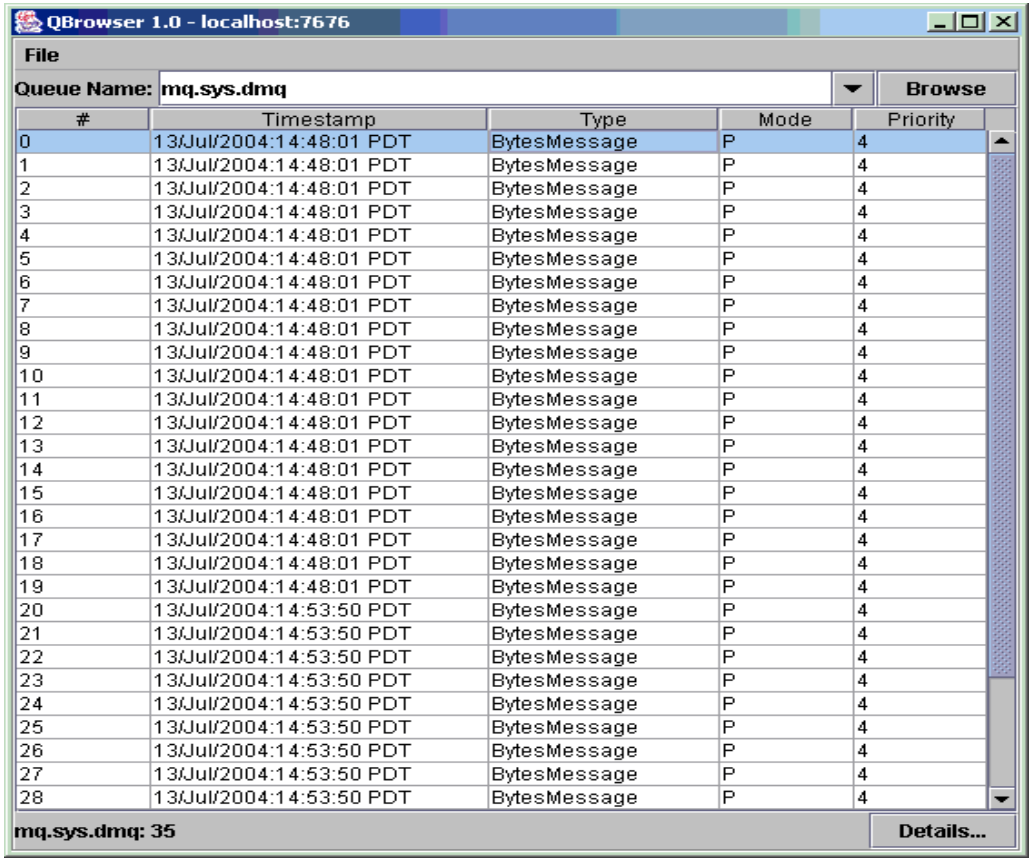
Use the QBrowser demo application to look at the DMQ contents. The QBrowser demo is in an operating system-specific location; for the location, see Appendix A, "Operating System-Specific Locations of Message Queue Data" and look in the tables for "Example Applications and Locations."

This is an example of invocation on Windows:

```
cd \MessageQueue3\demo\applications\qbrowser java QBrowser
```

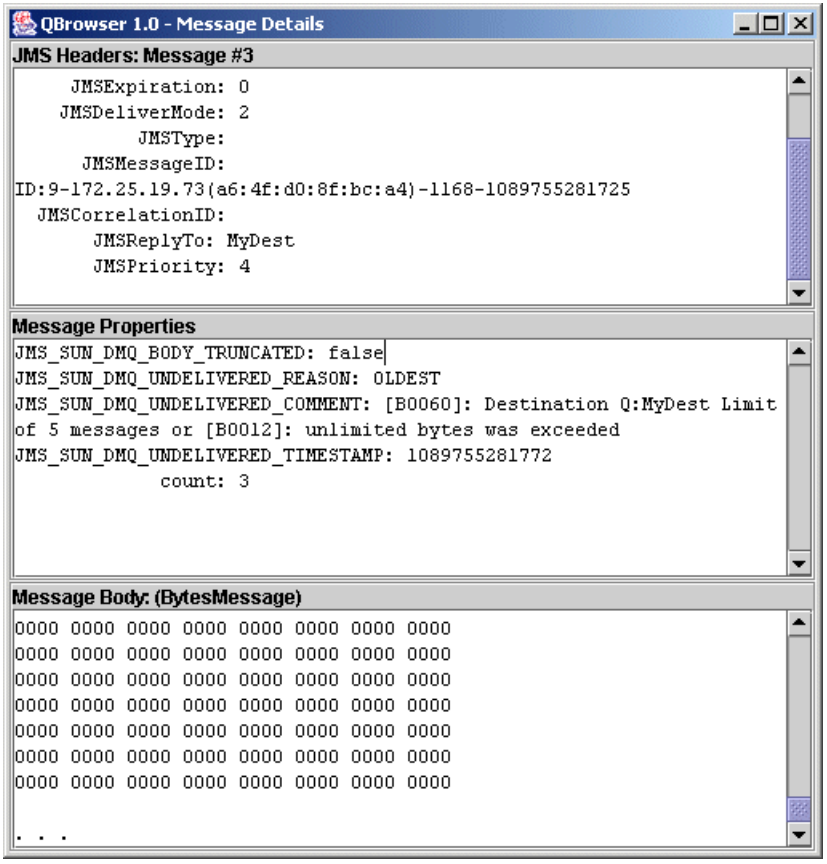
When the QBrowser main window appears, select the queue name `mq.sys.dmq` and then click Browse. A list like the following appears.

Figure 12-1 QBrowser Window



Double click a message to display details about that message.

Figure 12-2 QBrowser Message Details



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 66.

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, "Operating System-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 267 appears.

Double click any message to display details about that message. The window shown in Figure 12-2 on page 268 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 66.

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 272.

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 272.

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 Reference

This chapter contains a section that describes common command line syntax, and then provides reference information for each of the Message Queue commands. The chapter contains 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. For this reason, there are no commands to start or quit the utility, and no need for such commands.

All the command line utilities share the following command syntax:

Utility_Name [*subcommand*] [*argument*] [[-*option_name* [-*option_argument*]]...]

Utility_Name specifies the name of a Message Queue utility, such as `imqcmd`, `imqobjmgr`, `imqusermgr`, and so on.

Rules for Entering Commands

These are some general rules for entering commands:

- Specify options *after* subcommands (and arguments, if the utility accepts both types of operands).
- If the value for an option contains a space, enclose the entire value in quotation marks. It is generally safest to enclose an attribute-value pair in quotes.
- If you specify the -v (version) or the -h/-H (help) options on a command line, nothing else on that command line is executed.
- Separate the subcommand, arguments, options, and option arguments with spaces.

Command Line Examples

The following is an example of a command line that has no subcommand clause. The command starts the default broker.

```
imqbrokerd
```

The following command is more complicated. The command destroys a destination of type `queue` named `myQueue`. Authentication is performed based on the user `admin`; the command will prompt for the user's password. The `-f` option specifies that there will be no confirmation and the `-s` option specifies that the command is executed in silent mode.

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

Common Command Options

Table 13-1 describes options that are common to all Message Queue administration utilities. You must specify these options *after* the subcommand on the command line. The options can be entered in any order.

Table 13-1 Common Message Queue Command Line Options

Option	Description
`-h`	Displays usage help for the specified utility.
`-H`	Displays expanded usage help, including attribute list and examples (supported only for `imqcmd` and `imqobjmgr`).
`-s`	Turns on silent mode: no output is displayed. Specify as `-silent` for `imqbrokerd`.
`-v`	Displays version information.
`-f`	Performs the given action without prompting for user confirmation.
`-pre`	(Used only with `imqobjmgr`) Turns on preview mode, allowing the user to see the effect of the rest of the command line without actually performing the command. This can be useful in checking for the value of default attributes.
`-javahome` *path*	Specifies an alternative Java 2 compatible runtime to use (default is to use the runtime on the system or the runtime bundled with Message Queue).

imqbrokerd

The `imqbrokerd` command starts a broker. Command-line options override values in the broker configuration files, but only for the current broker session.

Syntax

```
imqbrokerd [[ -Dproperty=value]…]
    [ -backup fileName]
    [ -cluster "[broker1] [[,broker2]…]"
    [ -dbuser userName]
    [ -force]
    [ -h|-help]
    [ -javahome path]
    [ -license licenseName]
    [ -loglevel level]
    [ -metrics interval]
    [ -name instanceName]
    [ -passfile fileName
    [ -port number]
    [ -remove instance]
    [ -reset data]
    [ -restore fileName]
    [ -shared]
    [ -silent|-s] [ -tty]
    [ -upgrade-store-nobackup]
    [ -version]
    [ -vmargs arg1 [[arg2]…]
```

Command Options

Table 13-2 describes the options to the `imqbrokerd` command and describes the configuration properties, if any, affected by each option.

Table 13-2 `imqbrokerd` Options

Option	Properties Affected	Description
`-backup` *fileName*	None affected.	Applies only to broker clusters. Backs up a master broker's configuration change record to the specified file. See "Managing the Configuration Change Record" on page 201.

Table 13-2 `imqbrokerd` Options *(Continued)*

Option	Properties Affected	Description
`-cluster"[`*broker1*`][[,`*broker2*`]...]"` where *broker* is either • *host* • *:port* • *host*:*port*	Overrides `imq.cluster.brokerlist` with a list of brokers to which to connect.	Applies only to broker clusters. Connects to all the brokers on the specified hosts and ports. This list is merged with the list in the `imq.cluster.brokerlist` property. If you don't specify a value for *host*, `localhost` is used. If you don't specify a value for *port*, the value `7676` is used. See "Working With Broker Clusters" on page 195 for more information on how to use this option to connect multiple brokers.
`-dbpassword` *password*	Overrides `imq.persist.jdbc.password` with the specified password	Specifies the password for a plugged-in JDBC-compliant data store. This option is being deprecated and will be removed in a future version. Use one of the following alternatives: • Omit the password from the command line so that the command prompts you for the password. • Use the `-passfile` option to specify a file that contains the database password.
`-dbuser` *userName*	Overrides `imq.persist.jdbc.user` with the specified user name	Specifies the user name for a plugged-in JDBC-compliant database. See "Setting Up a Persistent Store" on page 99
`-D`*property*`=`*value*	Sets system properties. Overrides corresponding property value in instance configuration file.	Sets the specified property to the specified value. 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 the `-D` option. If you pass incorrect values, the system will not warn you, and Message Queue will not be able to set them.
`-force`	None affected.	Performs action without user confirmation. This option applies only to the `-remove instance` and the `-upgrade-store-nobackup` options, which normally require confirmation.
`-h|-help`	None affected.	Displays help. Nothing else on the command line is executed.
`-javahome` *path*	None affected.	Specifies the path to an alternative Java 2-compatible JDK. The default is to use the bundled runtime.

Table 13-2 `imqbrokerd` Options *(Continued)*

Option	Properties Affected	Description
`-ldappassword` *password*	Overrides `imq.user_repository.ldap.password` with the specified password	Specifies the password for accessing a LDAP user repository. This option is being deprecated and will be removed in a future version. Use one of the following alternatives: • Omit the password from the command line so that the command prompts you for the password. • Use the `-passfile` option to specify a file that contains the LDAP password.
`-license` [*licenseName*]	None affected.	Specifies the license to load, if different from the default for your Message Queue product edition. If you don't specify a license name, this lists all licenses installed on the system. Depending on the installed Message Queue edition, the values for *licenseName* are `pe` (Platform Edition—basic features), `try` (Platform Edition—90-day trial enterprise features), and `unl` (Enterprise Edition).
`-loglevel` *level*	Overrides `imq.broker.log.level` with the specified level.	Specifies the logging level as being one of `NONE`, `ERROR`, `WARNING`, or `INFO`. The default value is `INFO`.
`-metrics` *interval*	Overrides `imq.metrics.interval` with the specified number of seconds.	Specifies that broker metrics are written to the logger at an interval specified in seconds.
`-name` *instanceName*	Sets `imq.instancename` to the specified name.	Specifies the instance name of this broker and uses the corresponding instance configuration file. If you do not specify a broker name, the name of the instance is set to `imqbroker`. **Note**: If you run more than one instance of a broker on the same host, each must have a unique name.
`-passfile` *fileName*	Overrides `imq.passfile.enabled` and sets it to `true`. Overrides `imq.passfile.dirpath` with the path containing the file. Overrides `imq.passfile.name` with the name of the file.	Specifies the name of the file from which to read the password for the `imqcmd` command utility, SSL keystore, LDAP user repository, or JDBC-compliant database, or for any combination of them. For more information, see "Using a Passfile" on page 169.

Table 13-2 `imqbrokerd` Options *(Continued)*

Option	Properties Affected	Description
`-password` *keypassword*	Overrides `imq.keystore.password` with the specified password.	Specifies the password for the SSL certificate keystore.This option is being deprecated and will be removed in a future version. Use one of the following alternatives: • Omit the password from the command line so that the command prompts you for the password. • Use the `-passfile` option to specify a file that contains the SSL certificate keystore password.
`-port` *number*	Overrides `imq.portmapper.port` with the specified number.	Specifies the broker's Port Mapper port number. By default, this is set to 7676. To run two instances of a broker on the same server, each broker's Port Mapper must have a different port number. Message Queue clients connect to the broker instance using this port number.
`-remove instance`	None affected.	Causes the broker instance to be removed: deletes the instance configuration file, log files, persistent store, and other files and directories associated with the instance. Requires user confirmation unless `-force` option is also specified.
`-reset store|messages|durables|props`	None affected.	Resets the data store (or a subset of the data store) or the configuration properties of a broker instance, depending on the argument given. Resetting the data store clears out all persistent data, including persistent messages, durable subscriptions, and transaction information. This allows you to start the broker instance with a clean slate. You can also clear only all persistent messages or only all durable subscriptions. (If you do not want the persistent store to be reset on subsequent restarts, restart the broker instance without using the `-reset` option.) Resetting the broker's properties, replaces the existing instance configuration file (`config.properties`) with an empty file: all properties assume default values.

Table 13-2 `imqbrokerd` Options *(Continued)*

Option	Properties Affected	Description
`-restore` *fileName*	None affected.	Applies only to broker clusters. Replaces the master broker's configuration change record with the specified backup file. This file must have been previously created using the `-backup` option. See "Managing the Configuration Change Record" on page 201.
`-shared`	Overrides `imq.jms.threadpool_model` and sets it to `shared`.	Specifies that the jms connection service be implemented using the shared thread pool model, in which threads are shared among connections to increase the number of connections supported by a broker instance.
`-silent|-s`	Overrides `imq.log.console.output` and sets it to `NONE`.	Turns off logging to the console.
`-tty`	Overrides `imq.log.console.output` and sets it to `ALL`	Specifies that all messages be displayed to the console. By default only `WARNING` and `ERROR` level messages are displayed.
`-upgrade-store-nobackup`	None affected	Specifies that an upgrade to Message Queue 3.5 or Message Queue 3.5 SP*x* from an incompatible version automatically removes the old data store. For additional details, see the *Message Queue Installation Guide*.
`-version`	None affected.	Displays the version number of the installed product.
`-vmargs` *arg1* `[[`*arg2*`]…]`	None affected	Specifies arguments to pass to the Java VM. Separate arguments with spaces. If you want to pass more than one argument or if an argument contains a space, use enclosing quotation marks. For example: `imqbrokerd -tty -vmargs "-Xmx128m -Xincgc"` These arguments can be passed only on the command line. There is no associated configuration property in the `config.props` file.

See Also

For more information about using `imqbrokerd` and for command examples, see "Starting Brokers Interactively" on page 67.

imqcmd

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

Syntax

```
imqcmd subcommand argument [options]
imqcmd -h|H
imqcmd -v
```

Subcommands

You always use a subcommand with `imqcmd`, unless you want to display help or display the product version. Table 13-3 lists the `imqcmd` subcommands and specifies where reference information for that subcommand is located.

Table 13-3 `imqcmd` Subcommands

Subcommand and Argument	Description	Reference
`commit txn`	Commits a transaction.	"Transaction Management Subcommands" on page 293
`destroy dur`	Destroys a durable subscription.	"Durable Subscription Subcommands" on page 293
`list cxn`	Lists connections for a broker.	"Connection Subcommands" on page 293
`list dur`	Lists durable subscriptions to a topic.	"Durable Subscription Subcommands" on page 293
`list svc`	Lists services on a broker.	"Connection Services Management Subcommands" on page 292
`list txn`	Lists transactions on a broker.	"Transaction Management Subcommands" on page 293
`metrics bkr`	Displays broker metrics.	"Broker Management Subcommands" on page 289
`metrics svc`	Displays service metrics.	"Connection Services Management Subcommands" on page 292
`pause bkr`	Pauses all services on a broker.	"Broker Management Subcommands" on page 289

Table 13-3 `imqcmd` Subcommands *(Continued)*

Subcommand and Argument	Description	Reference
`pause svc`	Pauses a single service on a broker.	"Connection Services Management Subcommands" on page 292
`purge dur`	Purges all messages on a durable subscription without destroying the durable subscription.	"Durable Subscription Subcommands" on page 293
`query bkr`	Queries and displays information on a broker.	"Broker Management Subcommands" on page 289
`query cxn`	Queries and displays information on a connection.	"Connection Subcommands" on page 293
`query svc`	Queries and displays information on a service.	"Connection Services Management Subcommands" on page 292
`query txn`	Queries and displays information on a transaction.	"Transaction Management Subcommands" on page 293
`reload cls`	Reloads broker cluster configuration.	"Broker Management Subcommands" on page 289
`restart bkr`	Restarts the current running broker instance.	"Broker Management Subcommands" on page 289
`resume bkr`	Resumes all services on a broker.	"Broker Management Subcommands" on page 289
`resume svc`	Resumes one service.	"Connection Services Management Subcommands" on page 292
`rollback txn`	Rolls back a transaction.	"Transaction Management Subcommands" on page 293
`shutdown bkr`	Shuts down the broker instance.	"Broker Management Subcommands" on page 289
`update bkr`	Updates attributes of a broker.	"Broker Management Subcommands" on page 289
`update svc`	Updates attributes of a service.	"Connection Services Management Subcommands" on page 292

The imqcmd command utility also has subcommands for use with physical destinations on a broker. Destination subcommands are described in Chapter 6, "Managing Physical Destinations."

The following sections list the `imqcmd` subcommands by function.

Broker Management Subcommands

Table 13-4 lists the `imqcmd` subcommands used to manage brokers. If no host name or port is specified, the default (`localhost:7676`) is assumed.

Table 13-4 `imqcmd` Subcommands Used to Manage a Broker

Subcommand Syntax	Description

`metrics bkr [-b` *hostName:port*`]`
`[-m` *metricType*`]`
`[-int` *interval*`]`
`[-msp` *numSamples*`]`
	Displays broker metrics for the default broker or 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. (default metric type)
	`rts` Displays metrics on 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).

`pause bkr [-b` *hostName:port*`]`	Pauses the default broker or a broker at the specified host and port. See "Pausing and Resuming a Broker" on page 113.
`query bkr -b` *hostName:port*	Lists the current settings of properties of the default broker or a broker at the specified host and port. Also shows the list of running brokers (in a multi-broker cluster) that are connected to the specified broker.
`reload cls`	Applies only to broker clusters. Forces all the brokers in a cluster to reload the `imq.cluster.brokerlist` property and update cluster information. See "Adding Brokers to a Cluster" on page 199 for more information.
`restart bkr [-b` *hostName:port*`]`	Shuts down and restart the default broker or a broker at the specified host and port, using the options specified when the broker started.
`resume bkr [-b` *hostName:port*`]`	Resumes the default broker or a broker at the specified host and port.
`shutdown bkr [-b` *hostName:port*`]`	Shuts down the default broker or a broker at the specified host and port.

Table 13-4 `imqcmd` Subcommands Used to Manage a Broker *(Continued)*

Subcommand Syntax	Description

`update bkr [-b` *hostName:port*`]`
`-o` *attribute=value*
`[-o` *attribute=value1*`]...`
	Changes the specified attributes for the default broker or a broker at the specified host and port.

Physical Destination Management Subcommands

Table 13-5 lists the `imqcmd` subcommands used to manage physical destinations. If no host name or port is specified, the default (`localhost:7676`) is assumed.

Table 13-5 `imqcmd` Subcommands Used to Manage Destinations

Subcommand Syntax	Description
`compact dst [-t` *destType* `-n` *destName*`]`	Compacts the built-in file-based data store for the destination of the specified type and name. If no destination type and name are specified, all destinations are compacted. Destinations must be paused before they can be compacted.

`create dst -t` *destType*
`-n` *destName*
`[-o` *attribute=value*`]`
`[-o` *attribute=value*`1]...`
	Creates a destination of the specified type, with the specified name, and the specified attributes. A destination name must contain only alphanumeric characters (no spaces) and can begin with an alphabetic character or the characters "_" and "$". It cannot begin with the character string "`mq.`"
	
	You cannot perform this operation in a cluster whose master broker is temporarily unavailable.

`destroy dst -t` *destType* `-n` *destName*	Destroys the destination of the specified type and name. You cannot destroy a system-created destination, such as a dead message queue. You cannot perform this operation in a cluster whose master broker is temporarily unavailable.
`list dst [-t` *destType*`] [-tmp]`	Lists all destinations of the specified type, with option of listing temporary destinations as well. The type argument can have two values: *destType* = `q` (queue) *destType* = `t` (topic) If the type is not specified, all destinations of all types are listed.

Table 13-5 `imqcmd` Subcommands Used to Manage Destinations *(Continued)*

Subcommand Syntax	Description

`metrics dst -t` *destType*
`-n` *destName*
`[-m` *metricType*`]`
`[-int` *interval*`]`
`[-msp` *numSamples*`]`
	Displays metrics information for the destination of the specified type and name.
	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 destination and residing in memory. (default metric type))
	`rts` Displays metrics on rate of flow of messages and packets into and out of the destination (per second) and other rate information.
	`con` Displays consumer-related metrics.
	`dsk` Displays disk usage metrics.
	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).

`pause dst [-t` *destType*
`-n` *destName*`]`
`[-pst` *pauseType*`]`
	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 destinations are paused. The default is `ALL`.

`purge dst -t` *destType* `-n` *destName*	Purges messages at the destination of the specified type and name.
`query dst -t` *destType* `-n` *destName*	Lists information about the destination of the specified type and name.
`resume dst [-t` *destType* `-n` *destName*`]`	Resumes the delivery of messages for the paused destination of the specified type and name. If no destination type and name are specified, all destinations are resumed.
`update dst -t` *destType* `-n` *destName* `-o` *attribute=value* `[-o` *attribute=value1*`]`...	Updates the value of the specified attributes at the specified destination. The attribute name may be any of the attributes described in Table 15-1, unless the destination is the dead message queue, `mq.sys.dmq`.

Connection Services Management Subcommands

Table 13-6 lists the `imqcmd` subcommands used to manage connection services. If no host name or port is specified, the default (`localhost:7676`) is assumed.

Table 13-6 `imqcmd` Subcommands Used to Manage Connection Services

Subcommand Syntax	Description
`list svc [-b` *hostName:port*`]`	Lists all connection services on the default broker or on a broker at the specified host and port.

`metrics svc -n` *serviceName*
`[-b` *hostName:port*`]`
`[-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.
	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 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).

`pause svc -n` *serviceName* `[-b` *hostName:port*`]`	Pauses the specified service running on the default broker or on a broker at the specified host and port. You cannot pause the admin service.
`query svc -n` *serviceName* `[-b` *hostName:port*`]`	Displays information about the specified service running on the default broker or on a broker at the specified host and port.
`resume svc -n` *serviceName* `[-b` *hostName:port*`]`	Resumes the specified service running on the default broker or on a broker at the specified host and port.
`update svc -n` *serviceName* `[-b` *hostName:port*`]` `-o` *attribute=value* `[-o` *attribute=value1*`]`...	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 Service Properties" on page 311.

Connection Subcommands

Table 13-7 lists the `imqcmd` subcommands that apply to connections. If no host name or port is specified, they are assumed to be `localhost`, `7676`.

Table 13-7 `imqcmd` Subcommands Used to Manage Connection Services

Subcommand Syntax	Description
`list cxn [-svn` *serviceName*`]` `[-b` *hostName:port*`]`	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.
`query cxn -n` *connectionID* `[-b` *hostName:port*`]`	Displays information about the specified connection on the default broker or on a broker at the specified host and port.

Durable Subscription Subcommands

Table 13-8 provides a summary of the `imqcmd` durable subscription subcommands. If no host name or port is specified, the default (`localhost:7676`) is assumed.

Table 13-8 `imqcmd` Subcommands Used to Manage Durable Subscriptions

Subcommand	Description
`list dur -d` *destName*	Lists all durable subscriptions for the specified destination.
`destroy dur -n` *subscrName* `-c` *client_id*	Destroys the specified durable subscription with the specified Client Identifier. You cannot perform this operation in a cluster whose master broker is temporarily unavailable.
`purge dur -n` *subscrName* `-c` *client_id*	Purges all messages for the specified durable subscription with the specified Client Identifier.

Transaction Management Subcommands

Table 13-9 provides a summary of the `imqcmd` transactions subcommands. If no host name or port is specified, the default (`localhost:7676`)is assumed.

Table 13-9 `imqcmd` Subcommands Used to Manage Transactions

Subcommand	Description
`list txn`	Lists all transactions, being tracked by the broker.
`query txn -n` *transaction_id*	Lists information about the specified transaction.
`commit txn -n` *transaction_id*	Commits the specified transaction.
`rollback txn -n` *transaction_id*	Rolls back the specified transaction.

Command Options

Table 13-10 lists the options to the `imqcmd` command.

Table 13-10 `imqcmd` Options

Option	Description
`-b` *hostName:port*	Specifies the name of the broker's host and its port number. The default value is `localhost:7676`. To specify port only: `-b :7878` To specify name only: `-b somehost`
`-c` *clientID*	Specifies the ID of the durable subscriber to a topic. See "Managing Durable Subscriptions" on page 122.
`-d` *destinationName*	Specifies the name of the topic. Used with the `list dur` and `destroy dur` subcommands. See "Managing Durable Subscriptions" on page 122.
`-f`	Performs action without user confirmation.
`-h`	Displays usage help. Nothing else on the command line is executed. A user name and password is not needed with this option.
`-H`	Displays usage help, attribute list, and examples. Nothing else on the command line is executed. A user name and password is not needed with this option.
`-int` *interval*	Specifies the interval, in seconds, at which the `metrics bkr`, `metrics dst`, and `metrics svc` subcommands display metrics output.
`-javahome` *path*	Specifies an alternative Java 2 compatible runtime to use (default is to use the runtime on the system or the runtime bundled with Message Queue).

Table 13-10 `imqcmd` Options *(Continued)*

Option	Description
`-m` *metricType*	Specifies the type of metric information to display. Use this option with the `metrics dst`, `metrics svc`, or `metrics bkr` subcommand. The value of *metricType* depends on whether the metrics are generated for a destination, a service, or a broker.
`-msp` *numSamples*	Specifies the number of metric samples the `metrics bkr`, `metrics dst`, and `metrics svc` subcommands display in their metrics output.
`-n` *argumentName*	Specifies the name of the subcommand argument. Depending on the subcommand, this might be the name of a service, a physical destination, a durable subscription, a connection ID, or a transaction ID.
`-o` *attribute=value*	Specifies the value of an attribute. Depending on the subcommand argument, this might be the attribute of a broker (see "Using the imqcmd Command Utility" on page 108), service (see "Managing Connection Services" on page 116), or destination (see "Managing Durable Subscriptions" on page 122).
`-p` *password*	Specifies your (the administrator's) password. This option is being deprecated and will be unsupported in a future release. Use one of the following alternatives: • Omit the password from the command line so that the command prompts you for the password. • Use the `-passfile` option to specify a file containing the administrator's password.
`-passfile` *path*	Specifies the path to a file containing the password for the user issuing the command. For more information, see "Using a Passfile" on page 169.
`-pst` *pauseType*	Specifies whether producers, consumers, or both are paused when pausing a destination. See "Managing Durable Subscriptions" on page 122.
`-rtm` *timeout*	Specifies the initial (retry) timeout period (in seconds) of an `imqcmd` subcommand. The timeout is the length of time the `imqcmd` subcommand will wait after making a request to the broker. Each subsequent retry of the subcommand will use a timeout value that is a multiple of the initial timeout period. Default: `10`
`-rtr` *numRetries*	Specifies the number of retries attempted after an `imqcmd` subcommand first times out. Default: `5`
`-s`	Silent mode. No output will be displayed.
`-secure`	Specifies a secure administration connection to the broker using the `ssladmin` connection service (see "Step 4. Configuring and Running SSL-Based Clients" on page 164). If you omit this option, the connection will not be secure.

Table 13-10 `imqcmd` Options *(Continued)*

Option	Description
`-svn` *serviceName*	Specifies the service for which connections are listed. See "Getting Information About Connections" on page 121.
`-t` *destType*	Specifies the type of a destination: `t` (topic) or `q` (queue). See "Managing Durable Subscriptions" on page 122.
`-tmp`	Displays temporary destinations. See Table 13-5 on page 290.
`-u` *userName*	Specifies your (the administrator's) name. If you omit this value, you will be prompted for it.
`-v`	Displays version information. Nothing else on the command line is executed. A user name and password is not needed with this option.

See Also

For more information about using `imqcmd` and for command examples, see Chapter 5, "Managing a Broker" and Chapter 6, "Managing Physical Destinations."

imqobjmgr

The Object Manager utility, `imqobjmgr`, creates and manages Message Queue administered objects.

Syntax

```
imqobjmgr subcommand [options]
imqobjmgr -h|H
imqobjmgr -v
```

Subcommands

The Object Manager utility (`imqobjmgr`) includes the subcommands listed in Table 13-3:

Table 13-11 `imqobjmgr` Subcommands

Subcommand	Description
`add`	Adds an administered object to the object store.
`delete`	Deletes an administered object from the object store.
`list`	Lists administered objects in the object store.
`query`	Displays information about the specified administered object.
`update`	Modifies an existing administered object in the object store.

Command Options

Table 13-12 lists the options to the `imqobjmgr` command. For a discussion of their use, see the task-based sections that follow.

Table 13-12 `imqobjmgr` Options

Option	Description
`-f`	Performs action without user confirmation.
`-h`	Displays usage help. Nothing else on the command line is executed.
`-H`	Displays usage help, attribute list, and examples. Nothing else on the command line is executed.
`-i` *fileName*	Specifies the name of an command file containing all or part of the subcommand clause, specifying object type, lookup name, object attributes, object store attributes, or other options. Typically used for repetitive information, such as object store attributes.
`-j` *attribute=value*	Specifies attributes necessary to identify and access a JNDI object store. See "About Object Stores" on page 174.
`-javahome` *path*	Specifies an alternative Java 2 compatible runtime to use (default is to use the runtime on the system or the runtime bundled with Message Queue).
`-l` *lookupName*	Specifies the JNDI lookup name of an administered object. This name must be unique in the object store's context.
`-o` *attribute=value*	Specifies attributes of an administered object. See Chapter 16, "Administered Object Attribute Reference" on page 333.
`-pre`	Preview mode. Indicates what will be done without performing the command.
`-r` *read-only_state*	Specifies whether an administered object is a read-only object. A value of `true` indicates the administered object is a read-only object. Clients cannot modify the attributes of read-only administered objects. The read-only state is set to `false` by default.
`-s`	Silent mode. No output will be displayed.

Table 13-12 `imqobjmgr` Options *(Continued)*

Option	Description
`-t` *objectType*	Specifies the type of a Message Queue administered object: q = queue t = topic cf = connection factory qf = queue connection factory tf = topic connection factory xcf = XA connection factory (distributed transactions) xqf = XA queue connection factory (distributed transactions) xtf = XA topic connection factory (distributed transactions) e = SOAP endpoint (This administered object type is used to support SOAP messages, as described in the *Message Queue Developer's Guide for Java Clients*.)
`-v`	Displays version information. Nothing else on the command line is executed.

See Also

For more information about `imqobjmgr` and for command examples, see Chapter 8, "Managing Administered Objects."

imqdbmgr

The Database Manager utility (`imqdbmgr`) sets up the schema needed for persistence. You can also use the `imqdbmgr` command `to` delete Message Queue database tables that become corrupted or to change the data store.

Syntax

```
imqdbmgr subcommand argument [options]
imqdbmgr -h|-help
imqdbmgr -v|-version
```

Subcommands

The Database Manager utility (`imqdbmgr`) includes the subcommands listed in Table 13-13:

Table 13-13 `imqdbmgr` Subcommands

Subcommand and Argument	Description
`create all`	Creates a new database and Message Queue persistent store schema. This command is used on an embedded database system, and when used, the property `imq.persist.jdbc.createdburl` needs to be specified.
`create tbl`	Creates the Message Queue persistent store schema in an existing database system. This command is used on an external database system.
`delete tbl`	Deletes the existing Message Queue database tables in the current persistent store database.
`delete oldtbl`	Deletes all Message Queue database tables in an earlier version persistent store database. Used after the persistent store has been automatically migrated to the current version of Message Queue.
`recreate tbl`	Deletes the existing Message Queue database tables in the current persistent store database and then re-creates the Message Queue persistent store schema.
`reset lck`	Resets the lock so the persistent store database can be used by other processes.

Command Options

Table 13-14 lists the options to the `imqdbmgr` command.

Table 13-14 `imqdbmgr` Options

Option	Description
`-D`*property=value*	Sets the specified property to the specified value.
`-b` *instanceName*	Specifies the broker instance name and use the corresponding instance configuration file.
`-h`	Displays usage help. Nothing else on the command line is executed.
`-p` *password*	Specifies the database password. This option is being deprecated and will be unsupported in a future release. Use one of the following alternatives: • Omit the password from the command line so that the command prompts you for the password. • Use the `-passfile` option to specify a file containing the database password.
`-passfile` *path*	Specifies the path to a file containing the database password. For more information, see "Using a Passfile" on page 169.
`-u` *name*	Specifies the database user name.
`-v`	Displays version information. Nothing else on the command line is executed.

See Also

For more information about setting up a persistent store, see "Setting Up a Persistent Store" on page 99.

imqusermgr

The User Manager utility (`imqusermgr`) lets you edit or populate a flat-file user repository. Before using `imqusermgr`, 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 need the 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.

Syntax

```
imqusermgr subcommand [options]
imqusermgr -h
imqusermgr -v
```

Subcommands

Table 13-15 lists the `imqusermgr` subcommands whose use is described in this chapter.

Table 13-15 `imqusermgr` Subcommands

Subcommand	Description
`add [-i` *instanceName*`] -u` *userName* `-p` *passwd* `[-g` *group*`] [-s]`	Adds a user and associated password to the specified (or default) broker instance repository, and optionally specifies the user's group.
`delete [-i` *instanceName*`] -u` *userName* `[-s] [-f]`	Deletes the specified user from the specified (or default) broker instance repository.
`list [-i` *instanceName*`] [-u` *userName*`]`	Displays information about the specified user or all users in the specified (or default) broker instance repository.
`update [-i` *instanceName*`] -u` *userName* `-p` *passwd* `[-a` *state*`] [-s] [-f]` `update [-i` *instanceName*`] -u` *userName* `-a` *state* `[-p` *passwd*`] [-s] [-f]`	Updates the password and/or state of the specified user in the specified (or default) broker instance repository.

Command Options

Table 13-16 lists the options to the `imqusermgr` command.

Table 13-16 `imqusermgr` Options

Option	Description
`-a` *active_state*	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 user repository 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.

See Also

For more information about setting up and managing a flat-file user repository, and for `imqusermgr` command examples, see "Using a Flat-File User Repository" on page 142.

imqsvcadmin

The Service Administration (`imqsvcadmin`) utility installs a broker as a Windows service.

Syntax

```
imqsvcadmin subcommand [options]
imqsvcadmin -h
```

Subcommands

The Message Queue Service Administrator utility (`imqsvcadmin`) includes the subcommands listed in Table 13-17:

Table 13-17 `imqsvcadmin` Subcommands

Subcommand	Description
`install`	Installs the service and specifies startup options.
`query`	Displays the startup options to the `imqsvcadmin` command. This includes whether the service is started manually or automatically, its location, the location of the java runtime, and the value of the arguments passed to the broker on startup.
`remove`	Removes the service.

Command Options

Table 13-18 lists the options to the `imqsvcadmin` command.

Table 13-18 `imqsvcadmin` Options

Option	Description
`-h`	Displays usage help. Nothing else on the command line is executed.
`-javahome` *path*	Specifies the path to an alternate Java 2 compatible runtime to use (default is to use the runtime on the system or the runtime bundled with Message Queue. Example: `imqsvcadmin -install -javahome d:\jdk1.4`

Table 13-18 `imqsvcadmin` Options *(Continued)*

Option	Description
`-jrehome` *path*	Specifies the path to a Java 2 compatible JRE. Example: `imqsvcadmin -install -jrehome d:\jre\1.4`
`-vmargs` *arg* [[*arg*]…]	Specifies additional arguments to pass to the Java VM that is running the broker service. (You can also specify these arguments in the Windows Services Control Panel Startup Parameters field.) Example: `-vmargs "-Xms16m -Xmx128m"`
`-args` *arg* [[*arg*]…]	Specifies additional command line arguments to pass to the broker service. For a description of the `imqbrokerd` options, see "imqbrokerd" on page 282. (You can also specify these arguments in the Windows Services Control Panel Startup Parameters field.) For example, `imqsvcadmin -install -args "-passfile d:\imqpassfile"`

The information that you specify using the `-javahome`, `-vmargs`, and `-args` options is stored in the Window's registry under the keys `JREHome`, `JVMArgs`, and `ServiceArgs` in the following path:

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet
    \Services\iMQ_Broker\Parameters
```

See Also

For more information about running Message Queue as a Windows service, see "Automatic Startup on Windows" on page 69.

imqkeytool

The (`imqusermgr` command generates a self-signed certificate for the broker. The same certificate can be used for the `ssljms`, `ssladmin`, or `cluster` connection service. On UNIX systems you may need to run `imqkeytool` from the `superuser` (`root`) account.

Syntax

```
imqkeytool -broker
```

See Also

For more information about setting up secure connections, see "Working With an SSL-Based Service" on page 159.

Chapter 14

Broker Properties Reference

This chapter lists and describes the broker configuration properties. The first section is an alphabetical list of all broker properties, with a reference to the section that contains a full description. All other sections group a set of broker properties by function and provide full descriptions of the properties.

This chapter contains the following sections:

In the description tables, properties are marked if you can set them by using the `imqcmd update bkr` command.

Alphabetical List of Properties

Table 14-1 is an alphabetical list of broker instance properties. Use it to determine the category of any property, and then use the category description to find a full property description elsewhere in this chapter.

In the table, the left column alphabetically lists each property. The right column shows the category to which the property belongs and provides a cross-reference to the appropriate section.

Table 14-1 Broker Instance Configuration Properties

Property Name	Reference
imq.accesscontrol.enabled	"Security Manager Properties" on page 320
imq.accesscontrol.file.filename	"Security Manager Properties" on page 320
imq.audit.enabled	"Security Manager Properties" on page 320
imq.authentication.basic.user_repository	"Security Manager Properties" on page 320
imq.authentication.client.response.timeout	"Security Manager Properties" on page 320
imq.authentication.type	"Security Manager Properties" on page 320
imq.autocreate.destination.isLocalOnly	"Auto-create Configuration Properties" on page 314
imq.autocreate.destination.limitBehavior	"Auto-create Configuration Properties" on page 314
imq.autocreate.destination.maxBytesPerMsg	"Auto-create Configuration Properties" on page 314
imq.autocreate.destination.maxNumMsgs	"Auto-create Configuration Properties" on page 314
imq.autocreate.destination.maxNumProducers	"Auto-create Configuration Properties" on page 314
imq.autocreate.destination.maxTotalMsgBytes	"Auto-create Configuration Properties" on page 314
imq.autocreate.destination.useDMQ	"Auto-create Configuration Properties" on page 314
imq.autocreate.queue	"Auto-create Configuration Properties" on page 314
imq.autocreate.queue.consumerFlowLimit	"Auto-create Configuration Properties" on page 314
imq.autocreate.queue.localDeliveryPreferred	"Auto-create Configuration Properties" on page 314
imq.autocreate.queue.maxNumActiveConsumers	"Auto-create Configuration Properties" on page 314
imq.autocreate.queue.maxNumBackupConsumers	"Auto-create Configuration Properties" on page 314
imq.autocreate.topic	"Auto-create Configuration Properties" on page 314
imq.autocreate.topic.consumerFlowLimit	"Auto-create Configuration Properties" on page 314
imq.cluster.*property_name*	"Cluster Configuration Properties" on page 327
imq.destination.DMQ.truncateBody	"Message Router Properties" on page 313
imq.destination.logDeadMsgs	"Monitoring and Logging Properties" on page 324
imq.hostname	"Connection Service Properties" on page 311
imq.httpjms.http.*property_name*	Table C-3 on page 385
imq.httpsjms.https.*property_name*	Table C-3 on page 385
imq.imqcmd.password	"Security Manager Properties" on page 320
imq.keystore.*property_name*	"Security Manager Properties" on page 320

Table 14-1 Broker Instance Configuration Properties *(Continued)*

Property Name	Reference
imq.log.console.output	"Monitoring Service Properties" on page 324
imq.log.console.stream	"Monitoring Service Properties" on page 324
imq.log.file.dirpath	"Monitoring Service Properties" on page 324
imq.log.file.filename	"Monitoring Service Properties" on page 324
imq.log.file.output	"Monitoring Service Properties" on page 324
imq.log.file.rolloverbytes	"Monitoring Service Properties" on page 324
imq.log.file.rolloversecs	"Monitoring Service Properties" on page 324
imq.log.level	"Monitoring Service Properties" on page 324
imq.log.syslog.facility	"Monitoring Service Properties" on page 324
imq.log.syslog.identity	"Monitoring Service Properties" on page 324
imq.log.syslog.logconsole	"Monitoring Service Properties" on page 324
imq.log.syslog.logpid	"Monitoring Service Properties" on page 324
imq.log.syslog.output	"Monitoring Service Properties" on page 324
imq.log.timezone	"Monitoring Service Properties" on page 324
imq.message.expiration.interval	"Message Router Properties" on page 313
imq.message.max_size	"Message Router Properties" on page 313
imq.metrics.enabled	"Message Router Properties" on page 313
imq.metrics.interval	"Message Router Properties" on page 313
imq.metrics.topic.enabled	"Message Router Properties" on page 313
imq.metrics.topic.interval	"Message Router Properties" on page 313
imq.metrics.topic.persist	"Message Router Properties" on page 313
imq.metrics.topic.timetolive	"Monitoring Service Properties" on page 324
imq.passfile.dirpath	"Security Manager Properties" on page 320
imq.passfile.enabled	"Security Manager Properties" on page 320
imq.passfile.name	"Security Manager Properties" on page 320
imq.persist.file.destination.message.filepool.limit	"Properties for File-Based Persistence" on page 317
imq.persist.file.message.cleanup	"Message Router Properties" on page 313
imq.persist.file.message.filepool.cleanratio	"Message Router Properties" on page 313

Table 14-1 Broker Instance Configuration Properties *(Continued)*

Property Name	Reference
imq.persist.file.message.max_record_size	"Message Router Properties" on page 313
imq.persist.file.sync.enabled	"Properties for File-Based Persistence" on page 317
imq.persist.jdbc.*property_name*	"Persistence Manager Properties" on page 316
imq.persist.store	"Message Router Properties" on page 313
imq.ping.interval	"Connection Service Properties" on page 311
imq.portmapper.backlog	"Connection Service Properties" on page 311
imq.portmapper.hostname	"Connection Service Properties" on page 311
imq.portmapper.port	"Connection Service Properties" on page 311
imq.*resource_state*.count	"Message Router Properties" on page 313
imq.*resource_state*.threshold	"Message Router Properties" on page 313
imq.service.activelist	"Connection Service Properties" on page 311
imq.*service_name*.accesscontrol.enabled	"Security Manager Properties" on page 320
imq.*service_name*.accesscontrol.file.filename	"Security Manager Properties" on page 320
imq.*service_name*.authentication.type	"Security Manager Properties" on page 320
imq.*service_name*.max_threads	"Connection Service Properties" on page 311
imq.*service_name*.min_threads	"Connection Service Properties" on page 311
imq.*service_name*.*protocol_type*.hostname	"Connection Service Properties" on page 311
imq.*service_name*.*protocol_type*.port	"Connection Service Properties" on page 311
imq.*service_name*.threadpool_model	"Connection Service Properties" on page 311
imq.shared.connectionMonitor_limit	"Connection Service Properties" on page 311
imq.system.max_count	"Message Router Properties" on page 313
imq.system.max_size	"Message Router Properties" on page 313
imq.transaction.autorollback	"Message Router Properties" on page 313
imq.user_repository.ldap.*property_name*	"Security Manager Properties" on page 320

Connection Service Properties

Table 14-2 lists the Connection Service properties. The first column lists the property names. For each property name, the second column describes the property, the third column specifies its type, and the fourth column gives its default value.

Table 14-2 Connection Service Properties

Property Name	Description	Type	Default
`imq.service.activelist`	A list of connection services, by name, separated by commas, to be made active at broker startup. Supported services are: `jms`, `ssljms`, `httpjms`, `httpsjms`, `admin`, `ssladmin`.	list	`jms,admin`
`imq.ping.interval`	The period, in seconds, between successive attempts of the broker to ping the Message Queue client runtime across a connection.	integer	`120`
`imq.hostname`	The host (hostname or IP address) to which all connection services bind if there is more than one host available (for example, if there is more than one network interface card in a computer).	string	All available IP addresses
`imq.portmapper.port`[1]	The broker's primary port—the port at which the Port Mapper resides. If you are running more than one broker instance on a host, each must be assigned a unique Port Mapper port.	integer	`7676`
`imq.portmapper.hostname`	The host (hostname or IP address) to which the Port Mapper binds if there is more than one host available (for example, if there is more than one network interface card in a computer).	string	Inherited from `imq.hostname`
`imq.portmapper.backlog`	The maximum number of concurrent requests that the Port Mapper can handle before rejecting requests. The property sets the number of requests that can be stored in the operating system backlog waiting to be handled by the Port Mapper.	integer	`50`
`imq.`*service_name*`.`*protocol_type*[2]`.port`	For jms, ssljms, admin, and ssladmin services only, the port number for the named connection service. To configure the httpjms and httpsjms connection services, see Appendix C, "HTTP/HTTPS Support."	integer	`0` (zero) The port is dynamically allocated by the Port Mapper.
`imq.`*service_name*`.`*protocol_type*[2]`.hostname`	For jms, ssljms, admin, and ssladmin services only, the host (hostname or IP address) to which the named connection service binds if there is more than one host available (for example, if there is more than one network interface card in a computer).	string	Inherited from `imq.hostname`

Table 14-2 Connection Service Properties *(Continued)*

Property Name	Description	Type	Default
`imq.`*service_name*`.min_threads`	The number of threads, which once reached, are maintained in the thread pool for use by the named connection service. The default value varies by connection service.	integer	`10` (jms) `10` (ssljms) `10` (httpjms) `10` (httpsjms) `4` (admin) `4` (ssladmin)
`imq.`*service_name*`.max_threads`	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.	integer	`1000` (jms) `500` (ssljms) `500` (httpjms) `500` (httpsjms) `10` (admin) `10` (ssladmin)
`imq.`*service_name*`.threadpool_model`	A string specifying whether threads are dedicated to connections (`dedicated`) or shared by connections as needed (`shared`) for the named connection service. Shared model (thread pool management) increases the number of connections supported by a broker, but is implemented only for the `jms` and `admin` connection services. The default value varies by connection service.	string	`dedicated` (jms) `dedicated` (ssljms) `dedicated` (httpjms) `dedicated` (httpsjms) `dedicated` (admin) `dedicated` (ssladmin)
`imq.shared.connectionMonitor_limit`	For shared thread pool model only, the maximum number of connections that can be monitored by a distributor thread. (The system allocates enough distributor threads to monitor all connections.) The smaller this value, the faster the system can assign active connections to threads. A value of `-1` means no limit. The default value varies by operating system.	integer	`512` (Solaris & Linux) `64` (Windows)

1. This property can be used with the `imqcmd update bkr` command.
2. *protocol_type* is specified in Table 4-2.

Message Router Properties

Table 14-3 lists the Message Router properties. The first column lists the property names. For each property name, the second column describes the property, the third column specifies its type, and the fourth column gives its default value.

The auto-create properties that configure the message server's ability to automatically create destinations are listed in Table 14-4 on page 314.

Table 14-3 Message Router Properties

Property Name	Description	Type	Default
`imq.destination.DMQ.truncateBody`[1]	A boolean value specifying whether the broker removes the body of a message before storing it in the dead message queue. A value of `true` causes the broker to save just the message header and property data. A value of `false` causes the broker to save the header and body.	boolean	`false`
`imq.message.expiration.interval`	The interval, in seconds, at which reclamation of expired messages occurs.	integer	`60`
`imq.system.max_count`[1]	The maximum number of messages held by the broker. Additional messages will be rejected. A value of -1 means no limit.	integer	`-1`
`imq.system.max_size`[1]	The maximum total size (in bytes, Kbytes, or Mbytes) of messages held by the broker. Additional messages will be rejected. A value of -1 means no limit.	byte string[2]	`-1`
`imq.message.max_size`[1]	The maximum allowed size (in bytes, Kbytes, or Mbytes) of a message body. Any message larger than this will be rejected. A value of -1 means no limit.	byte string[2]	`70m`

`imq.`*resource_state*`.threshold`	The percent memory utilization at which each memory resource state is triggered. The resource state can have the values `green`, `yellow`, `orange`, and `red`.	integer (percent)	`0 (green)`
			`80 (yellow)`
			`90(orange)`
			`98 (red)`

`imq.`*resource_state*`.count`	The maximum number of incoming messages allowed in a batch before system memory is checked to see whether a new memory threshold has been reached. This limit throttles back message producers as system memory becomes increasingly scarce.	integer (percent)	`5000 (green)`
			`500 (yellow)`
			`50(orange)`
			`0 (red)`

Table 14-3 Message Router Properties *(Continued)*

Property Name	Description	Type	Default
`imq.transaction.autorollback`	A boolean value specifying whether distributed transactions left in a `PREPARED` state are automatically rolled back when a broker is started up. If `false`, you must manually commit or roll back transactions using `imqcmd` (see "Managing Transactions" on page 123).	boolean	`false`

1. This property can be used with the `imqcmd update bkr` command.
2. A value that is typed as a *byte string* can be expressed in bytes, Kbytes, and Mbytes: For example: 1000 means 1000 bytes; 7500b means 7500 bytes; 77k means 77 kilobytes (77 x 1024 = 78848 bytes); 17m means 17 megabytes (17 x 1024 x 1024 = 17825792 bytes)

Table 14-4 lists the properties that the broker uses when automatically creating destinations.

Table 14-4 Auto-create Configuration Properties

Property Name	Description	Type	Default
`imq.autocreate.destination.isLocalOnly`	(Applies only to broker clusters.) A boolean value specifying that a destination is not replicated on other brokers, and is therefore limited to delivering messages only to local consumers (consumers connected to the broker on which the destination is created). This attribute cannot be updated once the destination has been created.	boolean	`false`
`imq.autocreate.destination.limitBehavior`	A string specifying how the broker responds when a memory-limit threshold is reached. Values are: • `FLOW_CONTROL` — Slows down producers. • `REMOVE_OLDEST` — Throws out oldest messages. • `REMOVE_LOW_PRIORITY` — Throws out lowest priority messages according to age of the messages. • `REJECT_NEWEST` — Rejects the newest messages. The producing client gets an exception for rejection of persistent messages only. To use this limit behavior with non-persistent messages, set the `imqAckOnProduce` connection factory attribute. If you set this property to `REMOVE_OLDEST` or `REMOVE_LOW_PRIORITY` and set `imq.autocreate.destination.useDMQ` to `true`, the broker moves excess messages to the dead message queue.	string	`REJECT NEWEST`

Table 14-4 Auto-create Configuration Properties *(Continued)*

Property Name	Description	Type	Default
imq.autocreate.destination.maxBytesPerMsg	The maximum size (in bytes) of any single message allowed in an auto-created destination. A value of -1 indicates that message size is unlimited.	byte string[2]	10k
imq.autocreate.destination.maxNumMsgs	The maximum number of unconsumed messages allowed in an auto-created destination. A value of -1 indicates that the number is unlimited.	integer	100,000
imq.autocreate.destination.maxNumProducers	The maximum number of producers allowed for the destination. When this limit is reached, no new producers can be created. A value of -1 indicates that number of producers is unlimited.	integer	100
imq.autocreate.destination.maxTotalMsgBytes	The maximum total amount of memory (in bytes) allowed for unconsumed messages in the destination. A value of -1 indicates that memory is unlimited.	byte string[2]	10m
imq.autocreate.destination.useDMQ	A boolean value specifying whether the broker moves dead messages for auto-created destinations to the dead message queue.	boolean	true
imq.autocreate.queue[1]	A boolean value specifying whether a broker is allowed to auto-create a queue destination.	boolean	true
imq.autocreate.queue.consumerFlowLimit	The maximum number of messages that will be delivered to a 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 commences. This limit can be overridden by a lower value set for the destination's consumers on their respective connections. A value of -1 means an unlimited number.	integer	1000
imq.autocreate.queue.localDeliveryPreferred	(Applies only to load-balanced queue delivery in broker clusters.) A boolean value specifying that messages be delivered to remote consumers only if there are no consumers on the local broker. Requires that the auto-created destination not be restricted to local-only delivery (isLocalOnly = false).	boolean	false
imq.autocreate.queue.maxNumActiveConsumers	The maximum number of consumers that can be active in load-balanced delivery from an auto-created queue destination. A value of -1 means an unlimited number.	integer	1
imq.autocreate.queue.maxNumBackupConsumers	The maximum number of backup consumers that can take the place of active consumers if any fail during load-balanced delivery from an auto-created queue destination. A value of -1 means an unlimited number.	integer	0 (zero)
imq.autocreate.topic	A boolean value specifying whether a broker is allowed to auto-create a topic destination.	boolean	true

Table 14-4 Auto-create Configuration Properties *(Continued)*

Property Name	Description	Type	Default
`imq.autocreate.topic.consumerFlowLimit`	The maximum number of messages that will be delivered to a consumer in a single batch. A value of `-1` means an unlimited number.	integer	1000

1. This property can be used with `imqcmd update bkr`.
2. A value that is typed as a *byte string* can be expressed in bytes, Kbytes, and Mbytes: For example: 1000 means 1000 bytes; 7500b means 7500 bytes; 77k means 77 kilobytes (77 x 1024 = 78848 bytes); 17m means 17 megabytes (17 x 1024 x 1024 = 17825792 bytes)

Persistence Manager Properties

To configure the persistence features of the broker, you must specify the value for `imq.persist.store`, or accept the default value.

Table 14-5 Required Persistence Manager Property

Property Name	Description	Type	Default
`imq.persist.store`	A string specifying whether the broker is using built-in, file-based persistence, or plugged-in JDBC compliant persistence. The value must be `file` or `jdbc`.	string	`file`

The properties that support file-based persistence and JDBC-based persistence are described in the next sections.

File-Based Persistence

Table 14-6 lists the properties that support file-based persistence. The first column lists the property names. For each property name, the second column describes the property, the third column specifies its type, and the fourth column gives its default value.

Table 14-6 Properties for File-Based Persistence

Property Name	Description	Type	Default
`imq.persist.file.sync.enabled`	A boolean value specifying whether persistence operations synchronize in-memory state with the physical storage device. If this property is set to `true`, data loss due to system crash is eliminated, but at the expense of performance of persistence operations. If you are running Sun Cluster and the Sun Cluster Data Service for Message Queue, set this property to `true` for brokers on all cluster nodes.	boolean	`false`
`imq.persist.file.message.max_record_size`	For built-in, file-based persistence, the maximum size of a message that will be added to the message storage file, rather than being stored in a separate file.	byte string[1]	`1m`
`imq.persist.file.destination.message.filepool.limit`	For built-in, file-based persistence, the maximum number of free files available for reuse in the destination file pool. The larger the number the faster the broker can process persistent data. Free files in excess of this value will be deleted. The broker will create and delete additional files, in excess of this limit, as needed.	integer	100
`imq.persist.file.message.filepool.cleanratio`	For built-in, file-based persistence, the percentage of free files in destination file pools that are maintained in a *clean* state (truncated to zero). The higher this value, the more overhead required to clean files during operation, but the less disk space required for the file pool.	integer	`0` (zero)
`imq.persist.file.message.cleanup`	For built-in, file-based persistence, a boolean value specifying whether or not the broker cleans up free files in destination file pools on shutdown. A value of `false` speeds up broker shutdown, but requires more disk space for the file store.	boolean	`false`

1. A value that is typed as a *byte string* can be expressed in bytes, Kbytes, and Mbytes. Examples: 1000 means 1000 bytes; 7500b means 7500 bytes; 77k means 77 kilobytes (77 x 1024 = 78848 bytes); 17m means 17 megabytes (17 x 1024 x 1024 = 17825792 bytes)

JDBC-Based Persistence

Table 14-7 contains the properties that support JDBC-based persistence. The table lists the properties, describes them, and then gives examples of how you would configure use with the PointBase product.

Table 14-7 Properties for JDBC-Based Persistence

Property Name	Description	Example
`imq.persist.store`	A string specifying a file-based or JDBC-based data store.	`jdbc`
`imq.persist.jdbc.brokerid`	*(Optional)* A broker instance identifier that 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. The attribute is usually unnecessary for an embedded database, which stores data for only one broker instance. The identifier must be an alphanumeric string whose length does not exceed the maximum table name length, minus 12, allowed by the database.	(Not needed for PointBase embedded version)
`imq.persist.jdbc.driver`	The java class name of the JDBC driver to connect to the database.	`com.pointbase.jdbc.jdbcUniversalDriver`
`imq.persist.jdbc.opendburl`	The database URL for opening a connection to an existing database.	`jdbc:pointbase:embedded:`*dbName*`;database.home=`*...*`/instances/`*instanceName*`/dbstore`
`imq.persist.jdbc.createdburl`	*(Optional)* The database URL for opening a connection to create a database. This attribute is specified only if the database will be created using `imqdbmgr`.	`jdbc:pointbase:embedded:`*dbName*`;new,database.home=/instances/`*instanceName*`/dbstore`
`imq.persist.jdbc.closedburl`	*(Optional)*The database URL for shutting down the current database connection when the broker is shut down.	Not required for PointBase
`imq.persist.jdbc.user`	*(Optional)* The user name used to open a database connection, if required. For security reasons, the value can be specified instead using command line options: `imqbrokerd -dbuser` and `imqdbmgr -u`	

Table 14-7 Properties for JDBC-Based Persistence *(Continued)*

Property Name	Description	Example
imq.persist.jdbc.needpassword	*(Optional)* A boolean value specifying whether the database requires a password for broker access. A value of true means that a password is required. If you set this option, the imqbrokerd and imqdbmgr commands prompt for the password, unless you use the -passfile option to specify a file that contains the password.	
imq.persist.jdbc.password	*(Optional)* The password for use in opening a database connection, if required. Specify this property only in a passfile.	
imq.persist.jdbc.table.IMQSV35	An SQL command used to create the version table.	CREATE TABLE ${name} (STOREVERSION INTEGER NOT NULL, BROKERID VARCHAR(100))
imq.persist.jdbc.table.IMQCCREC35	An SQL command used to create the configuration change record table.	CREATE TABLE ${name} (RECORDTIME BIGINT NOT NULL, RECORD BLOB(10k))
imq.persist.jdbc.table.IMQDEST35	An SQL command used to create the destination table.	CREATE TABLE ${name} (DID VARCHAR(100) NOT NULL, DEST BLOB(10k), primary key(DID))
imq.persist.jdbc.table.IMQINT35	An SQL command used to create the interest table.	CREATE TABLE ${name} (CUID BIGINT NOT NULL, INTEREST BLOB(10k), primary key(CUID))
imq.persist.jdbc.table.IMQMSG35	An SQL command used to create the message table. The default maximum length for the MSG column is 1 Megabyte (1m). If you expect to have messages that are larger than this, set the length accordingly. If the tables have already been created, you need to recreate them to make the change.	CREATE TABLE ${name} (MID VARCHAR(100) NOT NULL, DID VARCHAR(100), MSGSIZE BIGINT, MSG BLOB(1m), primary key(MID))

Table 14-7 Properties for JDBC-Based Persistence *(Continued)*

Property Name	Description	Example
imq.persist.jdbc.table.IMQPROPS35	An SQL command used to create the property table.	CREATE TABLE ${name} (PROPNAME VARCHAR(100) NOT NULL, PROPVALUE BLOB(10k), primary key(PROPNAME))
imq.persist.jdbc.table.IMQILIST35	An SQL command used to create the interest state table.	CREATE TABLE ${name} (MID VARCHAR(100) NOT NULL, CUID BIGINT, DID VARCHAR(100), STATE INTEGER, primary key(MID, CUID))
imq.persist.jdbc.table.IMQTXN35	An SQL command used to create the transaction table.	CREATE TABLE ${name} (TUID BIGINT NOT NULL, STATE INTEGER, TSTATEOBJ BLOB(10K), primary key(TUID))
imq.persist.jdbc.table.IMQTACK35	An SQL command used to create the transaction acknowledgment table.	CREATE TABLE ${name} (TUID BIGINT NOT NULL, TXNACK BLOB(10k))

Security Manager Properties

Table 14-8 lists the Security Manager properties. The first column lists the property names. For each property name, the second column describes the property, the third column specifies its type, and the fourth column gives its default value.

If you are using SSL, refer to the keystore configuration properties listed in Table 14-9, which follows.

Table 14-8 Security Manager Properties

Property Name	Description	Type	Default
imq.accesscontrol.enabled	A boolean value specifying whether to set access control for all connection services supported by a broker. Indicates whether system will check if an authenticated user has permission to use a connection service or to perform specific Message Queue operations with respect to specific destinations, as specified in the access control properties file.	boolean	true

Table 14-8 Security Manager Properties *(Continued)*

Property Name	Description	Type	Default
`imq.accesscontrol.file.filename`	The name of an access control properties file for all connection services supported by a broker instance. The file name specifies a relative file path to the access control directory (see Appendix A, "Operating System-Specific Locations of Message Queue Data").	string	`accesscontrol.properties`
`imq.audit.enabled`	A boolean value specifying whether to start audit logging (Enterprise Edition only) to the broker log file.	boolean	`false`
`imq.authentication.basic.user_repository`	A string specifying (for base 64 coding) the type of user repository used for authentication, either file-based (`file`) or LDAP (`ldap`).	string	`file`
`imq.authentication.client.response.timeout`	The interval, in seconds, for the system to wait for a client to respond to an authentication request from the broker.	integer	`180`
`imq.authentication.type`	A string specifying whether the password should be passed in base 64 coding (`basic`) or as an MD5 digest (`digest`). Sets encoding for all connection services supported by a broker.	string	`digest`
`imq.imqcmd.password`	The password of an administrative user. The `imqcmd` command utility uses this password to authenticate the user of a command before performing an operation.	string	None
`imq.keystore.`*property_name*	For SSL-based services, a string specifying security properties relating to the SSL keystore. See Table 14-9 on page 324	string	None
`imq.passfile.dirpath`	The path to the directory containing the passfile (depends on operating system).	string	See Appendix A
`imq.passfile.enabled`	A boolean value specifying whether user passwords (for SSL, LDAP, JDBC™) for secure communications are specified in a passfile.	boolean	`false`
`imq.passfile.name`	The name of the passfile.	string	`passfile`
`imq.`*service_name*`.accesscontrol.enabled`	A boolean value specifying whether to set access control for named connection service, overriding broker-wide setting. Indicates whether system will check if an authenticated user has permission to use the named connection service or to perform specific Message Queue operations with respect to specific destinations, as specified in the access control properties file.	boolean	Inherited from the system-wide property `imq.accesscontrol.enabled`.

Table 14-8 Security Manager Properties *(Continued)*

Property Name	Description	Type	Default
`imq.`*service_name*`.accesscontrol.file.filename`	The name of an access control properties file for a named connection service of a broker instance. The file name specifies a relative file path to the access control directory (see Appendix A, "Operating System-Specific Locations of Message Queue Data"). The default value is inherited from the system-wide property `imq.accesscontrol.file.filename`	string	See description
`imq.`*service_name*`.authentication.type`	A string specifying whether the password should be passed in base 64 coding (`basic`) or as an MD5 digest (`digest`). Sets encoding for named connection service, overriding any broker-wide setting. The default value is inherited from the system-wide property `imq.authentication.type`.	string	See description
`imq.user_repository.ldap.base`	The directory base for user entries.	string	None
`imq.user_repository.ldap.gidattr`	The provider-specific attribute identifier whose value is a group name.	string	None
`imq.user_repository.ldap.grpbase`	The directory base for group entries.	string	None
`imq.user_repository.ldap.grpfiltler`	A JNDI search filter (a search query expressed as a logical expression). By specifying a search filter for groups, the broker can narrow the scope of a search and thus make it more efficient. For more information, see the JNDI tutorial at the following location. `http://java.sun.com/products/jndi/tutorial` This property does not have to be set.	string	None
`imq.user_repository.ldap.grpsearch`	A boolean value specifying whether to enable group searches. Consult the documentation provided by your LDAP provider to determine whether you can associate users into groups. Note that nested groups are not supported in Message Queue.	boolean	`false`
`imq.user_repository.ldap.memattr`	The attribute identifier in a group entry whose values are the distinguished names of the group's members.	string	None
`imq.user_repository.ldap.password`	The password associated with the distinguished name used by the broker. Specify this property only in a passfile. If the directory server allows anonymous searches, no password is needed.	string	None

Table 14-8 Security Manager Properties *(Continued)*

Property Name	Description	Type	Default
`imq.user_repository.ldap.principal`	The distinguished name that the broker uses to bind to the directory server for a search. If the directory server allows anonymous searches, this property does not need to be assigned a value.	string	None
`imq.user_repository.ldap.`*property_name*	To be supplied	To be supplied	To be supplied
`imq.user_repository.ldap.server`	The *host:port* for the LDAP server, where *host* specifies the fully qualified DNS name of the host running the directory server and *port* specifies the port number that the directory server is using for communications. 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 begins with `ldap://`. Use this format even if you use SSL and have set the property `imq.user_repository.ldap.ssl.enabled` to `true`. You do not need to specify "`ldaps`" in the address.	string	None
`imq.user_repository.ldap.ssl.enabled`	A boolean value specifying whether the broker should use the SSL protocol when talking to an LDAP server.	boolean	`false`
`imq.user_repository.ldap.timeout`	The time limit for a search, in seconds.	integer	`280`
`imq.user_repository.ldap.uidattr`	The provider-specific attribute identifier whose value uniquely identifies a user. For example: `uid`, `cn`, etc.	string	None
`imq.user_repository.ldap.usrfilter`	A JNDI search filter (a search query expressed as a logical expression). By specifying a search filter for users, the broker can narrow the scope of a search and thus make it more efficient. For more information, see the JNDI tutorial at the following location: `http://java.sun.com/products/jndi/tutorial`. This property does not have to be set.	string	None

The configurable properties for the Message Queue keystore are shown in Table 14-9. Use these properties with SSL.

Table 14-9 Keystore Properties

Property Name	Description	Type	Default
`imq.keystore.file.dirpath`	For SSL-based services, the path to the directory containing the keystore file. Default: see Appendix A, "Operating System-Specific Locations of Message Queue Data."	string	None
`imq.keystore.file.name`	For SSL-based services: the name of the keystore file.	string	`keystore`
`imq.keystore.password`	For SSL-based services: the keystore password. Specify this property only in a passfile.	string	None

Monitoring and Logging Properties

Table 14-10 lists the properties related to monitoring and logging. The first column lists the property names. For each property name, the second column describes the property, the third column specifies its type, and the fourth column gives its default value.

Table 14-10 Monitoring Service Properties

Property Name	Description	Type	Default
`imq.destination.logDeadMsgs`[1]	A boolean value specifying whether the broker logs the following types of events: • A destination is full, having reached its maximum size or maximum message count. • The broker discards a message for a reason other than an administration command or delivery acknowledgment. • The broker moves a message to the dead message queue.	boolean	`false`

Table 14-10 Monitoring Service Properties *(Continued)*

Property Name	Description	Type	Default
imq.log.console.output	A string specifying the categories of logging information that are written to the console. The value can be one of the following: • ALL • NONE • One or more of the following values, separated by vertical bars (|): ERROR, WARNING, and INFO. Specify each category of log message separately; none of the message categories include other categories.	string	ERROR|WARNING
imq.log.console.stream	A string specifying whether console output is written to stdout (OUT) or stderr (ERR).	string	ERR
imq.log.file.dirpath	The path to the directory containing the log file (depends on operating system).	string	See Appendix A
imq.log.file.filename	The name of the log file.	string	log.txt
imq.log.file.output	The categories of logging information to be written to the console. The value can be one of the following: • ALL • NONE • One or more of the following values, separated by vertical bars (|): ERROR, WARNING, and INFO. Specify each category of log message separately; none of the message categories include other categories.	string	ALL
imq.log.file.rolloverbytes[1]	The size, in bytes, of the log file at which output rolls over to a new log file. A value of -1 disables rollover based on file size.	integer	-1
imq.log.file.rolloversecs[1]	The age of the log file, in seconds, at which output rolls over to a new log file. A value of -1 disables rollover based on file age.	integer	604800 (one week)
imq.log.level[1]	A string specifying the logger level: the categories of output that can be written to an output channel. Includes the specified category and all higher level categories as well. Values, from high to low, are: ERROR, WARNING, INFO.	string	INFO

Table 14-10 Monitoring Service Properties *(Continued)*

Property Name	Description	Type	Default
imq.log.syslog.facility	(Solaris only) A string specifying what syslog facility the Message Queue broker should log as. Values mirror those listed in the syslog(3C) man page. Appropriate values for use with Message Queue are: LOG_USER, LOG_DAEMON, and LOG_LOCAL0 through LOG_LOCAL7.	string	LOG_DAEMON
imq.log.syslog.identity	(Solaris only) The identity string that should be added to the front of every message logged to syslog. The default value is imqbrokerd_${imq.*instanceName*}	string	See the description
imq.log.syslog.logconsole	(Solaris only) A boolean value specifying whether to write messages to the system console if they cannot be sent to syslog.	boolean	false
imq.log.syslog.logpid	(Solaris only) A boolean value specifying (true/false) whether to log the broker process ID with the message or not.	boolean	true
imq.log.syslog.output	(Solaris only) A string specifying the categories of logging information that are written to syslogd(1M). The value can be one of the following: • ALL • NONE • One or more of the following values, separated by vertical bars (|): ERROR, WARNING, and INFO. Specify each category of log message separately; none of the message categories include other categories.	string	ERROR
imq.log.timezone	A string representing the time zone for log time stamps. The identifiers are the same as those used by java.util.TimeZone.getTimeZone(). For example: GMT, America/LosAngeles, Europe/Rome, Asia/Tokyo.	string	Local time zone
imq.metrics.enabled	A boolean value specifying whether metrics information is being written to the logger. Does not affect production of metrics messages (see imq.metrics.topic.enabled).	boolean	true

Table 14-10 Monitoring Service Properties *(Continued)*

Property Name	Description	Type	Default
`imq.metrics.interval`	If metrics logging is enabled (`imq.metrics.enabled=true`), the time interval, in seconds, at which metrics information is written to the logger. Does not affect time interval for production of metrics messages (see `imq.metrics.topic.interval`). A value of -1 means never.	integer	`-1`
`imq.metrics.topic.enabled`	A boolean value specifying whether metrics message production is enabled. If `false`, an attempt to subscribe to a metric topic destination will throw a client-side exception.	boolean	`true`
`imq.metrics.topic.interval`	The time interval, in seconds, at which metrics messages are produced (sent to metric topic destinations).	integer	`60`
`imq.metrics.topic.persist`	A boolean value specifying whether metrics messages are persistent.	boolean	`false`
`imq.metrics.topic.timetolive`	The lifetime, in seconds, of metrics messages sent to metric topic destinations.	integer	`300`

1. This property can be used with `imqcmd update bkr`.

Cluster Configuration Properties

Table 14-11 summarizes the configuration properties related to broker clusters.

Table 14-11 Cluster Configuration Properties

Property Name	Description	Type	Default
`imq.cluster.brokerlist`	A comma-separated list of *host:port* entries identifying all the brokers in the cluster, where *host* is the host name of a broker and *port* is its Port Mapper port number. Example: `host1:3000,host2:8000,ctrlhost` Must have the same value for all brokers in a cluster.	string	None

Table 14-11 Cluster Configuration Properties *(Continued)*

Property Name	Description	Type	Default
`imq.cluster.masterbroker`	The host name and port number of the cluster's master broker, if any. The value has the form *host*:*port*, where *host* is the host name of the master broker and *port* is its Port Mapper port number. Example: `ctrlhost:7676` Must have the same value for all brokers in a cluster.	string	None

`imq.cluster.url`[1]	The URL of the 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)
	Must have the same value for all brokers in a cluster.
		string	None

`imq.cluster.hostname`	The host name or IP address to which the `cluster` connection service (used for internal communication between brokers in the cluster) binds if more than one host is available: for example, if there is more than one network interface card in a computer. Can be specified independently for each broker in a cluster.	string	Inherited from the value of `imq.hostname` (see Table 14-2 on page 311)
`imq.cluster.port`	The port number for the `cluster` connection service. Can be specified independently for each broker in a cluster.	integer	`0` (dynamically allocated.)
`imq.cluster.transport`	The network transport protocol used by the `cluster` connection service. For secure, encrypted message delivery between brokers, set this property to `ssl`. Must have the same value for all brokers in a cluster.	string	`tcp`

1. This property can be used with `imqcmd update bkr`.

Chapter 15

Physical Destination Property Reference

This chapter describes the properties you can set for each type of physical destination. You can set the property values when you create or update a physical destination.

For auto-created destinations, you set default values in the broker's instance configuration file (see Table 14-4 on page 314).

Table 15-1 Physical Destination Properties

Property	Destination Type	Default Value	Description
`maxNumMsgs`[1]	Queue Topic	`-1` (unlimited)	The maximum number of unconsumed messages allowed in the destination. For the dead message queue, the default value is 1000.
`maxTotalMsgBytes`[1]	Queue Topic	`-1` (unlimited)	The maximum total amount of memory, in bytes, allowed for unconsumed messages in the destination. The default value for the dead message queue is 10 Mbytes.

Table 15-1 Physical Destination Properties *(Continued)*

Property	Destination Type	Default Value	Description
limitBehavior	Queue Topic	REJECT_NEWEST	A string specifying how the broker responds when a memory-limit threshold is reached. Values are: FLOW_CONTROL — Slows down producers. REMOVE_OLDEST — Throws out the oldest messages. REMOVE_LOW_PRIORITY — Throws out the lowest priority messages according to age of the messages (producing client receives no notification of message deletion). REJECT_NEWEST — Rejects the newest messages. The producing client gets an exception for rejection of persistent messages only. To use this limit behavior with non-persistent messages, set the imqAckOnProduce connection factory attribute. If you set this property to REMOVE_OLDEST or REMOVE_LOW_PRIORITY and set the destination property useDMQ to true, the broker moves excess messages to the dead message queue. The dead message queue itself, unlike other destinations, has the default limit behavior REMOVE_OLDEST and cannot be set to FLOW_CONTROL behavior.
maxBytesPerMsg	Queue Topic	-1 (unlimited)	The maximum size, in bytes, of any single message allowed in the destination. The producing client gets an exception for rejection of persistent messages, but no notification for rejection of non-persistent messages, unless the ackOnProduce property is set.
maxNumProducers	Queue Topic	-1 (unlimited)	The maximum number of producers allowed for the destination. When this limit is reached, no new producers can be created. You cannot set this property for the dead message queue.
maxNumActiveConsumers	Queue only	1	The maximum number of consumers that can be active in load-balanced delivery from a queue destination. A value of -1 means an unlimited number. Platform Edition limits this value to 2.

Table 15-1 Physical Destination Properties *(Continued)*

Property	Destination Type	Default Value	Description
maxNumBackupConsumers	Queue only	0	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. A value of -1 means an unlimited number. Platform Edition limits this value to 1 (one).
consumerFlowLimit	Queue Topic	Topics: 1000 Queues: 1000	The maximum number of messages that will be delivered to a 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 commences. A destination consumer can override this limit by specifying a lower value on a connection. A value of -1 means an unlimited number.
localDeliveryPreferred	Queue only	false	For load-balanced queue delivery in broker clusters, a boolean value specifying whether messages should be delivered to remote consumers only if there are no consumers on the local broker. Requires that the destination not be restricted to local-only delivery (isLocalOnly = false). This property does not apply to the dead message queue.
isLocalOnly	Queue Topic	false	For destinations in broker clusters, a boolean value specifying whether the destination is restricted to local-only delivery. 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). This property cannot be changed once the destination has been created. This property does not apply to the dead message queue.
useDMQ	Queue Topic	true	A boolean value specifying whether dead messages should be sent to the dead message queue, rather than discarded. This property does not apply to the dead message queue.

1. In a cluster environment, this property applies to each instance of the destination in the cluster, rather than collectively to all instances in the cluster.

Chapter 16

Administered Object Attribute Reference

This chapter provides reference information about the attributes of administered objects. It contains the following sections:

Destination Properties

Table 16-1 lists the attributes that configure destination administered objects.

Table 16-1 Destination Administered Object Attributes

Attribute Name	Description	Type	Default
`imqDestinationDescription`	A description for the destination objects.	string	None
`imqDestinationName`	The name of the physical destination.	string[1]	`Untitled_Destination_Object`

1. Destination names can contain only alphanumeric characters (no spaces) and must begin with an alphabetic character or the characters "_" and/or "$".

Connection Factory Attributes

This section contains reference information on the attributes that configure connection factory administered objects. Attributes are categorized into the following sections:

Table 16-2 is an index to the connection factory administered object attributes. The first column alphabetically lists each attribute; the second column lists its category; and the third column is a cross-reference to the table in which the attribute is described.

Connection Handling

Table 16-2 lists the connection factory attributes for connection handling.

Table 16-2 Connection Factory Attributes: Connection Handling

Attribute Name	Description	Type	Default
`imqAddressList`	A comma-separated list of one or more message server addresses. There are several addressing schemes, which are specific to the connection service and port assignment method you want to use. For information about how to specify the address list and examples that illustrate list entries, see "Syntax for the imqAddressList Attribute Value" on page 335.	string	An existing Message Queue 3.0 address if any, or if not, to the first entry in Table 16-3 on page 336.
`imqAddressListBehavior`	A string specifying whether connection attempts are in the order of addresses in the `imqAddressList` attribute (`PRIORITY`) or in a random order (`RANDOM`). If many clients attempt a connection using the same connection factory, you can use a random order to prevent them from all connecting to the same address.	string	`PRIORITY`

Table 16-2 Connection Factory Attributes: Connection Handling *(Continued)*

Attribute Name	Description	Type	Default
imqAddressListIterations	The number of times the client runtime iterates through the imqAddressList in an effort to establish or re-establish a connection. A value of -1 indicates that the number of attempts is unlimited.	integer	5
imqPingInterval	The frequency, in seconds, with which the client runtime tests the connection between an application and broker. A value of -1 or 0 (zero) disables the client runtime from periodically testing the connection.	integer	30
imqReconnectEnabled	A boolean value specifying whether the client runtime should attempt to reconnect to a message server (or the list of addresses in imqAddressList) when a connection is lost.	boolean	false
imqReconnectAttempts	The number of attempts to connect (or reconnect) for each address in the imqAddressList before the client runtime moves on to try the next address in the list. A value of -1 indicates that the number of reconnect attempts is unlimited (the client runtime will attempt to connect to the first address until it succeeds).	integer	0 (zero)
imqReconnectInterval	The interval between reconnect attempts, in milliseconds. This value applies for attempts on each address in the imqAddressList and for successive addresses in the list. If the value is too small, a broker has insufficient recovery time. If the value is too large, the reconnect might represent an unacceptable delay.	long	3000 (milliseconds)
imqSSLIsHostTrusted	A boolean value specifying whether the client can accept a broker's self-signed certificate. To use signed certificates from a certificate authority, set this value to false.	boolean	true

Syntax for the imqAddressList Attribute Value

Each address in the imqbrokerlist value corresponds to a broker instance to which a client runtime can connect.

For each connection service, you specify the broker address differently. The syntax is generally as follows:

scheme://*address_syntax*

To add an address to the list, add a comma and another address. The list can have any number of entries, in the following format:

scheme`://`*address_syntax*,*scheme*`://`*address_syntax*...

The *scheme* variable specifies which of the following addressing types you are using, as described in `Table 16-3`: `mq`, `mqtcp`, `mqssl`, `http`, or `https`. The *address_syntax* variable represents the scheme-specific broker address. Table 16-3 lists the addressing schemes. The first column contains the name of an addressing scheme; the second column shows what connection service is associated with the name; the third column is a description; and the fourth column provides the syntax to use.

Table 16-3 Addressing Schemes for the imqAddressList Attribute

Scheme	Connection Service	Description	Syntax
mq	ms ssljms	Provides dynamic port assignment for use with the jms or ssljms services. You specify the Port Mapper host and port. The Port Mapper dynamically assigns the port used for the connection.	[*hostName*][:*port*][/*serviceName*] For the jms connection service, the following default values apply: *hostName* = `localhost` *port* = `7676` *serviceName* = `jms` For the ssljms connection service, there are no default values. You must specify all variables.
mqtcp	jms	Specifies a port number and uses the jms connection service. Message Queue client runtime makes a tcp connection to the specified host and port to establish a connection.	*hostName*`:`*port*`/jms`
mqssl	ssljms	Specifies a port number and uses the ssljms connection service. Message Queue client runtime makes a secure ssl connection to the specified host and port to establish a connection.	*port*`/ssljms`
http	httpjms	Uses the httpjms connection service. The client runtime 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.	`http://`*hostName*:*port*/*contextRoot*`/tunnel` If multiple broker instances use the same tunnel servlet, this is the syntax for connecting to a specific broker instance, rather than a randomly selected one: `http://`*hostName*:*port*/*contextRoot* /tunnel?ServerName=*hostName*:*instanceName*

Table 16-3 Addressing Schemes for the imqAddressList Attribute

Scheme	Connection Service	Description	Syntax
https	httpsjms	Uses the httpsjms connection service.	`https://`*hostName:port/contextRoot*`/tunnel`
		Message Queue client runtime makes a secure HTTPS connection to the specified Message Queue tunnel servlet URL. The broker must be configured to access the HTTPS tunnel servlet.	If multiple broker instances use the same tunnel servlet, this is the syntax for connecting to a specific broker instance, rather than a randomly selected one: `https://`*hostName:port/contextRoot*`/` `tunnel?ServerName=`*hostName:instanceName*

Table 16-4 contains examples of the addressing formats. The first column is the name of a connection service. The second column specifies whether the host in the example is the local host, an unspecified host, a specified host, or not applicable. The third column specifies whether the port in the example is specified, not specified, or not applicable, and the fourth column is an example.

Table 16-4 Message Server Address Examples

Connection Service	Broker Host	Port	Example Address
Not specified	Local host	Not specified	Default `(mq://localHost:7676/jms)`
Not specified	Specified host	Not specified	`myBkrHost` `(mq://myBkrHost:7676/jms)`
Not specified	Not specified	Portmapper port specified	`1012` `(mq://localHost:1012/jms)`
ssljms	Local host	Portmapper port not specified	`mq://localHost:7676/ssljms`
ssljms	Specified host	Portmapper port	`mq://myBkrHost:7676/ssljms`
ssljms	Specified host	Portmapper port specified	`mq://myBkrHost:1012/ssljms`
jms	Local host	Service port specified	`mqtcp://localhost:1032/jms`
ssljms	Specified host	Service port specified	`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 16-5 lists the connection factory attributes for client identification.

Table 16-5 Connection Factory Attributes: Client Identification

Attribute Name	Description	Type	Default
imqDefaultUsername	The default user name for authenticating with the broker.	string	guest
imqDefaultPassword	The default password for authenticating with the broker.	string	guest
imqConfiguredClientID	An administratively configured client ID.	string	null
imqDisableSetClientID	A boolean value specifying whether to prevent the client from changing the client ID using the setClientID() method in the JMS API.	boolean	false

Message Header Overrides

Table 16-6 lists the connection factory attributes for overriding JMS message header fields.

Table 16-6 Connection Factory Attributes: Message Header Overrides

Attribute Name	Description	Type	Default
imqOverrideJMSDeliveryMode	A boolean value specifying whether the client-set JMSDeliveryMode field can be overridden.	boolean	false
imqJMSDeliveryMode	The override value of JMSDeliveryMode. Values are 1 (non-persistent) and 2 (persistent).	integer	2
imqOverrideJMSExpiration	A boolean value specifying whether the client-set JMSExpiration field can be overridden.	boolean	false
imqJMSExpiration	The override value of JMSExpiration, in milliseconds.	long	0 (no expiration)
imqOverrideJMSPriority	A boolean value specifying whether the client-set JMSPriority field can be overridden.	boolean	false
imqJMSPriority	The override value of JMSPriority (an integer from 0 to 9).	integer	4 (normal)
imqOverrideJMSHeadersTo TemporaryDestinations	A boolean value specifying whether overrides apply to temporary destinations.	boolean	false

Reliability and Flow Control

Table 16-7 lists the connection factory attributes that configure reliability and flow control.

Table 16-7 Connection Factory Attributes: Reliability and Flow Control

Attribute Name	Description	Type	Default
imqAckTimeout	The maximum time, in milliseconds, that the client runtime waits for a broker response before throwing an exception. A value of 0 means there is no time-out and the client runtime waits forever. In some situations, this value can be too low and cause the client runtime to time out. For example, for each user that a broker authenticates using an LDAP user repository and a secure (SSL) connection, the first authentication can take more than 30 seconds.	string	0
imqAckOnProduce	A string specifying how the broker responds to messages from a producing client. If this attribute is set to true, the broker responds to receipt of all JMS messages (persistent and non-persistent) from the producing client. The producing client thread blocks while waiting for those responses. If this attribute is not specified, the broker responds to persistent messages only. The producing client thread blocks while waiting for those responses.	string	not specified
imqConnectionFlowCount	The number of JMS messages in a metered batch. When this number of JMS messages is delivered to the client runtime, delivery is temporarily suspended, allowing any control messages that had been held up to be delivered. Payload message delivery is resumed upon notification by the client runtime, and continues until the count is again reached. If the count is set to 0 then there is no restriction in the number of JMS messages in a metered batch. A non-zero setting allows the client runtime to meter message flow so that Message Queue control messages are not blocked by heavy JMS message delivery.	integer	100
imqConnectionFlowLimitEnabled	A boolean value specifying whether to use the value of imqConnectionFlowLimit to limit message flow at the connection level.	boolean	false

Table 16-7 Connection Factory Attributes: Reliability and Flow Control *(Continued)*

Attribute Name	Description	Type	Default
imqConnectionFlowLimit	The maximum number of messages that can be delivered over a connection and buffered in the client runtime, waiting to be consumed. Note however, that unless imqConnectionFlowIsLimited is enabled, this limit is not checked. When the number of JMS messages delivered to the client runtime (in accordance with the flow metering governed by imqConnectionFlowCount) exceeds this limit, message delivery stops. It is resumed only when the number of unconsumed messages drops below the value set with this attribute. This limit prevents a consuming client that is taking a long time to process messages from being overwhelmed with pending messages that might cause it to run out of memory.	integer	1000
imqConsumerFlowLimit	The maximum number of messages per consumer that can be delivered over a connection and buffered in the client runtime, waiting to be consumed. This limit is used to improve load-balancing among consumers in multi-consumer queue delivery situations (no one consumer can be sent a disproportionate number of messages). This limit can be overridden by a lower value set on the broker side for the queue's consumerFlowLimit attribute (see information on destination attributes in the *Message Queue Administration Guide*). This limit also helps prevent any one consumer on a connection from starving other consumers on the connection. When the number of JMS messages delivered to the client runtime exceeds this limit 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. (Note that if the total number of messages buffered for *all* consumers on a connection exceeds the imqConnectionFlowLimit, delivery of messages through the connection will stop until that total drops below the connection limit.)	integer	100
imqConsumerFlow Threshold	The number of messages per consumer buffered in the client runtime, as a percentage of imqConsumerFlowLimit, below which delivery of messages for a consumer will resume.	integer	50

Queue Browser Behavior and Server Session

Table 16-8 describes attributes that affect queue browsing for clients.

Table 16-8 Connection Factory Attributes: Queue Browser Behavior

Attribute Name	Description	Type	Default
imqQueueBrowserMax MessagesPerRetrieve	The maximum number of messages that the client runtime will retrieve at one time when browsing the contents of a queue destination.	integer	1000
imqQueueBrowserRetrieveTimeout	The maximum time, in milliseconds, that the client runtime will wait to retrieve messages, when browsing the contents of a queue destination, before throwing an exception.	long	60000
imqLoadMaxToServerSession	For JMS application server facilities, a boolean value specifying whether a Message Queue connection consumer should load up to the maxMessages number of messages into a ServerSession's session. If false, the client will load only a single message at a time.		true

JMS-Defined Properties Support

JMS-defined properties are names reserved by JMS, and which a JMS provider can choose to support to enhance client programming capabilities. Table 16-9 describes the JMS-defined properties supported by Message Queue.

Table 16-9 Connection Factory Attributes: JMS-defined Properties Support

Property Name	Description	Type	Default
imqSetJMSXUserID	A boolean value specifying whether to set the JMS-defined property JMSXUserID (identity of user sending the message) on produced messages.	boolean	false
imqSetJMSXAppID	A boolean value specifying whether to set the JMS-defined property JMSXAppID (identity of application sending the message) on produced messages.	boolean	false
imqSetJMSXProducerTXID	A boolean value specifieying whether to set the JMS-defined property JMSXProducerTXID (transaction identifier of the transaction within which this message was produced) on produced messages.	boolean	false
imqSetJMSXConsumerTXID	A boolean value specifying whether to set the JMS-defined property JMSXConsumerTXID (transaction identifier of the transaction within which this message was consumed) on consumed messages.	boolean	false
imqSetJMSXRcvTimestamp	A boolean value specifying whether to set the JMS-defined property JMSXRcvTimestamp (the time the message is delivered to the consumer) on consumed messages.	boolean	false

SOAP Endpoint Attributes

Table 16-10 lists the attributes that configure endpoint URLs for applications that use SOAP. For information on applications that use SOAP, see the *Message Queue Developer's Guide for Java Clients*.

Table 16-10 SOAP Endpoint Attributes

Attribute Name	Description	Type	Default
`imqSOAPEndpointList`	A list of one of more space-separated URLs representing SOAP endpoints to which to send messages. If you specify more than one URL, messages are broadcast to all URLs in the list. Each URL should be associated with a servlet that can receive and process a SOAP message. Example: `http://www.myServlet/ http://www.myServlet2/`	string	
`imqEndpointName`	The name of the SOAP endpoint. Example: MyTopicEndpoint	string	`Untitled_ Endpoint_ Object`
`imqEndpointDescription`	A description of the SOAP endpoint. Example: `"imqEndpointDescription=my endpoints for broadcast"`	string	`A description for the endpoint object`

Chapter 17

JMS Resource Adapter Attribute 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 attributes through three JavaBean components:

- `ResourceAdapter` configuration affects the behavior of the resource adapter as a whole.
- `ManagedConnectionFactory` configuration affects connections created by he 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 attribute 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 attributes of the Message Queue JMS resource adapter. It contains the following sections:

ResourceAdapter JavaBean

The `ResourceAdapter` configuration configures the default JMS resource adapter behavior. Table 17-1 lists and describes the attributes with which you can configure this JavaBean. A footnote marks each required property.

Table 17-1 Resource Adapter Attributes

Name	Description	Default
`addressList`[1]	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 attribute name, `addressList`, is specific to Sun Java System Message Queue, but has the same meaning as the standard attribute `connectionURL`. Sun Java System Message Queue provides both attribute names. You must set either `connectionURL` or `addressList`; they are equivalent.	`mq://localhost:7676/jms`
`addressListBehavior`	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.	`PRIORITY`
`addressListIterations`	The number of times to iterate through the address list. This value applies to the initial connection and to subsequent reconnection attempts.	`1`
`connectionURL`	The connection that the resource adapter makes to the Message Queue service, specified using the message service address format. Equivalent to the `addressList` attribute; see description above for further details.	`mq://localhost:7676/jms`
`userName`[1]	The default user name with which the resource adapter connects to the Message Queue service. The resource adapter supplies the default value.	`guest`

Table 17-1 Resource Adapter Attributes *(Continued)*

Name	Description	Default
`password`[1]	The default password with which the resource adapter connects to the Message Queue service. The resource adapter supplies the default value.	`guest`
`reconnectAttempts`	The number of times to attempt reconnection to a single entry in the address list. This attribute is used when `reconnectEnabled` is set to `true`.	`6`
`reconnectEnabled`	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`.	`false`
`reconnectInterval`	The interval between reconnection attempts, in milliseconds. This attribute is used when `reconnectEnabled` is set to `true`.	`30000`

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 attribute, 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

Name	Description	Default
`addressList`	A list of connections derived from this managed connection factory. The format of this property adheres to the Message Service `addressList`, as described in Table 17-1 on page 344. If this value is not set, connections use the `addressList` value specified for the `ResourceAdapter` JavaBean and described in that table.	None

Table 17-2 Managed Connection Factory Attributes *(Continued)*

Name	Description	Default
addressListBehavior	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.	PRIORITY
addressListIterations	The number of times to iterate through the address list. This value applies to the initial connection and to subsequent reconnection attempts.	1
clientID	The client identifier to use for connections derived from this managed connection factory.	None
password	*(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 344.	guest
reconnectAttempts	The number of times to attempt reconnection to a single entry in the address list.	6
reconnectEnabled	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 properties.	false
reconnectInterval	The minimum number of milliseconds to wait between attempts to reconnect to the Message Queue service.	30000
userName	*(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 344.	guest

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 attributes 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 Attributes

Name	Description	Default
`acknowledgeMode`	*(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`.	`Auto-acknowledge`
`addressList`	*(Optional)* The specification of the connection made by the resource adapter on behalf of the message endpoint. This attribute is specific to the Message Queue JMS resource adapter. The valid values must conform to the message service connection address syntax.	Inherited from `addressList` in the `ResourceAdapter` JavaBean configuration
`clientId`	The JMS client ID to be used by the JMS connection created for this consumer. You must set this attribute if you set `subscriptionDurability` attribute to `Durable`. This is a standard EJB 2.1 and J2EE CA 1.5 property.	None
`customAcknowledgeMode`	A string specifying the mode for MDB message consumption. The valid values for this attribute 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.	None
`destination`	The name of the destination from which this MDB consumes messages. This is a required attribute. 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.	None

Table 17-3 Activation Specification Attributes *(Continued)*

Name	Description	Default
destinationType	The type of destination specified by the destination attribute. Valid values are javax.jms.Queue or javax.jms.Topic. This is a required attribute. It is a standard EJB 2.1 and J2EE CA1.5 property.	None
endpointExceptionRedeliveryAttempts	The number of times to redeliver a message to the MDB when the MDB throws an exception during message delivery.	6
messageSelector	*(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.	None
sendUndeliverableMsgsToDMQ	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.	true
subscriptionDurability	A string specifying whether a consumer for a topic destination is durable or nondurable. The value can be NonDurable or Durable. This attribute is optional for nondurable subscriptions and required for durable subscriptions. If you set this value to Durable, you must also set the attributes clientID and subscriptionName. This is a standard EJB 2.1 and J2EE CA1.5 property and is valid only if the destinationType attribute is set to avax.jms.Topic.	NonDurable
subscriptionName	A string to use to name durable subscriptions. You must set this attribute if you set subscriptionDurability attribute to Durable. This is a standard EJB 2.1 and J2EE CA 1.5 property.	None

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 216.

Broker-wide Metrics

Table 18-2 lists and describes the data the broker reports regarding broker-wide metrics information. It also shows which of the data can be obtained using the different metrics monitoring tools.

Table 18-2 Broker-wide 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 JMS messages currently stored in broker memory and persistent store	No Use `query bkr`	No	Yes (…broker)
Total message bytes	Number of JMS 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 JMS messages that have flowed into the broker since it was last started	Yes (`ttl`)	Yes	Yes (…broker)
Message bytes in	Number of JMS 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 JMS messages and control messages	Yes (`ttl`)	Yes	Yes (…broker)
Packet bytes in	Number of packet bytes that have flowed into the broker since it was last started; includes both JMS messages and control messages	Yes (`ttl`)	Yes	Yes (…broker)

Table 18-2 Broker-wide Metrics *(Continued)*

Metric Quantity	Description	imqcmd `metrics bkr` (*metricType*)	Log File	Metrics Message (*metrics topic*)[1]
Num messages out	Number of JMS messages that have flowed out of the broker since it was last started.	Yes (`ttl`)	Yes	Yes (…broker)
Message bytes out	Number of JMS 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 JMS 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 JMS messages and control messages	Yes (`ttl`)	Yes	Yes (…broker)
Rate messages in	Current rate of flow of JMS messages into the broker	Yes (`rts`)	Yes	No
Rate message bytes in	Current rate of flow of JMS message bytes into the broker	Yes (`rts`)	Yes	No
Rate packets in	Current rate of flow of packets into the broker; includes both JMS messages and control messages	Yes (`rts`)	Yes	No
Rate packet bytes in	Current rate of flow of packet bytes into the broker; includes both JMS messages and control messages	Yes (`rts`)	Yes	No
Rate messages out	Current rate of flow of JMS messages out of the broker	Yes (`rts`)	Yes	No
Rate message bytes out	Current rate of flow of JMS 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 JMS 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 JMS messages and control messages	Yes (`rts`)	Yes	No
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 216.

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 JMS messages that have flowed into the connection service since the broker was last started	Yes (`ttl`)	No	No
Message bytes in	Number of JMS 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 JMS 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 JMS messages and control messages	Yes (`ttl`)	No	No

Table 18-3 Connection Service Metrics *(Continued)*

Metric Quantity	Description	imqcmd `metrics svc` (*metricType*)	Log File	Metrics Message (*metrics topic*)
Num messages out	Number of JMS messages that have flowed out of the connection service since the broker was last started.	Yes (`ttl`)	No	No
Message bytes out	Number of JMS 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 JMS 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 JMS messages and control messages	Yes (`ttl`)	No	No
Rate messages in	Current rate of flow of JMS messages into the broker through the connection service.	Yes (`rts`)	No	No
Rate message bytes in	Current rate of flow of JMS 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 JMS 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 JMS messages and control messages	Yes (`rts`)	No	No
Rate messages out	Current rate of flow of JMS messages out of the connection service	Yes (`rts`)	No	No
Rate message bytes out	Current rate of flow of JMS 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 JMS 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 JMS 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*)
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 JMS messages currently stored in destination memory and persistent store	Yes (`con`) (`ttl`) (`rts`) Also `query dst`	No	Yes (…*destName*)

Table 18-4 Destination Metrics *(Continued)*

Metric Quantity	Description	imqcmd metrics dst (*metricType*)	Log File	Metrics Message (*metrics topic*)[1]
Avg num messages	Average number of JMS 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 JMS 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 JMS 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 JMS 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 JMS message bytes stored in destination memory and persistent store since the broker was last started	Yes (ttl) (rts)	No	Yes (…*destName*)
Peak message bytes	Peak number of JMS 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 JMS messages that have flowed into this destination since the broker was last started	Yes (ttl)	No	Yes (…*destName*)
Msg bytes in	Number of JMS message bytes that have flowed into this destination since the broker was last started	Yes (ttl)	No	Yes (…*destName*)
Num messages out	Number of JMS messages that have flowed out of this destination since the broker was last started	Yes (ttl)	No	Yes (…*destName*)
Msg bytes out	Number of JMS 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 JMS messages into the destination	Yes (rts)	No	No

Table 18-4 Destination Metrics *(Continued)*

Metric Quantity	Description	imqcmd `metrics dst` (*metricType*)	Log File	Metrics Message (*metrics topic*)[1]
Rate num messages out	Current rate of flow of JMS messages out of the destination	Yes (`rts`)	No	No
Rate msg bytes in	Current rate of flow of JMS message bytes into the destination	Yes (`rts`)	No	No
Rate Msg bytes out	Current rate of flow of JMS 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 216.

Part IV

Appendixes

Appendix A

Operating System-Specific Locations of Message Queue Data

Sun Java System Message Queue data is stored in different locations on different operating systems, as the following sections show.

This appendix provides the location of various types of Message Queue data on the following operating systems:

- "Solaris" on page 359
- "Linux" on page 361
- "Windows" on page 362

In the tables that follow, *instanceName* identifies 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 the structure described under "Windows" on page 362.

Table A-1 Location of Message Queue Data on Solaris

Data Category	Location on Solaris
Broker instance configuration properties	`/var/imq/instances/`*instanceName*`/props/config.properties`

Table A-1 Location of Message Queue Data on Solaris *(Continued)*

Data Category	Location on Solaris
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: passfile directory (default location)	`/var/imq/instances/`*instanceName*`/etc/`
Security: example passfile	`/etc/imq/passfile.sample`
Security: broker's keystore 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 Location of Message Queue Data on Linux

Data Category	Location on Windows
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: passfile directory (default location)	`/var/opt/sun/mq/instances/`*instanceName*`/etc/`
Security: example passfile	`/etc/opt/sun/mq/passfile.sample`
Security: broker's keystore 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 shows the location of Message Queue data on Solaris, 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 (/). For more information, see the definitions for `IMQ_HOME` and `IMQ_VARHOME` in Table 3 on page 25.

Table A-3 Location of Message Queue Data on Windows

Data Category	Location on Windows
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)	`IMQ_VARHOME\instances\`*instanceName*`\etc\accesscontrol.properties`
Security: passfile directory (default location)	`IMQ_HOME\etc\`
Security: example passfile	`IMQ_HOME\etc\passfile.sample`
Security: broker's keystore file location	`IMQ_HOME\etc\`

Table A-3 Location of Message Queue Data on Windows *(Continued)*

Data Category	Location on Windows
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 Files 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:

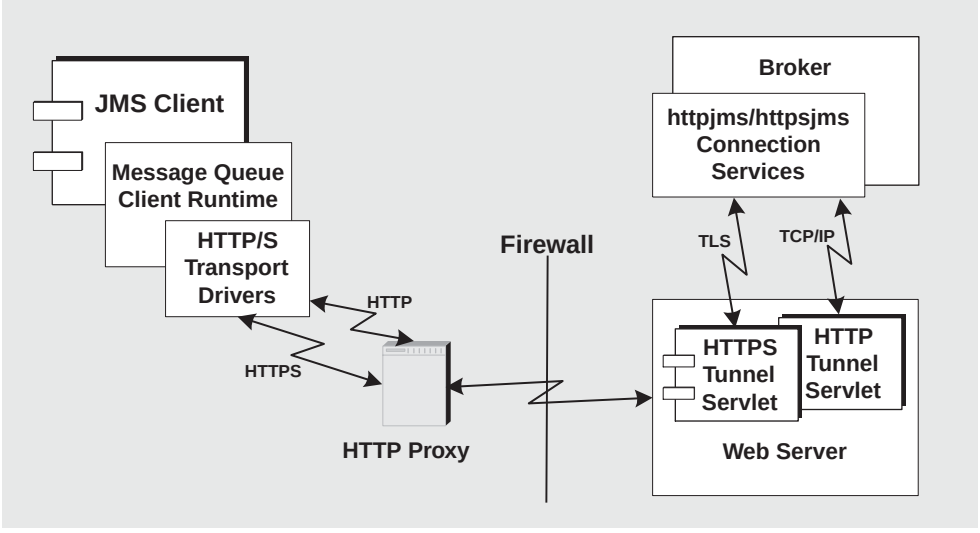
HTTP/HTTPS Support Architecture

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 370 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 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 376 for more information.
- An HTTP or HTTPS tunnel servlet (both bundled with Message Queue) is loaded in the web server and used to pull JMS 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 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 and the broker fail, and the broker is not restarted, the Web 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.

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 on a web server.
2. Configure the broker's httpjms connection service and start the broker.
3. Configure an HTTP connection.

Step 1. Deploying the HTTP Tunnel Servlet on a Web Server

There are two general ways you can deploy the HTTP tunnel servlet on a web server:

- deploying it as a jar file—for web servers that support Servlet 2.1 or earlier
- deploying it as a web archive (WAR) file—for web servers that support Servlet 2.2 or later

Deploying as a Jar File

Deploying the Message Queue tunnel servlet consists of making the appropriate jar files accessible to the host web server, configuring the web server to load the servlet on startup, and specifying the context root portion of the servlet's URL.

The tunnel servlet jar file (`imqservlet.jar`) contains all the classes needed by the HTTP tunnel servlet, and can be found in a directory that depends upon operating system (see Appendix A, "Operating System-Specific Locations of Message Queue Data").

Any web server with servlet 2.x support can be used to load this servlet. The servlet class name is:

```
com.sun.messaging.jmq.transport.
httptunnel.servlet.HttpTunnelServlet
```

The web server must be able to see the `imqservlet.jar` file. If you are planning to run the web server and the broker on different hosts, you should place a copy of the `imqservlet.jar` file in a location where the web server can access it.

You also need to configure the web server to load this servlet on startup, and you might need to specify the context root portion of the servlet's URL (see "Example 1: Deploying the HTTP Tunnel Servlet on Sun Java System Web Server" on page 376).

It is also recommended that you disable your web server's access logging feature in order to improve performance.

Deploying as a Web Archive File

Deploying the HTTP tunnel servlet as a WAR file consists of using the deployment mechanism provided by the web 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, "Operating System-Specific Locations of Message Queue Data").

The WAR file includes a deployment descriptor that contains the basic configuration information needed by the web server to load and run the servlet. Depending on the web server, you might also need to specify the context root portion of the servlet's URL (see "Example 2: Deploying the HTTP Tunnel Servlet on Sun Java System Application Server 7.0" on page 380).

Step 2. Configuring 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 Interactively" on page 67.

➤ **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, "Operating System-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 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 on page 373.

Table C-1 httpjms Connection Service Properties

Property Name	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`

Table C-1 httpjms Connection Service Properties *(Continued)*

Property Name	Description
`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`
`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 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.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. Configuring 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:port/contextRoot`*`/tunnel`

where *hostName:port* is the name and port of the web server hosting the HTTP 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 189), 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 a 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 and servlet instances if you are running multiple brokers. You can share a single web 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:port/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 2002
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.

Example 1: Deploying the HTTP Tunnel Servlet on Sun Java System Web Server

This section describes how you deploy the HTTP tunnel servlet both as a jar file and as a WAR file on the Sun Java System Web Server. The approach you use depends on the version of Sun Java System Web Server: If it does not support Servlet 2.2 or later, it will not be able to handle WAR file deployment.

Deploying as a Jar File

The instructions below refer to deployment on Sun Java System Web Server 6.1 using the browser-based administration GUI. This procedure consists of the following general steps:

1. add a servlet
2. configure the servlet virtual path
3. load the servlet
4. disable the servlet access log

These steps are described in the following subsections. You can verify successful HTTP tunnel servlet deployment by accessing the servlet URL using a web browser. It should display status information.

Adding a Servlet

➤ **To Add a Tunnel Servlet**

1. Select the Servlets tab.
2. Choose Configure Servlet Attributes.
3. Specify a name for the tunnel servlet in the Servlet Name field.

4. Set the Servlet Code (class name) field to the following value:

   `com.sun.messaging.jmq.transport.httptunnel.servlet.HttpTunnelServlet`

5. Enter the complete path to the `imqservlet.jar` in the Servlet Classpath field. For example:

   `/usr/share/lib/imq/imqservlet.jar` (Solaris)
   `/opt/sun/mq/share/lib/imqservlet.jar` (Linux)
   `IMQ_HOME/lib/imqservlet.jar` (Windows)

6. In the Servlet args field, enter any optional arguments, as shown in Table C-2:

Table C-2 Servlet Arguments for Deploying HTTP Tunnel Servlet Jar File

Argument	Default Value	Reference
servletHost	all hosts	See Table C-1 on page 373
servletPort	7675	See Table C-1 on page 373

If using both arguments, separate them with a comma:

`servletPort=`*portNumber*`, servletHost=`*...*

The `servletHost` and `servletPort` argument apply only to communication between the Web Server and broker, and are set only if the default values are problematic. However, in that case, you also must set the broker configuration properties accordingly (see Table C-1 on page 373), for example:

`imq.httpjms.http.servletPort`

Configuring a Servlet Virtual Path (Servlet URL)

➤ **To Configure a Virtual Path (Servlet URL) for a Tunnel Servlet**

1. Select the Servlets tab.
2. Choose Configure Servlet Virtual Path Translation.
3. Set the Virtual Path field.

   The Virtual Path is the */contextRoot*/`tunnel` portion of the tunnel servlet URL:

   `http://`*hostName*`:`*port*`/`*contextRoot*`/tunnel`

   For example, if you set the *contextRoot* to `imq`, the Virtual Path field would be:

   `/imq/tunnel`

4. Set the Servlet Name field to the same value as in step 3 in "Adding a Servlet" on page 376.

Loading a Servlet

➤ **To Load the Tunnel Servlet at Web Server Startup**

1. Select the Servlets tab.
2. Choose Configure Global Attributes.
3. In the Startup Servlets field, enter the same servlet name value as in step 3 in "Adding a Servlet" on page 376.

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.
3. Use the Log client accesses control to disable logging

Deploying as a WAR File

The instructions below refer to deployment on Sun Java System Web Server 6.0 Service Pack 2. 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, "Operating System-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*`:`*port*`/`*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*`:`*port*`/`*contextRoot*`/tunnel`

Clients can now use this URL to connect to the message service using an HTTP connection.

Example 2: Deploying the HTTP Tunnel Servlet on Sun Java System Application Server 7.0

This section describes how you deploy the HTTP tunnel servlet as a WAR file on the Sun Java System Application Server 7.0.

Two steps are required:

- deploy the HTTP tunnel servlet using the Application Server 7.0 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 7.0 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`).

   The location of the `imqhttp.war` file depends on your operating system (see Appendix A, "Operating System-Specific Locations of Message Queue Data")

4. Click OK.
5. On the next screen, set the value for the Context Root text field.

   The Context Root field value is the */contextRoot* portion of the tunnel servlet URL:

   `http://`*hostName*`:`*port*`/`*contextRoot*`/tunnel`

   For example, you could set the Context Root field to `/imq`.

6. Click OK.

   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/SUNWappserver7/domains/domain1/server1/applications/
   j2ee-modules/imqhttp_1
   ```

The servlet is now available at the following address:

`http://`*hostName*`:`*port*`/`*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 7.0 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/SUNWappserver7/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/SUNWappserver7/domains/domain1/server1/
            applications/j2ee-modules/imqhttp_1/-”
{
    permission java.net.SocketPermission "*",
            “connect,accept,resolve";
};
```

Enabling HTTPS Support

The following sections describe the steps you need to take to enable HTTPS support. They are similar to those in "Enabling HTTP Support" on page 371 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. Deploy the HTTPS tunnel servlet on a web server.
3. Configure the broker's httpsjms connection service and start the broker.
4. 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 `imqkeytool` utility to generate a self-signed certificate for the tunnel servlet. Enter the following at the command prompt:

```
imqkeytool -servlet keystore_location
```

The utility will prompt you for the information it needs. (On Unix systems you may need to run `imqkeytool` as the superuser (root) in order to have permission to create the keystore.)

First, `imqkeytool` prompts you for a keystore 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 keystore password.

NOTE Remember the password you provide—you must provide this password later to the tunnel servlet so it can open the keystore.

Running `imqkeytool` runs the JDK `keytool` utility to generate a self-signed certificate and to place it in Message Queue's keystore file located as specified in the *keystore_location* argument. (The keystore is in the same keystore format as that supported by the JDK1.2 `keytool`.)

NOTE The HTTPS tunnel servlet must be able to see the keystore. Make sure you move/copy the generated keystore located in *keystore_location* to a location accessible by the HTTPS tunnel servlet (see "Step 2. Deploying the HTTPS Tunnel Servlet on a Web Server" on page 383).

Step 2. Deploying the HTTPS Tunnel Servlet on a Web Server

There are two general ways you can deploy the HTTPS tunnel servlet on a web server:

- deploying it as a jar file—for web servers that support Servlet 2.1 or earlier
- deploying it as a web archive (WAR) file—for web servers that support Servlet 2.2 or later

In either case, 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 Jar File

Deploying the Message Queue tunnel servlet consists of making the appropriate jar files accessible to the host web server, configuring the web server to load the servlet on startup, and specifying the context root portion of the servlet's URL.

The tunnel servlet jar file (`imqservlet.jar`) contains all the classes needed by the HTTPS tunnel servlet, and can be found in a directory that depends upon operating system (see Appendix A, "Operating System-Specific Locations of Message Queue Data").

Any web server with servlet 2.x support can be used to load this servlet. The servlet class name is:

```
com.sun.messaging.jmq.transport.
httptunnel.servlet.HttpsTunnelServlet
```

The web server must be able to see the `imqservlet.jar` file. If you are planning to run the web server and the broker on different hosts, you should place a copy of the `imqservlet.jar` file in a location where the web server can access it.

You also need to configure the web server to load this servlet on startup, and you might need to specify the context root portion of the servlet's URL (see "Example 3: Deploying the HTTPS Tunnel Servlet on Sun Java System Web Server" on page 389).

Make sure that the JSSE jar files are in the classpath for running servlets in the web server. Check the web server's documentation for how to do this.

An important aspect of configuring the web server is specifying the location and password of the self-signed certificate to be used by the HTTPS tunnel servlet to establish a secure connection with a broker. You must place the keystore created in "Step 1. Generating a Self-signed Certificate for the HTTPS Tunnel Servlet" on page 382 in a location accessible by the HTTPS tunnel servlet.

It is also recommended that you disable your web server's access logging feature in order to improve performance.

Deploying as a Web Archive File

Deploying the HTTPS tunnel servlet as a WAR file consists of using the deployment mechanism provided by the web server. The HTTPS tunnel servlet WAR file (`imqhttps.war`) is located in a directory that depends on your operating system (see Appendix A, "Operating System-Specific Locations of Message Queue Data").

The WAR file includes a deployment descriptor that contains the basic configuration information needed by the web server to load and run the servlet. Depending on the web server, you might also need to specify the context root portion of the servlet's URL (see "Example 4: Deploying the HTTPS Tunnel Servlet on Sun Java System Application Server 7.0" on page 394).

However, the deployment descriptor of the `imqhttps.war` file cannot know where you have placed the keystore file needed by the tunnel servlet (see "Step 1. Generating a Self-signed Certificate for the HTTPS Tunnel Servlet" on page 382). This requires you to edit the tunnel servlet's deployment descriptor (an XML file) to specify the keystore location before deploying the `imqhttps.war` file.

Step 3. 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 Interactively" on page 67.

➤ **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, "Operating System-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-3.

Table C-3 httpsjms Connection Service Properties

Property Name	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`

Table C-3 httpsjms Connection Service Properties *(Continued)*

Property Name	Description
`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 4. 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 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 *cert_file* and that *trust_store_file* is your keystore, run the following command:

```
JRE_HOME/bin/keytool -import -trustcacerts
-alias alias_for_certificate -file cert_file
-keystore trust_store_file
```

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=`*trust_store_file*

`javax.net.ssl.trustStorePassword=`*trust_store_passwd*

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:port/contextRoot*`/tunnel`

where *hostName:port* 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 189), 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:port/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.

Example 3: Deploying the HTTPS Tunnel Servlet on Sun Java System Web Server

This section describes how you deploy the HTTPS tunnel servlet both as a jar file and as a WAR file on the Sun Java System Web Server. The approach you use depends on the version of Sun Java System Web Server: If it does not support Servlet 2.2 or later, it will not be able to handle WAR file deployment.

Deploying as a Jar File

The instructions below refer to deployment on Sun Java System Web Server 6.1 using the browser-based administration GUI. This procedure consists of the following general steps:

1. add a servlet
2. configure the servlet virtual path
3. load the servlet
4. disable the servlet access log

These steps are described in the following subsections. You can verify successful HTTPS tunnel servlet deployment by accessing the servlet URL using a web browser. It should display status information.

Adding a Servlet

➤ **To Add a Tunnel Servlet**

1. Select the Servlets tab.
2. Choose Configure Servlet Attributes.
3. Specify a name for the tunnel servlet in the Servlet Name field.

4. Set the Servlet Code (class name) field to the following value:

   `com.sun.messaging.jmq.transport.`
   `httptunnel.servlet.HttpsTunnelServlet`

5. Enter the complete path to the `imqservlet.jar` in the Servlet Classpath field. For example:

   `/usr/share/lib/imq/imqservlet.jar` (Solaris)

   `/opt/sun/mq/share/lib/imqservlet.jar` (Linux)

   `IMQ_HOME/lib/imqservlet.jar` (Windows)

6. In the Servlet args field, enter required and optional arguments, as shown in Table C-4.

Table C-4 Servlet Arguments for Deploying HTTPS Tunnel Servlet Jar File

Argument	Default Value	Required?
`keystoreLocation`	none	Yes
`keystorePassword`	none	Yes
`servletHost`	all hosts	No
`servletPort`	7674	No

Separate the arguments with commas. For example:

`keystoreLocation=`*keystore_location*`,keystorePassword=`*keystore_password*`,`
`servletPort=`*portnumber*

The `servletHost` and `servletPort` argument apply only to communication between the Web Server and broker, and are set only if the default values are problematic. However, in that case, you also must set the broker configuration properties accordingly (see Table C-3 on page 385). For example:

```
imq.httpsjms.https.servletPort
```

Configuring a Servlet Virtual Path (Servlet URL)

➤ **To Configure a Virtual Path (servlet URL) for a Tunnel Servlet**

1. Select the Servlets tab.
2. Choose Configure Servlet Virtual Path Translation.
3. Set the Virtual Path field.

   The Virtual Path is the */contextRoot*/`tunnel` portion of the tunnel servlet URL:

   `https://`*hostName*`:`*port*`/`*contextRoot*`/tunnel`

   For example, if you set the *contextRoot* to `imq`, the Virtual Path field would be:

   `/imq/tunnel`

4. Set the Servlet Name field to the same value as in step 3 in "Adding a Servlet" on page 389.

Loading a Servlet

➤ **To Load the Tunnel Servlet at Web Server Startup**

1. Select the Servlets tab.
2. Choose Configure Global Attributes.
3. In the Startup Servlets field, enter the same servlet name value as in step 3 in "Adding a Servlet" on page 389.

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.
3. Use the Log client accesses control to disable logging

Deploying as a WAR File

The instructions below refer to deployment on Sun Java System Web Server 6.0 Service Pack 2. 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 `jsse.jar`, `jnet.jar`, and `jcert.jar` to `IWS60_TOPDIR/bin/https/jre/lib/ext`.

Also, before deploying the HTTPS tunnel servlet, you must modify its deployment descriptor to point to the location where you have placed the keystore file and to specify the keystore password.

➤ **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 keystore password, you can use file system permissions to restrict access to the `imqhttps.war` file.)

➤ **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 392.)
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*`:`*port*`/`*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:

`https://`*hostName*`:`*port*`/imq/tunnel`

Clients can now use this URL to connect to the message service using a secure HTTPS connection.

Example 4: Deploying the HTTPS Tunnel Servlet on Sun Java System Application Server 7.0

This section describes how you deploy the HTTPS tunnel servlet as a WAR file on the Sun Java System Application Server 7.0.

Two steps are required:

- deploy the HTTPS tunnel servlet using the Application Server 7.0 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 7.0 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`).

   The location of the `imqhttps.war` file depends on your operating system (see Appendix A, "Operating System-Specific Locations of Message Queue Data")

4. Click OK.
5. On the next screen, set the value for the Context Root text field.

   The Context Root field value is the */contextRoot* portion of the tunnel servlet URL:

   `https://`*hostName*`:`*port*`/`*contextRoot*`/tunnel`

   For example, you could set the Context Root field to:

   ```
   /imq
   ```

6. Click OK.

   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/SUNWappserver7/domains/domain1/server1/applications/
   j2ee-modules/imqhttps_1
   ```

The servlet is now available at the following address:

`https://`*hostName*`:`*port*`/`*contextRoot*`/tunnel`

Clients can now use this URL to connect to the message service using an HTTPS connection.

Modifying the server.policy file

The Application Server 7.0 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/SUNWappserver7/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/SUNWappserver7/domains/domain1/server1/
            applications/j2ee-modules/imqhttps_1/-"
{
    permission java.net.SocketPermission "*",
            "connect,accept,resolve";
};
```

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.

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/816-6873) for a complete list of terms that are used in the Sun Java System product suite.

Index

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X